Ф. Препарата, М. Шеймос

# ВЫЧИСЛИТЕЛЬНАЯ ГЕОМЕТРИЯ: Введение

Издательство «Мир»

# COMPUTATIONAL GEOMETRY
## An introduction

Franco P. Preparata
Michael Ian Shamos

Springer-Verlag
New York Berlin Heidelberg Tokyo

Ф. Препарата, М. Шеймос

# ВЫЧИСЛИТЕЛЬНАЯ ГЕОМЕТРИЯ:
## Введение

Перевод с английского
С. А. Вичеса, М. М. Комарова
под редакцией
Ю. М. Баяковского

Москва «Мир» 1989

ББК 22.19
П72
УДК 681.3 513

**Препарата Ф., Шеймос М.**

П72 Вычислительная геометрия: Введение: Пер. с англ. — М.: Мир, 1989. — 478 с.

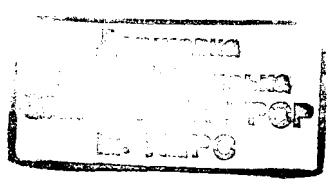
ISBN 5-03-001041-6

Монография известных американских специалистов, содержащая основы разработки и анализа алгоритмов вычислительной геометрии. Изложение основано на детальном рассмотрении конкретных задач и алгоритмов их решения. Особое внимание уделено способам описания алгоритмов на упрощенном Алголе.

Для математиков-прикладников, аспирантов и студентов вузов как учебное пособие по машинному проектированию, машинной графике, распознаванию образов.

**П $\frac{1702070000-127}{041(01)-89}$ 28—89** **ББК 22.19**

*Редакция литературы по математическим наукам*



**ISBN 5-03-001041-6 (русск.)**
**ISBN 0-387-96131-3 (англ.)**



# Предисловие редактора перевода

В 1982 г. в издательстве «Мир» вышла книга А. Фокса и М. Пратта «Вычислительная геометрия. Применение в проектировании и прозводстве», в которой рассматриваются проблемы геометрического моделирования. И вот спустя 6 лет в книге по существу с таким же названием речь идет совсем о другом, а именно об оценке сложности геометрических вычислений, об анализе и построении эффективных комбинаторных алгоритмов для решения геометрических задач. Парадокс объясняется просто: как одно, так и другое научное направление родилось совсем недавно, около 15 лет назад, их становление и интенсивное развитие протекали одновременно, оба направления имеют глубокие геометрические корни и самым непосредственным образом связаны с применением ЭВМ. К настоящему моменту предмет исследований в каждом из направлений обрел четкие очертания и в специальной литературе за одним из них утвердилось название — «Геометрическое моделирование», а за другим, которому и посвящена данная книга, — «Вычислительная геометрия».

Книга Ф. Препараты и М. Шеймоса — первая в мировой литературе монография, в которой систематизированы результаты, рассеянные по многочисленным журнальным статьям и отчетам. Эту книгу долго ждали, ссылки на нее появились значительно раньше, чем она вышла в свет.

Вычислительная геометрия имеет широкий круг применений. Это робототехника, распознавание образов, проектирование СБИС, базы данных, машинная графика, раскрой материалов и т. д. Популярность вычислительной геометрии непосредственно связана с визуализацией вычислений и, как следствие, с удобством геометрической интерпретации необязательно геометрических объектов.

Читатель увидит, что вычислительная геометрия опирается на многие разделы как классической, так и современной мате-

матики. Богатство математического аппарата и широта применения вычислительной геометрии серьезно осложнили работу над переводом. Определенные трудности создавала также принятая в американской литературе образность терминов, противоречащая сложившейся традиции в русской научно-технической литературе.

При переводе были исправлены мелкие ошибки (но нет гарантии, что не допущены другие). Часть таких ошибок любезно указал Ф. Препарата. Он же внес отдельные изменения в текст книги, сделав ее более современной. Мы благодарны проф. Ф. Препарате за внимание, с которым он отнесся к русскому изданию.

Большую помощь при переводе и редактировании книги оказали большие энтузиасты вычислительной геометрии Н. М. Корнеенко и Б. М. Чудинович. Их советы и замечания безусловно содействовали улучшению качества книги. К сожалению, в нашей стране людей, активно работающих в новой, актуальной области весьма и весьма мало. Поэтому можно присоединиться к Ф. Препарате, выразившему надежду, что появление книги на русском языке привлечет внимание молодых математиков и информатиков к новой и быстро развивающейся области научного поиска и соревнования.

Перевод книги выполнен С. А. Вичесом (гл. 1, 2, 7, 8) и М. М. Комаровым (гл. 3, 4, 5, 6).

Апрель 1988 г. *Ю. Баяковский*

# Предисловие автора к русскому изданию

Прошло 3 года с момента выхода книги на английском языке, и теперь я с удовольствием приветствую ее перевод на русский язык. В течение этого короткого периода времени стремительные темпы развития вычислительной геометрии возросли еще больше. Ряд задач, которые оставались нерешенными почти десятилетие, были либо полностью решены, либо на пути к их решению были получены блестящие результаты. Поэтому я с радостью воспользовался предоставленной мне возможностью внести изменения в те предложения и абзацы книги, которые в связи с самыми последними достижениями несколько устарели. Сегодня исследования в вычислительной геометрии продолжаются так же энергично, как и прежде. Я надеюсь, что эта книга сможет способствовать увеличению числа наших коллег в Советском Союзе, присоединившихся к этому благородному и увлекательному состязанию.

Февраль 1988 г. *Ф. Препарата*

*Паоле и Клаудии,*
*а также Юлии*

# Предисловие

Цель этой книги — единообразное изложение богатых результатов, появившихся в основном за последнее десятилетие в области вычислительной геометрии. Эта молодая дисциплина, названная своим именем в его современном значении одним из нас — М. Шеймосом, привлекла к себе огромный интерес и выросла из набора разрозненных результатов в зрелую область знаний. Достигнутая зрелость, однако, не помешала вычислительной геометрии оставаться постоянным источником научных и прикладных задач.

По мере того, как исследования достигали действительно высокого уровня, с появлением мощных методов и развитием плодотворного взаимодействия с комбинаторной и алгебраической геометрией, возрастала потребность в методически организованном изложении накопленных результатов. Эта потребность ощущалась как в учебной аудитории, где экспериментальные спецкурсы велись на материале журнальных статей и заметок, так и в профессиональной среде, где некоторые прикладные области, такие как автоматизация проектирования, машинная графика, робототехника и т. д., были готовы для восприятия этих результатов.

Данная книга — попытка удовлетворить эту потребность. Задуманная, в основном, как учебное пособие для студентов младших курсов, она ориентирована также на специалистов в вышеупомянутых прикладных областях. Однако необходимо отметить, что эта книга не просто каталог готовых к употреблению методов, хотя некоторые алгоритмы весьма практичны. Наша точка зрения скорее восходит к дисциплине, известной как построение и анализ алгоритмов (или «алгоритмика»), нацеленной на определение трудоемкости решения отдельных задач. Наши исследования в основном направлены на изучение поведения алгоритмов в худшем случае; кроме того, результаты анализа



полностью проявляются лишь для достаточно больших задач (асимптотический анализ). Оба эти момента нельзя упускать из виду при выборе алгоритма, ибо метод, наиболее пригодный для задач малой размерности, не обязательно будет асимптотически оптимальным, а алгоритм не оптимальный в худшем случае, может превзойти другие при оценке его среднего поведения.

Мы попытались дать достаточно детальную панораму непрерывной ветви вычислительной геометрии, а не ее «дискретизированного» варианта. Однако главной целью было систематическое изложение предмета, а не педантичный обзор. Мы пытались частично восполнить этот недостаток в разделах «Замечания и комментарии» в конце каждой главы. Приносим свои извенения многим авторам, чьи работы не были описаны или хотя бы упомянуты.

Начальным ядром книги была докторская диссертация М. Шеймоса. Когда в 1981 г. Р. Грэхем предложил одному из нас (Ф. Препарате) превратить эту исходную рукопись в учебник, никто из нас не представлял себе реальных усилий, которые пришлось затратить впоследствии. К счастью, многие наши коллеги и друзья щедро помогли выполнению этой задачи, терпеливо читая поступающие варианты, предлагая улучшения и вылавливая ошибки. Мы весьма признательны им; и мы рады искренне поблагодарить (в алфавитном порядке): Бентли Дж., Бхаттачарья Б., Ван-Леювена Дж., Вуда Д., Дайера М., Добкина Д., Зайделя Р., Каллаи М., Ли Д. Т., Липски-мл. У., О'Рурка Дж., Перлиса А., Рафски Л., Туссена Г., Хоуи Дэна, Чазелле Б., Шульца М., Эйвиса Д., Эйзенстата С. Мы также благодарны за частичную поддержку Национальному научному фонду (субсидия MCS 81-05552 для работы Ф. Препараты), управлению военно-морских исследований, корпорации ИБМ, Университету Карнеги — Меллона и Лабораториям Белла (за работу М. Шеймоса) и руководству издательства «Шпрингер» за энтузиазм, проявленный при сотрудничестве.

Наконец, мы благодарим наших жен, соответственно Розу-Марию и Юлию, за их терпение и поддержку во время тех долгих часов, которые привели к окончательному варианту книги.

Май 1985 г.

*Ф. Препарата*
*М. Шеймос*

ГЛАВА 1

# Введение

## 1.1. Исторический обзор

Египетская и греческая геометрия были шедевром прикладной математики. Исходным импульсом для геометрических задач была нужда в точном и справедливом земельном налогообложении, а также в возведении зданий. Как это часто бывает, созданная математика обладала общностью и значением, которые далеко превосходили породившие ее проблемы фараонова департамента налогов, ибо геометрия — это сердцевина математического мышления. Это область, дающая простор интуиции, а новые открытия в ней вполне доступны для неспециалистов.

Принято считать, что главным вкладом Евклида в геометрию является его изложение аксиоматического метода доказательства — утверждение, которое мы не станем оспаривать. Однако для нашего обсуждения важнее изобретение *евклидова построения* — схемы, состоящей из алгоритма *и его доказательства*, сплетенных в весьма стилизованном формате. Это евклидово построение удовлетворяет всем требованиям, предъявляемым к алгоритму: оно компактно, корректно и действенно. После Евклида геометрия продолжала процветать, в то время как анализ алгоритмов, к сожалению, попал в 2000-летнюю полосу застоя. Это обстоятельство частично может быть объяснено успехом метода *приведения к абсурду*, который упрощает математикам доказательство существования некоего объекта путем построения этого доказательства от противного вместо того, чтобы дать точную конструкцию для его получения (алгоритм).

Евклидово построение замечательно по разным причинам; например, оно определяет набор допустимых инструментов (линейка и циркуль) и множество допустимых операций (примитивов), которые можно выполнить с их помощью. Древних чрезвычайно интересовала полнота системы евклидовых примитивов при условии конечности числа операций. В частности,



их интересовало, допускает ли такая система все мыслимые геометрические построения (например, трисекцию угла). В современной формулировке тот же вопрос ставит информатика: достаточно ли евклидовых примитивов для реализации всех геометрических «вычислений»? При попытке ответить на этот вопрос рассматривались многие альтернативные модели вычислений, получаемые путем изменения как набора примитивов, так и самих инструментов. Архимед предложил (корректную) конструкцию для трисекции угла в 60 градусов со следующим добавлением к множеству примитивов: даны два круга $A$ и $B$ и точка $P$; мы можем выбрать отрезок $MN$ прямой и разместить его так, чтобы прямая прошла через $P$, причем точка $M$ оказалась бы на границе $A$, а $N$ — на границе $B$. В ряде случаев изучался ограниченный набор инструментов, например только циркуль. Такие идеи выглядят почти предвосхищением методов теории автоматов, которая проверяет мощность вычислительных моделей при разных ограничениях. Увы, доказательство неполноты евклидовых средств должно было дожидаться развития алгебры.

Влияние «*Начал*» Евклида было столь фундаментальным, что никаких других формулировок геометрии не было предложено вплоть до Декарта. Введение последним координат позволило выразить геометрические задачи алгебраически, вымостив путь к изучению высших плоских кривых и ньютоновскому анализу. Координаты позволили резко повысить вычислительные возможности, соединив две великие области математики и дав начало возрождению конструктивистского мышления. Теперь появилась возможность получать новые геометрические объекты, решая связанные с ними алгебраические уравнения. И вскоре снова возникли вопросы вычислимости. Так, Гаусс, уже вооруженный алгебраическими средствами, вновь вернулся к задаче о том, какие из правильных многоугольников с простым числом сторон можно построить, используя евклидовы инструменты, и полностью решил ее. При этом стала ясной близкая связь между построениями с помощью циркуля и линейки, расширениями поля [1]) и алгебраическими уравнениями. В своей докторской диссертации Гаусс показал, что у любого алгебраического уравнения существует по крайней мере один корень (основная теорема алгебры). Абель в 1828 г. занялся решением той же задачи на *ограниченной модели вычислений*. Он задался вопросом, можно ли найти корень любого алгебраического уравнения, воспользовавшись только арифметическими операциями и извлечением корней $n$-й степени, и доказал, что этого сделать нельзя. Хотя было известно, что все рациональные числа действительны, по-

[1]) Имеется в виду поле действительных чисел. — *Прим. перев.*

следний ответ показал, что не все действительные числа рациональны. Вскоре после этого Абель описал те алгебраические уравнения, которые *могут* быть решены в радикалах, и это дало ему возможность обсуждать осуществимость специфически геометрических задач, таких как трисекция угла.

### 1.1.1. Представление о сложности в классической геометрии

Евклидовы построения для любых задач, кроме самых тривиальных, весьма сложны, ибо в них допустимы только рудиментарные примитивы. Весьма частым развлечением геометров после Евклида было улучшение его построений так, чтобы они могли выполняться за меньшее число «операций». Однако задача определения количественной меры сложности построения не была сформулирована вплоть до двадцатого столетия. В 1902 г. Эмиль Лемуан ввел науку *геометрографию*, систематизировав евклидовы примитивы следующим образом [Lemoin (1902)]:

1. Поставить одну ножку циркуля в данную точку.
2. Поставить одну ножку циркуля на данную прямую.
3. Провести окружность.
4. Совместить ребро линейки с данной точкой.
5. Провести линию.

Общее число таких операций, выполняемых в процессе построения, называется его *простотой*, хотя Лемуан полагал, что, возможно, более уместен термин «мера сложности». Это определение близко связано с нашей современной идеей *временно́й сложности* алгоритма, хотя в работе Лемуана нет функциональной связи между размером исходной информации (числом заданных точек и линий) для геометрического построения и его простотой. Безусловно, интерес Лемуана был направлен на улучшение исходных евклидовых построений, а не на развитие теории сложности вычислений. И в первом он добился замечательного успеха — евклидово решение задачи о кругах Аполлония требовало 508 шагов, в то время как Лемуан сократил его менее чем до двухсот [Coolidge (1916)]. К сожалению, Лемуан не почувствовал важности доказательства, а возможно, просто не смог доказать, что определенное число операций было *необходимо* для данного построения, и, таким образом, идея о нижней оценке алгоритма ускользнула от него.

Однако Гильберт оценил важность нижних оценок. Работая с ограниченной моделью, он рассматривал только такие построения, которые можно выполнить с помощью односторонней линейки и *шкалы* — инструмента, используемого только для пометки отрезка фиксированной длины вдоль некоей линии. Не



все евклидовы построения выполнимы с помощью этого набора инструментов. Но для тех, что выполнимы, мы можем рассматривать координаты построенных точек как функцию $F$ от заданных точек. Гильберт дал необходимое и достаточное условие вычислимости $F$ с использованием ровно $n$ операций извлечения квадратного корня — это одна из наиболее ранних теорем в алгебраической теории вычислительной сложности [Hilbert (1899)].

Есть основания думать, что многие из наших современных методов анализа алгоритмов были предвосхищены геометрами минувших столетий. В 1672 г. Джордж Мор показал, что любое построение, осуществимое с помощью линейки и циркуля, можно выполнить только циркулем в том случае, когда исходные и искомые объекты определяются точками. (Так, хотя прямую линию нельзя нарисовать одним циркулем, две точки на этой линии можно определить с помощью пересечения двух окружностей.) В доказательстве Мора примечательно *моделирование*, с помощью которого он показывает, что любую операцию, в которой участвует линейка, можно заменить конечным числом операций циркуля. Можно ли тут поставить вопрос о близкой связи с теорией автоматов? В этом же русле лежит и результат о том, что линейка, используемая в любом построении, может иметь произвольную положительную длину, и, даже будучи маленькой, все же способна моделировать линейку какой угодно длины.

Лемуан и другие занимались временно́й сложностью евклидовых построений, и тем не менее также поднимался вопрос об объеме *пространства*, необходимого для этих построений. Хотя использовавшаяся мера пространства (а именно площадь на плоскости, необходимая для реализации построения) не совпадает с нашим современным определением в смысле количества памяти, занимаемой алгоритмом, она была замечательно близка к нему и весьма естественна. Использованная площадь зависит, вообще говоря, от площади выпуклой оболочки заданных геометрических мест точек и от размера искомого результата, а также от размера любых промежуточных геометрических мест, которые необходимо сформировать при построении [Eves (1972)]. Мы подчеркиваем здесь, что геометрии не чужды представления о времени и памяти.

Когда Галуа продемонстрировал невозможность реализации некоторых евклидовых построений, то стало ясно, что *точная* трисекция любого угла неосуществима, но нет запрета на выполнимость какого-нибудь приближенного построения. Фактически асимптотически сходящиеся процедуры для квадратуры круга и удвоения куба были известны еще древним грекам [Heath (1921)]. История приближенных алгоритмов, безусловно, является долгой.

### 1.1.2. Теория выпуклых множеств, метрическая и комбинаторная геометрии

Геометрия XIX века развивалась по многим направлениям. Одно из них, провозглашенное Клейном, включало в себя всестороннее изучение поведения геометрических объектов при различных преобразованиях, а проективная геометрия была его важным ответвлением (см. разд. 1.3.2). Хотя исследование конечных проективных плоскостей приводит к чарующим вопросам и в комбинаторике, и в теории дискретных алгоритмов, этот аспект геометрии не будет предметом данной книги.

Развитие вещественного анализа возымело решающее влияние на геометрию, приведя к формальным абстракциям концепций, которые ранее были лишь интуитивными. Две такие разработки — метрическая геометрия и теория выпуклости — дали главные математические инструменты, которые помогают при построении быстрых алгоритмов.

Расстояние — важное понятие в геометрии. *Метрика* — его обобщение — дала возможность распространить геометрические концепции и интуицию на анализ, где идея «расстояния» между функциями привела к появлению функциональных пространств и других мощных конструкций. К сожалению, многое из созданного уводило в неконструктивизм. Функциональные пространства по своей природе не вычислимые объекты.

Важность теории выпуклости заключается в том, что она аналитически работает с *глобальными* свойствами объектов и позволяет нам решать экстремальные задачи. К сожалению, многие вопросы выпуклости имеют громоздкие алгебраические формулировки и субъективно подталкивают к неконструктивным методам.

Комбинаторная геометрия гораздо ближе по духу к нашей цели — алгоритмической геометрии. Она основана на описании геометрических объектов в терминах свойств *конечных* подмножеств. Например, некоторое множество выпукло тогда и только тогда, когда отрезок, определяемый любой *парой* его точек, лежит полностью в этом множестве. Неадекватность комбинаторной геометрии для нас заключается в том, что для большинства интересующих нас множеств число их конечных подмножеств бесконечно, а это препятствует их алгоритмической обработке. Работа последних лет в области геометрических алгоритмов направлена на избавление от этих недостатков и развитие математики, ведущей к созданию хороших алгоритмов.

### 1.1.3. Смежные направления

Геометрические по своему характеру алгоритмы разрабатывались в нескольких разных контекстах, и термин «вычислительная геометрия» уже употреблялся по крайней мере в двух других значениях. Здесь мы попытаемся рассмотреть эти смежные направления в соответствующей перспективе и сопоставить их с современной трактовкой:



1. Геометрическое моделирование средствами сплайновых кривых и поверхностей — предмет, который по духу ближе к численному анализу, чем к геометрии — подробно разрабатывалось Безье, Форрестом и Ризенфельдом. Надо отметить, что Форрест называл свою дисциплину «Вычислительной геометрией» [Bézier (1972), Forrest (1971), Riesenfeld (1973)][1]).

2. В элегантной и захватывающей книге, названной «*Персептроны*» (подзаголовком которой является также «Вычислительная геометрия»), Минский и Пейперт [Minsky, Papert (1969)] изучают сложность предикатов, которые распознают определенные геометрические свойства, такие как выпуклость. Цель их работы заключалась в получении вывода о возможности использования больших сетчаток, состоящих из простых схем, для задач распознавания образов. Их теория полностью самостоятельна и не укладывается в алгоритмические рамки нашей книги.

3. В графическом программном обеспечении и геометрических редакторах широко используются многие алгоритмы, представленные в данной книге. Однако в данном случае в фокусе оказываются детали реализации и интерфейс с пользователем, а отнюдь не анализ алгоритмов. К этому же классу относятся: программное обеспечение станков с ЧПУ, программы для графопостроителей, картографические системы и программное обеспечение для архитектурного дизайна и строительства.

4. Наконец, словосочетание «вычислительная геометрия» для некоторых может означать деятельность в области доказательства геометрических теорем с помощью ЭВМ. Хотя это и увлекательное исследование, но оно дает намного больше информации о наших эвристиках, служащих для доказательства теорем и понимания процедур доказательства, чем данных о геометрии *как таковой*, и поэтому мы здесь этим заниматься не будем.

### 1.1.4. Предыстория вычислительной геометрии

У колыбели дисциплины, известной ныне как вычислительная геометрия, стояло большое число приложений, ибо они порождают свойственные им геометрические задачи, для которых должны создаваться эффективные алгоритмы. К числу таких задач относятся: задача коммивояжера на евклидовой метрике, построение минимального остовного дерева, удаление

[1]) См. также [Фокс, Пратт (1982)]. — *Прим. перев.*

невидимых линий, линейное программирование и многие другие. Чтобы убедительно продемонстрировать широту приложений вычислительной геометрии, мы отложим представление основного материала по этим задачам до той поры, пока они не встретятся в тексте.

Алгоритмические исследования этих и других задач появились в научной литературе в текущем столетии, причем их интенсивность увеличилась в последние два десятилетия. Однако систематическое изучение геометрических алгоритмов было предпринято только совсем недавно и эта дисциплина, окрещенная «вычислительной геометрией» в статье М. Шеймоса [Shamos (1975a)], привлекает к себе все большее число исследователей.

Перспективы и методология вычислительной геометрии проясняется, как мы надеемся, при детальном изучении конкретных задач, представленных в данной книге. Одно из основных свойств этой дисциплины заключается в осознании того, что классические характеристики геометрических объектов часто не влияют на проектирование эффективных алгоритмов. Чтобы преодолеть этот недостаток, необходимо найти полезные понятия и установить их свойства, способствующие эффективности вычислений. Короче говоря, вычислительная геометрия должна преобразовать — там, где это необходимо, — классическую дисциплину в ее вычислительную форму.

## 1.2. Алгоритмические основы

В течение последних пятнадцати лет анализ и построение вычислительных алгоритмов остаются одной из самых продуктивных областей информатики. Фундаментальные труды Кнута [Knuth (1968; 1973)] и Ахо, Хопкрофта, Ульмана [Aho, Hopcroft, Ullman (1974)] позволили упорядочить и систематизировать богатую коллекцию отдельных результатов; в них сформулированы основные концептуальные модели и установлена методология, ставшая стандартом в этой области. Последующие работы [Reingold, Nievergelt, Deo (1977), Wirth (1976)] укрепили фундамент этой теории.

Поэтому детальный обзор материала этих превосходных книг, с которыми читатель, наверное, основательно знаком, останется за рамками нашей работы. Однако целесообразно — по крайней мере с точки зрения терминологии — привести краткий обзор основных компонент языка, который будет применяться для описания вычислительной геометрии. Эти компоненты суть алгоритмы и структуры данных. Алгоритмы — это программы, исполнимые на удобной абстракции реальных «фон-неймановских» ЭВМ; структуры данных — это способы организации информации, которые в совокупности с алгоритмами приводят к эффективному и элегантному решению вычислительных задач.



### 1.2.1. Алгоритмы: их запись и оценка эффективности

Алгоритмы формулируют по отношению к конкретной модели вычислений; как мы увидим в разд. 1.4, эта модель представляет собой удобную абстракцию реальной машины, используемой для исполнения программ. Однако, как было подчеркнуто в работе [Aho, Hopcroft, Ullman (1974)], нет ни нужды, ни желания записывать алгоритмы машинным кодом. Напротив, в интересах ясности, изобразительной эффективности и краткости мы будем, как правило[1]), использовать язык высокого уровня, ставший стандартным в литературе на данную тему — *псевдоалгол*. Псевдоалгол — это неформализованная и гибкая версия языка Алгол; он достаточно строг в своих управляющих структурах, но весьма волен в формате других своих операторов, где употребление общепринятых математических обозначений свободно чередуется с использованием выражений естественного языка. Разумеется, программа на псевдоалголе может быть рутинно превращена в строгую программу на языке уровня.

Следуя за работой Ахо, Хопкрофта, Ульмана, мы кратко проиллюстрируем конструкции псевдоалгола. Формальные определения типов данных опущены, а тип любой переменной обычно становится ясным из контекста. Кроме того, не выбирается специальный формат для выражений и условий.

Любая программа, обычно называемая **процедурой**, имеет следующий формат:

```
procedure имя (параметры) оператор.
```

Оператор можно заменить (согласно терминологии порождающих грамматик) цепочкой из двух или более операторов, заключая ее в этом случае в «операторные скобки» «begin ... end»:

```
begin оператор;
        .
        .
        .
      оператор
end.
```

В свою очередь смысл любого оператора можно раскрыть или предложением на естественном языке, или одним из нижеследующих особых более формальных способов.

[1]) Иногда алгоритмы могут быть представлены в сугубо описательной форме.

1. Присваивание:

| переменная := источник |
|---|

Термин «источник» определяет вычисление, порождающее значение заданной переменной; вычисление это, вообще говоря, некоторое *выражение* над множеством переменных. (Некоторые из этих переменных могут в свою очередь выражаться через «функции»; **функция** — это особая форма программы, как показано ниже.)

2. Условие:

| **if** условие **then** оператор (**else** оператор) |
|---|

где составляющая **else** необязательна.

3. Цикл. Он представим в одном из трех форматов:

| 3а. **for** переменная := значение **until** значение **do** оператор<br>3б. **while** условие **do** оператор<br>3в. **repeat** оператор **until** условие |
|---|

Конструкции **while** и **repeat** различаются, поскольку в цикле типа «**while**» условие проверяется *перед* выполнением оператора, в то время как в цикле типа «**repeat**» имеет место обратное.

4. Возврат:

| **return** выражение |
|---|

Оператор этого вида должен появиться в программе функционального типа, которая имеет формат

| **function** имя (параметры) оператор. |
|---|

Выражение, стоящее аргументом в операторе **return**, становится источником в операторе присваивания, как указано в п. 1.

Часто алгоритмы на псевдоалголе содержат комментарии, предназначенные для пояснения происходящего. В данной книге типовой формат комментария будет следующим: (* предложение на естественном языке *).

Время, затраченное на вычисление при выполнении алгоритма, представляет собой сумму времен отдельных исполненных операторов (см. также разд. 1.4). Как упоминалось выше, программу, написанную на псевдоалголе, можно превратить прямым (хотя и утомительным) путем в программу на машинном коде заданной ЭВМ. Это в принципе дает метод оценки времени выполнения указанной программы. Однако такой подход не только скучен, но также и мало вразумителен, ибо он ориентирован на конкретную ЭВМ, тогда как нас, по сути, интересует более общая функциональная зависимость времени счета от размеров задачи (т. е. то, как быстро время счета растет с ростом размеров задачи). В области анализа и построения алгоритмов принято выражать время выполнения, так же как и любую другую меру эффективности, с точностью до мультипликативной константы. Это обычно делается путем подсчета лишь определенных «ключевых операций», выполняемых алгоритмом (что легко осуществить, анализируя версию этого алгоритма, записанную на языке высокого уровня). Такой подход абсолютно правомерен при определении нижних оценок времени выполнения, поскольку любые неучтенные операции могут лишь увеличить их; однако при работе с верхними оценками мы обязаны убедиться в том, что вклад выбранных ключевых операций в сумме отличается не более чем в константу раз от вклада всех операций, выполненных алгоритмом. Кнут популяризовал аппарат обозначений, прекрасно отличающих верхние оценки от нижних, который мы и заимствуем [Knuth (1976)]:



$O(f(N))$ служит для обозначения множества всех функций $g(N)$ таких, что существуют положительные константы $C$ и $N_0$, для которых $|g(N)| \leqslant Cf(N)$ при всех $N \geqslant N_0$.

$\Omega(f(N))$ служит для обозначения множества всех функций $g(N)$ таких, что существуют положительные константы $C$ и $N_0$, для которых $g(N) \geqslant Cf(N)$ при всех $N \geqslant N_0$.

$\theta(f(N))$ служит для обозначения множества всех функций $g(N)$ таких, что существуют положительные константы $C_1$, $C_2$ и $N_0$, для которых $C_1 f(N) \leqslant g(N) \leqslant C_2 f(N)$ при всех $N \geqslant N_0$.

$o(f(N))$ служит для обозначения множества всех функций $g(N)$ таких, что *для всех положительных констант* $C$ существует некое $N_0$, при котором $g(N) \leqslant Cf(N)$ для всех $N \geqslant N_0$ (или, что то же самое, $\lim\limits_{N \to \infty} g(N)/f(N) = 0$).

Итак, $O(f(N))$ используется для указания функций, которые *не более* чем в постоянное число раз превосходят $f(N)$, этот способ нужен для описания верхних оценок; напротив, $\Omega(f(N))$ используется для указания функций, которые *не менее* чем в постоянное число раз превосходят $f(N)$ — аналогичный способ, но для нижних оценок. Наконец, $\theta(f(N))$ используется для указания функций того же порядка, что и $f(N)$; этот способ нужен для «оптимальных» алгоритмов.

Предыдущее обсуждение было сосредоточено на времени счета алгоритма. Другая важная мера его эффективности — это

пространство, обычно совпадающее с объемом памяти, использованной алгоритмом. Несомненно, *пространственная* и *временна́я* сложности как функции от размеров задачи являются двумя фундаментальными оценками эффективности при анализе алгоритмов.

Главная цель данной книги, по сути, заключается в представлении алгоритмов для геометрических задач и оценке их *сложности для худшего случая*. Сложность для худшего случая — это максимальная мера эффективности данного алгоритма для всех задач данного размера, и ее нельзя путать со *сложностью в среднем* (или *ожидаемой*), которая отличается тем, что дает оценку *наблюдаемого* поведения этого алгоритма. К сожалению, анализ поведения в среднем значительно более сложная вещь, чем анализ худшего случая, по двум причинам: во-первых, существенные математические трудности возникают, даже если удачно выбрано исходное распределение; во-вторых, часто с трудом достигается согласие в том, что именно выбранное распределение является реальной моделью изучаемой ситуации. Вот почему преобладающее большинство результатов связано с анализом худших случаев; соответственно и в данной книге будут лишь изредка обсуждаться результаты поведения в среднем.

Еще один важный момент, который следует отметить, связан с тем, что обозначения «порядка» скрывают мультипликативные константы. Следовательно, оценка сложности достигает своей законной силы только при весьма больших размерах задачи; именно поэтому такой подход называется *асимптотическим анализом*. Следовательно, вполне вероятно, — и действительно часто бывает так, — что при малых размерах задачи наиболее пригоден алгоритм, который асимптотически не оптимален. Это *предостережение* никак нельзя игнорировать при выборе алгоритма для конкретного приложения.

### 1.2.2. Несколько замечаний об общих алгоритмических методах

Эффективные алгоритмы для геометрических задач часто конструируются при помощи общих методов теории алгоритмов, таких как «разделяй и властвуй», балансировка, рекурсия и динамическое программирование. Превосходное обсуждение этих методов содержится в ставших теперь классическими книгах по анализу и проектированию алгоритмов (см., например, [Aho, Hopcroft, Ullman (1974)]), и было бы излишеством повторять его здесь.

Однако существует метод, который подсказан — исключительно и естественно — природой некоторых геометрических задач. Этот метод называется *заметанием*, и наиболее часто встречаются примеры *плоского заметания* (в двух измерениях) и *пространственного заметания* (в трех измерениях). Сейчас опишем главные черты плоского заметания; его можно весьма просто обобщить на случай трех измерений.



Для определенности иллюстрируем этот метод на примере конкретной задачи (обсуждаемой во всех подробностях в разд. 7.2.3): дано множество отрезков на плоскости, надо найти все их пересечения. Возьмем прямую $l$ (предположим без ограничения общности, что она вертикальная), которая разбивает данную плоскость на левую и правую полуплоскости. Допустим, что каждая из этих полуплоскостей содержит концы заданных отрезков. Очевидно, что решение нашей задачи есть объединение решений для каждой из двух полуплоскостей; поэтому, если предположить, что у нас уже есть множество точек пересечения слева от $l$, то на это множество не будут влиять отрезки, расположенные справа от $l$. Теперь заметим, что пересечение (искомое) может произойти только между такими двумя отрезками, чьи пересечения с некоторой вертикалью смежны; итак, если построить все вертикальные сечения отрезков заданного множества, то наверняка будут обнаружены и все искомые пересечения. Однако этой (неразрешимой) задачи построения (континуально) бесконечного множества всех секущих вертикалей можно избежать, если заметить, что плоскость разбивается на вертикальные полосы, ограниченные или концами отрезков, или точками их пересечения, и такие, что вертикальный порядок точек пересечения вдоль любой вертикальной секущей в такой полосе постоянен. Таким образом, все, что надо сделать, состоит в скачке с левого края такой полосы на ее правый край с обновлением порядка точек пересечения вертикали и поиском новых пересечений среди «соседних» в этом порядке отрезков.

Предыдущее обсуждение намечает все существенные черты метода плоского заметания. Есть вертикаль, которая заметает плоскость слева направо, останавливаясь в особых точках, именуемых «точками событий». Пересечение заметающей прямой с входными данными задачи содержит всю *полезную* для продолжения поиска информацию. Итак, мы имеем две основные структуры:

1. *Список точек событий* — последовательность абсцисс, упорядоченных слева направо и определяемых позициями остановок заметающей прямой. Отметим, что список точек событий не требуется обязательно извлекать полностью из исходных данных, вместо этого его можно динамически обновлять при работе алгоритма плоского заметания. В разных приложениях для этого могут понадобиться различные структуры данных.

2. *Статус заметающей прямой* — адекватное описание пересечения этой заметающей прямой с заметаемой геометрической

структурой. «Адекватное» означает, что это пересечение содержит информацию, полезную для данного приложения. Статус заметающей прямой обновляется в каждой точке событий, а подходящая для него структура данных должна выбираться применительно к каждому случаю.

Примеры алгоритмов плоского заметания можно найти в разд. 2.2.2.

### 1.2.3. Структуры данных

Геометрические алгоритмы включают в себя манипулирование с объектами, которые не обрабатываются на уровне машинного языка. Поэтому пользователь должен описать эти сложные объекты посредством более простых типов данных, непосредственно представимых в машине. Такие описания принято называть *структурами данных*.

Самые распространенные сложные объекты, встречающиеся при построении геометрических алгоритмов, — это множества и последовательности (упорядоченные множества). Структуры данных, особенно удобные для этих сложных комбинаторных объектов, хорошо описаны в стандартной литературе по алгоритмам, к которой и отсылается читатель [Aho, Hopcroft, Ullman (1974); Reingold, Nievergelt, Deo (1977)]. Приведем их здесь только затем, чтобы бегло очертить классификацию этих структур данных наряду с функциональными возможностями и вычислительной эффективностью.

Пусть $S$ — некоторое множество, представленное структурой данных, и пусть $u$ — произвольный элемент некоторого универсального множества, подмножеством которого является $S$. Приведем основные операции, встречающиеся при работе со множествами:

1. ПРИНАДЛЕЖНОСТЬ ($u$, $S$).Верно, что $u \in S$? (Ответ типа ДА/НЕТ.)
2. ВСТАВИТЬ ($u$, $S$). Включить $u$ в $S$.
3. УДАЛИТЬ ($u$, $S$). Исключить $u$ из $S$.

Предположим теперь, что $\{S_1, S_2, \ldots, S_k\}$ — набор множеств (попарно непересекающихся). На этом наборе полезны следующие операции:

4. НАЙТИ ($u$). Найти такое $j$, что $u \in S_j$.
5. ОБЪЕДИНИТЬ ($S_i$, $S_j$; $S_k$). Сформировать объединение $S_i$ и $S_j$ и назвать его $S_k$.



Если универсальное множество полностью упорядочено, то очень важны следующие операции:

6. MIN ($S$). Найти наименьший элемент $S$.
7. РАСЦЕПИТЬ ($u$, $S$). Разделить $S$ на $\{S_1, S_2\}$ такие, что $S_1 = \{v: v \in S \text{ и } v \leqslant u\}$, а $S_2 = S - S_1$.
8. СЦЕПИТЬ ($S_1$, $S_2$). Предположим, что для любых $u' \in S_1$ и $u'' \in S_2$ имеем $u' \leqslant u''$; следует сформировать множество $S = S_1 \cup S_2$.

Структуры данных можно классифицировать на базе операций, которые на них допустимы (без учета их эффективности). Так, для упорядоченных множеств имеется следующая таблица:

*Таблица 1*

| Структура данных | Допустимые операции |
|---|---|
| Словарь | ПРИНАДЛЕЖНОСТЬ, ВСТАВИТЬ, УДАЛИТЬ |
| Приоритетная очередь | MIN, ВСТАВИТЬ, УДАЛИТЬ |
| Сцепляемая очередь | ВСТАВИТЬ, УДАЛИТЬ, РАСЦЕПИТЬ, СЦЕПИТЬ |

Для лучшей эффективности каждая из этих структур данных обычно реализуется как дерево двоичного поиска, сбалансированное по высоте (часто как АВЛ- или 2-3-дерево) [Aho, Hopcroft, Ullman (1974)]. При такой реализации каждая из вышеописанных операций осуществляется за время, пропорциональное логарифму числа элементов, хранимых в структуре данных; требуемая память пропорциональна мощности исходного множества.

Описанные выше структуры данных можно мысленно представить себе в виде линейного массива из элементов (*списка*) так, что вставки и удаления можно осуществить в любой позиции этого массива. Иногда для каких-то приложений вполне приемлемы некоторые более ограниченные режимы доступа, допускающие упрощение. Приведем эти структуры: *очереди* — вставки происходят с одного конца, а удаления — с другого; *стеки* — и вставки, и удаления происходят с одного конца (вершины стека). Очевидно, что все, что необходимо для управления соответственно стеком или очередью, — это один или два указателя. Для краткости мы будем пользоваться обозначениями «$\Rightarrow U$» и «$U \Rightarrow$» для указания соответственно добавления или удаления из $U$, где $U$ — это очередь или стек.

Неупорядоченные множества всегда могут обрабатываться, как упорядоченные: путем искусственного введения порядка над их элементами (например, присвоением «имен» элементам и использованием алфавитного порядка). Типовая структура данных в этой ситуации такова:

*Таблица II*

| Структура данных | Допустимые операции |
|---|---|
| Сливаемая пирамида | ВСТАВИТЬ, УДАЛИТЬ, НАЙТИ, ОБЪЕДИНИТЬ, (MIN) |

Каждую из вышеуказанных операций можно выполнить за время $O(\log N)$, где $N$ — размер множества, запасенного в этой структуре данных путем использования, как обычно, дерева, сбалансированного по высоте. Если элементы рассматриваемого множества представлены целыми числами от 1 до $N$, то более сложная реализация этой структуры данных позволяет выполнить $N$ операций над множеством размером $N$ за время $O(N \cdot A(N))$, где $A(N)$ — чрезвычайно медленно растущая функция, связанная с обратной функцией Аккермана (например, для $N \leqslant 2^{2^{16}}$, или $\sim 10^{20\,000}$, $A(N) \leqslant 5$).

Стандартные структуры данных, рассмотренные выше, широко используются в алгоритмах вычислительной геометрии. Однако природа геометрических задач привела к созданию специальных, необычных структур данных, две из которых оказались настолько общезначимыми, что целесообразно представить их в данной вводной главе. Эти структуры — дерево отрезков и реберный список с двойными связями.

### 1.2.3.1. Дерево отрезков

*Дерево отрезков*, впервые введенное Дж. Бентли [Bentley (1977)], это структура данных, созданная для работы с такими интервалами на числовой оси, концы которых принадлежат *фиксированному* множеству из $N$ абсцисс. Поскольку множество абсцисс фиксировано, то дерево отрезков представляет собой *статическую* структуру по отношению к этим абсциссам, т. е. структуру, на которой не разрешены вставки и удаления абсцисс; кроме того, эти абсциссы можно нормализовать, заменяя каждую из них ее порядковым номером при обходе их слева направо. Не теряя общности, можно считать эти абсциссы целыми числами в интервале $[1, N]$.



*Дерево отрезков* — это двоичное дерево с корнем. Для заданных целых чисел $l$ и $r$ таких, что $l < r$, дерево отрезков $T(l, r)$ строится рекурсивно следующим образом: оно состоит из корня $v$ с параметрами $B[v] = l$ и $E[v] = r$ ($B$ и $E$ мнемонически соответствуют словам «beginning» (начало) и «end» (конец)), а если $r - l > 1$, то оно состоит из левого поддерева $T(l, \lfloor (B[v] + E[v])/2 \rfloor)$ и правого поддерева $T(\lfloor (B[v] + E[v])/2 \rfloor, r)$. (Корни этих поддеревьев естественно обозначить через ЛСЫН$[v]$ и ПСЫН$[v]$ соответственно.) Параметры $B[v]$

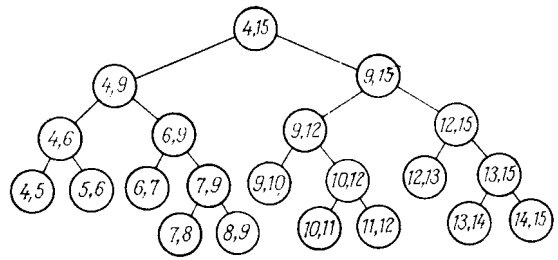



Рис. 1.1. Дерево отрезков $T(4, 15)$.

и $E[v]$ обозначают *интервал* $[B[v], E[v]] \subseteq [l, r]$, связанный с узлом $v$. Пример дерева отрезков приведен на рис. 1.1. Интервалы, принадлежащие множеству $\{[B[v], E[v]]: v$ — узел $T(l, r)\}$, называются *стандартными интервалами* дерева $T(l, r)$. Стандартные интервалы, принадлежащие листьям $T(l, r)$, называются *элементарными интервалами*[1]). Можно непосредственно убедиться, что $T(l, r)$ сбалансировано (все его листья принадлежат двум смежным уровням) и имеет глубину $\lceil \log_2 (r - l) \rceil$.

Дерево отрезков $T(l, r)$ предназначено для *динамического* запоминания тех интервалов, чьи концы принадлежат множеству $\{l, l+1, \ldots, r\}$ (т. е. допустимы операции вставки и удаления). В частности, при $r - l > 3$ произвольный интервал $[b, e]$ с целыми $b < e$ будет разбит в набор из не более чем $\lceil \log_2 (r - l) \rceil + \lfloor \log_2 (r - l) \rfloor - 2$ стандартных интервалов дерева $T(l, r)$. Фрагментация интервала $[b, e]$ полностью определяется операцией, которая заносит (вставляет) $[b, e]$ в дерево отрезков $T$, и, значит, обращением ВСТАВИТЬ$(b, e$; корень $(T))$ к следующей процедуре:

[1]) Строго говоря, интервал, связанный с $v$, это полуоткрытый интервал $[B[v], E[v])$, за исключением узлов самого правого пути в $T(l, r)$, чьи интервалы замкнуты.

```
      procedure ВСТАВИТЬ(b,e;v)
begin if (b ⩽ B[v]) and (E[v] ⩽ e) then назначить [b, e] узлу v
      else begin if (b < ⌊(B[v] + E[v])/2⌋) then
                    ВСТАВИТЬ(b,e;ЛСЫН[v]);
                 if (⌊(B[v] + E[v])/2⌋ < e) then
                    ВСТАВИТЬ(b,e;ПСЫН[v])
           end
end.
```

Действие ВСТАВИТЬ$(b, e;$ корень $(T))$ соответствует «маршруту» в $T$, имеющему следующую общую структуру (рис. 1.2): (возможно, пустой) начальный путь, именуемый $P_{\text{нач}}$, от корня до узла $v^*$, называемого *развилкой*, из которого выходят два

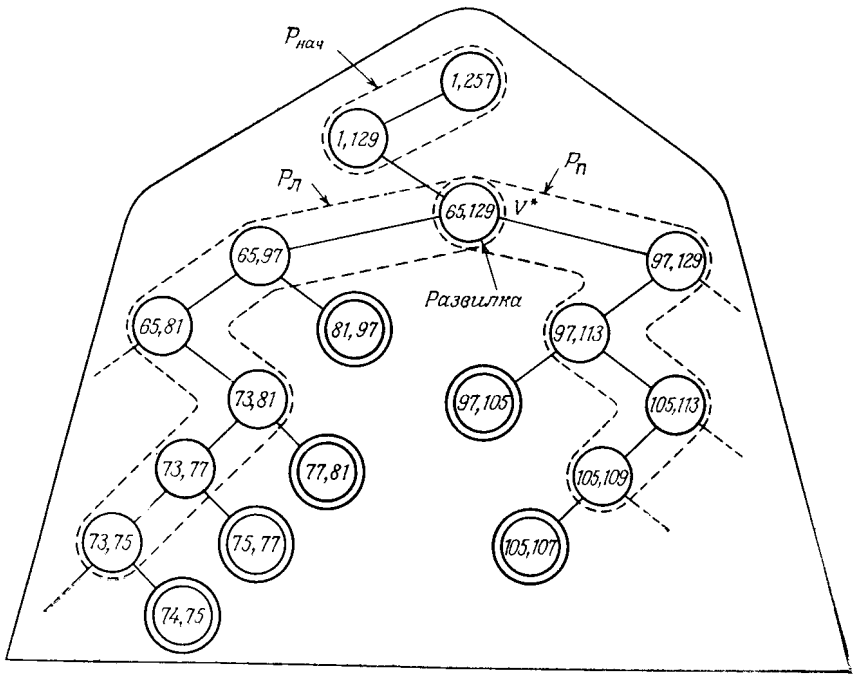



Рис. 1.2. Вставка интервала [74, 107] в $T(1, 257)$. Узлы отнесения окружены дважды.

(возможно, пустых) пути — $P_{\text{Л}}$ и $P_{\text{П}}$. Вставляемый интервал относится или полностью к развилке (в этом случае $P_{\text{Л}}$ и $P_{\text{П}}$ оба пусты), или ко всем правым сыновьям пути $P_{\text{Л}}$, которые сами не на $P_{\text{Л}}$, наряду со всеми левыми сыновьями пути $P_{\text{П}}$, которые сами не на $P_{\text{П}}$: при этом определяется фрагментация $[b, e]$ (*узлы отнесения*).



Отнесение интервала к узлу $v$ из $T$ может принимать разные формы в зависимости от потребностей конкретного приложения. Часто все, что надо знать, — это мощность множества интервалов, отнесенных к любому заданному узлу $v$; ее можно запомнить в единственном неотрицательном целом параметре $C[v]$, обозначающем эту мощность. Тогда отнесение $[b, e]$ к узлу $v$ — это всего лишь

$$C[v] := C[v] + 1.$$

В других приложениях надо сохранить сведения об интервалах, отнесенных к узлу $v$. Тогда к каждому узлу дерева $T$ добавляется вторичная структура — связный список $\mathscr{L}[v]$, чьи записи являются идентификаторами этих интервалов.

Совершенно симметрична операции ВСТАВИТЬ операция УДАЛИТЬ, выражаемая следующей процедурой (здесь предполагается, что нас интересует только состояние параметра $C[v]$):

```
      procedure УДАЛИТЬ(b,e;v)
begin if (b ⩽ B[v]) and (E[v] ⩽ e) then C[v] := C[v] − 1
      else begin if (b < ⌊(B[v] + E[v])/2⌋) then
                    УДАЛИТЬ(b,e;ЛСЫН[v]);
                 if (⌊(B[v] + E[v])/2⌋ < e) then
                    УДАЛИТЬ(b,e;ПСЫН[v])
           end
end.
```

(Заметим, что удаление только ранее вставленных интервалов гарантирует корректность.)

Дерево отрезков — чрезвычайно гибкая структура данных, в чем мы еще сможем убедиться в связи с многочисленными приложениями (гл. 2 и 8). Отметим только, что если надо знать число интервалов, содержащих данную точку $x$, то простой двоичный поиск в $T$ (т. е. прохождение пути от корня к листу) полностью решает эту задачу.

### 1.2.3.2. Реберный список с двойными связями

Реберный список с двойными связями (РСДС) особенно удобен для представления планарного графа, уложенного на плоскости[1]) [Muller, Preparata (1978)]. Плоская укладка планарного графа $G = (V, E)$ — это отображение каждой вершины из $V$ в точку на плоскости, а каждого ребра из $E$

[1]) Планарный граф, уложенный на плоскости, принято называть *плоским*. — *Прим. перев.*

в простую линию, соединяющую пару образов концевых вершин этого ребра так, чтобы образы ребер пересекались только в своих концевых точках. Хорошо известно, что любой планарный граф можно уложить на плоскости так, чтобы все ребра отобразились в прямолинейные отрезки [Fary (1948)].

Пусть $V = \{v_1, \ldots, v_N\}$, а $E = \{e_1, \ldots, e_M\}$. Главная компонента РСДС для планарного графа $(V, E)$ это *реберный узел*. Между ребрами и реберными узлами существует взаимно однозначное соответствие, т. е. каждое ребро представлено ровно один раз. Реберный узел содержит четыре информационных поля ($V1$, $V2$, $F1$, $F2$) и два поля указателей ($P1$ и $P2$); следовательно, соответствующую структуру данных легко реализовать как шесть массивов с теми же именами, каждый из которых состоит из $M$ ячеек. Значения этих полей таковы. Поле $V1$ содержит начало ребра, а поле $V2$ содержит его конец; так ребро получает условленную ориентацию. Поля $F1$ и $F2$ содержат имена граней, которые лежат соответственно слева и справа от ребра, ориентированного от $V1$ к $V2$. Указатель $P1$ (соответственно $P2$) задает реберный узел, содержащий первое ребро, встречаемое вслед за ребром ($V1$, $V2$), при повороте от него против часовой стрелки вокруг $V1$ (соответственно $V2$). Имена граней и вершин могут быть заданы целыми числами. В качестве иллюстрации фрагмент графа и соответствующий ему фрагмент РСДС показаны на рис. 1.3.

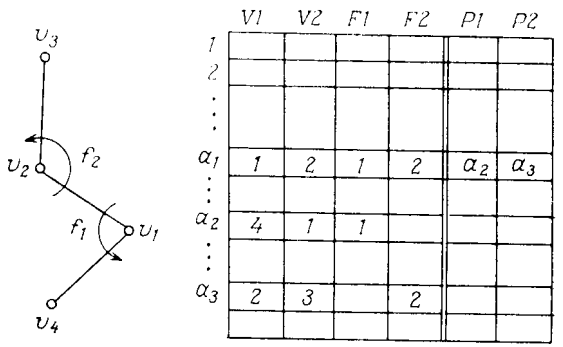



Рис. 1.3. Пример РСДС.

Теперь легко видеть, как с помощью РСДС можно вычислить ребра, инцидентные заданной вершине, или ребра, ограничивающие заданную грань. Если в графе $N$ вершин и $F$ граней, то мы можем ввести два массива ($HV[1:N]$ и $HF[1:F]$) для заголовков списков этих вершин и граней. Эти массивы можно заполнить при просмотре массивов $V1$ и $F1$ за время $O(N)$.



С помощью приведенной ниже процедуры ВЕРШИНА($j$) можно построить последовательность ребер, инцидентных $v_j$, как последовательность адресов, занесенных в массив $A$:

```
        procedure ВЕРШИНА(j)
begin   a := HV[j];
        a0 := a; A[1] := a;
        i := 2;
        if (V1[a] = j) then a := P1[a] else a := P2[a];
        while (a ≠ a0) do
            begin A[i] := a;
                  if (V1[a] = j) then a := P1[a] else a := P2[a];
                  i := i + 1
            end
end.
```

Ясно, что время работы процедуры ВЕРШИНА($j$) пропорционально числу ребер, инцидентных $v_j$. Аналогично мы можем создать процедуру ГРАНЬ($j$), с помощью которой можно получить последовательность ребер, ограничивающих $f_j$, заменив соответственно $HV$ и $V1$ на $HF$ и $F1$ в процедуре ВЕРШИНА($j$). Отметим, что процедура ВЕРШИНА перечисляет ребра в порядке обхода вокруг вершины против часовой стрелки, в то время как ГРАНЬ перечисляет их в порядке обхода по часовой стрелке вокруг грани.

Часто плоский граф $G = (V, E)$ представляется в форме реберного списка, где каждой вершине $v_j \in V$ сопоставлен список инцидентных ей ребер, перечисленных в том порядке, в котором они следуют при обходе против часовой стрелки вокруг $v_j$. Легко показать, что представление $G$ в виде реберного списка можно преобразовать в представление в форме РСДС за время $O(|V|)$.

## 1.3. Геометрические предпосылки

### 1.3.1. Общие определения и обозначения

Объекты, рассматриваемые в вычислительной геометрии, обычно задаются множествами точек в евклидовом пространстве[1]). Предполагается, что задана система координат, в которой каждая точка представима набором (вектором)

[1]) Ограничение рамками евклидовой геометрии (частного, но весьма важного случая метрической геометрии) позволяет нам прибегнуть к своему непосредственному опыту, но оно обосновано еще и тем, что подавляющее большинство приложений формулируется в терминах евклидова пространства. Однако такое ограничение несущественно для множества тех приложений, которые будут рассмотрены в следующих главах. Позднее мы еще раз вернемся к этой важной теме (разд. 1.3.2).

декартовых координат. Геометрические объекты не обязаны состоять из конечных множеств точек, но должны удовлетворять условию: *быть описанными конечным образом* (обычно конечными цепочками параметров). Следовательно, помимо изолированных точек будут рассматриваться также: прямая линия, содержащая две заданные точки; отрезок прямой линии, определенный парой своих концевых точек; плоскость, содержащая три заданные точки; многоугольник, определенный (упорядоченной) последовательностью точек, и т. п.

Этот раздел не претендует на введение формальных определений геометрических понятий, используемых в данной книге; мы хотим только напомнить о понятиях, хорошо известных читателю, и ввести удобные обозначения.

Обозначим через $E^d$ *d-мерное евклидово пространство*, т. е. пространство $d$-плексов $(x_1, \ldots, x_d)$, состоящих из действительных чисел $x_i$, $i = 1, \ldots, d$, с расстоянием $\left(\sum_{i=1}^{d} x_i^2\right)^{1/2}$. Определим теперь важнейшие объекты, рассматриваемые вычислительной геометрией.

*Точка.* Точкой $p$ в пространстве $E^d$ называется $d$-плекс $(x_1, \ldots, x_d)$. Эту точку можно интерпретировать также и как $d$-компонентный вектор, исходящий из начала координат в $E^d$, свободным концом которого является точка $p$.

*Прямая, плоскость, линейное многообразие.* Пусть даны две разные точки $q_1$ и $q_2$, принадлежащие $E^d$; тогда линейная комбинация

$$\alpha q_1 + (1 - \alpha) q_2 \quad (\alpha \in \mathbb{R})$$

называется *прямой* в $E^d$. В общем случае для заданных $k$ линейно независимых точек $q_1, \ldots, q_k$, принадлежащих $E^d$ $(k \leqslant d)$, линейная комбинация

$$\alpha_1 q_1 + \alpha_2 q_2 + \ldots + \alpha_{k-1} q_{k-1} + (1 - \alpha_1 - \ldots - \alpha_{k-1}) q_k$$
$$(\alpha_j \in \mathbb{R},\ j = 1, \ldots, k - 1)$$

называется *линейным многообразием* размерности $(k - 1)$ в $E^d$.

*Отрезок.* Пусть даны две разные точки $q_1$ и $q_2$, принадлежащие $E^d$, тогда линейная комбинация $\alpha q_1 + (1 - \alpha) q_2$ при условии $0 \leqslant \alpha \leqslant 1$ определит *выпуклую комбинацию* для $q_1$ и $q_2$, т. е.

$$\alpha q_1 + (1 - \alpha) q_2 \quad (\alpha \in \mathbb{R},\ \ 0 \leqslant \alpha \leqslant 1).$$

Эта выпуклая комбинация описывает *прямолинейный отрезок*, соединяющей две точки: $q_1$ и $q_2$. Обычно этот отрезок обозначают как $\overline{q_1 q_2}$ (неупорядоченная пара).



*Выпуклое многообразие.* Область $D$, принадлежащая пространству $E^d$, называется *выпуклой*, если для любой пары точек $q_1$ и $q_2$ из $D$ отрезок $\overline{q_1 q_2}$ целиком принадлежит $D$.

Можно доказать, что пересечением выпуклых областей является выпуклая область.

*Выпуклая оболочка. Выпуклой оболочкой* множества точек $S$, принадлежащих пространству $E^d$, называется граница наименьшей выпуклой области в $E^d$, которая охватывает $S$.

*Многоугольник.* Многоугольником в пространстве $E^2$ называется конечное множество отрезков, в котором каждый конец отрезка принадлежит ровно двум отрезкам и никакое подмножество этих отрезков не обладает указанным свойством. Эти отрезки называются *сторонами* (иногда *ребрами*), а их концы — *вершинами* многоугольника. (Заметим, что число сторон и число вершин совпадают.) Многоугольник с $N$ вершинами называется $N$-угольником.

Многоугольник называется *простым*, если никакая пара непоследовательных его ребер не имеет общих точек. Простой многоугольник разбивает плоскость на две непересекающиеся области — *внутреннюю* (конечную) и *внешнюю* (бесконечную), разделенные этим многоугольником (теорема Жордана). (Замечание: в обиходе термин «многоугольник» часто употребляется для обозначения объединения границы и внутренней области.)

Простой многоугольник $P$ называется *выпуклым*, если его внутренняя область является выпуклым множеством.

Простой многоугольник $P$ называется *звездным*[1]), если существует точка $z$, не внешняя для $P$, такая, что для всех точек $p$, принадлежащих $P$, отрезок $\overline{zp}$ полностью лежит внутри $P$. (Итак, любой выпуклый многоугольник звездный.) Множество точек $z$, обладающих указанным выше свойством, называется *ядром* $P$. (Очевидно, выпуклый многоугольник совпадает со своим ядром.)

*Планарный граф.* Граф $G = (V, E)$ (где $V$ — множество вершин, $E$ — множество ребер) называется *планарным*, если его можно уложить на плоскости без самопересечений (см. разд. 1.2.3.2). Прямолинейная укладка ребер планарного графа определяет разбиение плоскости, называемое *планарным подразбиением* или *картой*. Пусть $v$ — число вершин, $e$ — число ребер и $f$ — число граней (включая единственную бесконечную

[1]) Следует отличать его от звездчатого многоугольника, т. е. от самопересекающегося многоугольника, все стороны которого конгруэнтны и все углы конгруэнтны. — *Прим. ред.*

грань) такого подразбиения. Эти три параметра связаны классической *формулой Эйлера* [Bollobás (1979)]

$$v - e + f = 2. \tag{1.1}$$

Если мы, кроме того, знаем, что степень каждой вершины $\geqslant 3$, то в качестве простого упражнения можно доказать следующие неравенства:

$$\begin{cases} v \leqslant \frac{2}{3}e, & e \leqslant 3v - 6, \\ e \leqslant 3f - 6, & f \leqslant \frac{2}{3}e, \\ v \leqslant 2f - 4, & f \leqslant 2v - 4, \end{cases} \tag{1.2}$$

которые показывают, что $v$, $e$ и $f$ попарно пропорциональны. (Заметим, что три правых неравенства верны при всех условиях.)

*Триангуляция.* Планарное подразбиение называется *триангуляцией*, если все его конечные грани являются треугольниками. *Триангуляцией конечного множества точек* $S$ называется плоский граф $S$, имеющий наибольшее возможное число ребер (другими словами, триангуляция $S$ получена путем соединения точек из $S$ непересекающимися прямолинейными отрезками так, что любая грань, лежащая внутри выпуклой оболочки $S$, является треугольником).

*Полиэдр. Полиэдром* в пространстве $E^3$ называется такое конечное множество плоских многоугольников, когда каждая сторона любого многоугольника принадлежит еще ровно одному из остальных многоугольников (смежным многоугольникам) и никакое из подмножеств этого множества многоугольников не обладает указанным свойством. Вершины и стороны этих многоугольников являются *вершинами* и *ребрами* данного полиэдра; сами многоугольники называются *гранями* полиэдра.

Полиэдр называется *простым*, если никакая пара несмежных его граней не имеет общих точек. Простой полиэдр разбивает пространство на две непересекающиеся области — *внутреннюю* (конечную) и *внешнюю* (бесконечную). (И опять, в обиходе термин «полиэдр» часто используется для обозначения объединения границы и внутренней области.)

Поверхность любого полиэдра (рода нуль) изоморфна планарному подразбиению. Поэтому числа $v$, $e$ и $f$ соответственно для его вершин, ребер и граней удовлетворяют формуле Эйлера (1.1).

Простой полиэдр называется *выпуклым*, если его внутренняя область является выпуклым множеством.



### 1.3.3. Геометрическая двойственность. Поляритет

Теперь рассмотрим другую интерпретацию преобразования (1.8), вновь для простоты на примере $d = 2$. Итак, мы видели, что (1.8) отображает точки в точки, а прямые в прямые $E^2$, т. е. это преобразование сохраняет размерность отображаемых объектов. Предполагая, что $|B| \neq 0$, перепишем (1.8) в виде

$$\boldsymbol{\eta} = \boldsymbol{\xi} B \tag{1.9}$$

и далее будем считать, что векторы, обозначенные через $\boldsymbol{\xi}$, представляют направление (т. е. прямую, проходящую через начало координат), а векторы, обозначенные через $\boldsymbol{\eta}$, представляют нормаль к плоскости, проходящую через начало координат. Поэтому соотношение (1.9) интерпретируется как преобразование в $E^3$, переводящее прямую (представленную $\boldsymbol{\xi}$) в плоскость (представленную $\boldsymbol{\eta}$) и называемое корреляцией. Это частный случай (при $d = 2$) общего свойства: в пространстве $E^{d+1}$, по которому соотношение (1.9) отображает линейное многообразие размерности $s \leqslant d + 1$ в *двойственное* ему многообразие размерности $d + 1 - s$; в такой интерпретации (1.9) называется *преобразованием двойственности* и, действительно, та же самая матрица $B$ может использоваться для описания корреляции, которая переводит плоскости в прямые линии. Для заданной корреляции, описываемой матрицей $B$ и переводящей прямую в плоскость, мы хотим найти корреляцию, описываемую матрицей $D$ и переводящую плоскость в прямую, так чтобы пара $(B, D)$ сохраняла *инцидентность*, т. е. если прямая $\boldsymbol{\xi}$ принадлежит плоскости $\boldsymbol{\eta}$, то прямая $\boldsymbol{\eta}D$ принадлежит плоскости $\boldsymbol{\xi}B$. Очевидно, что прямая $\boldsymbol{\xi}$ принадлежит плоскости $\boldsymbol{\eta}$ тогда и только тогда, когда

$$\boldsymbol{\xi}\boldsymbol{\eta}^T = 0.$$

Поэтому потребуем, чтобы

$$\boldsymbol{\xi}B(\boldsymbol{\eta}D)^T = \boldsymbol{\xi}BD^T\boldsymbol{\eta}^T = 0.$$

Поскольку последнее уравнение должно выполняться для любых $\boldsymbol{\xi}$ и $\boldsymbol{\eta}$, мы получаем $BD^T = kI$, или

$$D = k(B^{-1})^T,$$

где $k$ — некоторая константа.

Заметим, что произведение $BD$ переводит прямые в прямые, а плоскости в плоскости. Теперь для заданной пары $(B, (B^{-1})^T)$, сохраняющей инцидентность, потребуем, чтобы произведение $B \cdot (B^{-1})^T$ переводило каждую прямую и каждую плоскость в самих себя, т. е. потребуем, чтобы $B \cdot (B^{-1})^T = kI$, или, что

то же самое, $B = kB^T$. В последнем равенстве нужно, чтобы $k = \pm 1$; особенно интересен случай, когда

$$B = B^T, \tag{1.10}$$

т. е. $B$ — невырожденная, симметричная матрица третьего порядка.

Теперь рассмотрим результат действия корреляции аналогичной (1.9) при центральном проецировании на плоскость $x_3 = 1$. Видно, что точки переходят в прямые, а прямые — в точки.

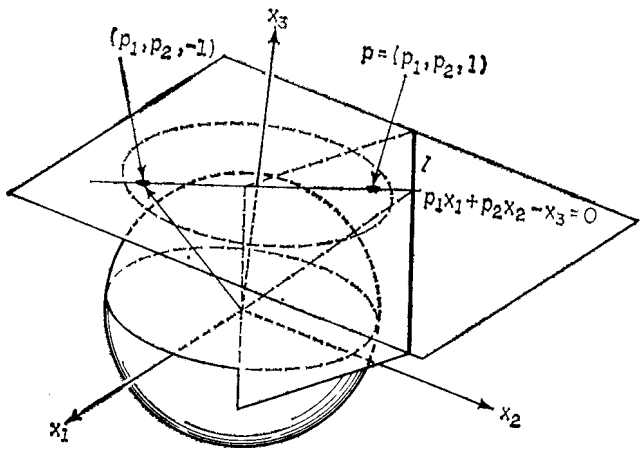



Рис. 1.6. Иллюстрация соответствия между полюсом и полярой при полярном преобразовании по отношению к единичной окружности.

Остаток этого раздела будет посвящен анализу этого отображения в $E^2$; через $\xi$ будем обозначать точки, а через $\eta$ — прямые. Если матрица $B$ третьего порядка удовлетворяет условию $B = B^T$, то, полагая $\mathbf{x} = (x_1, x_2, x_3)$, получаем хорошо известное соотношение [Birkhoff, MacLane (1965)]

$$\mathbf{x}B\mathbf{x}^T = 0;$$

это соотношение в однородных координатах задает коническое сечение на плоскости (известное как *коника, заданная* $B$). Возьмем фиксированный вектор $\xi$ (представляющий точку на плоскости в однородных координатах) такой, что $\xi B\xi^T = 0$. По определению точка $\xi$ лежит на конике, заданной $B$; если назвать точку $\xi$ *полюсом*, а прямую $\xi B\mathbf{x}^T$ *полярой* для $\xi$, то увидим, что полюс на конике инцидентен своей собственной поляре. Такое преобразование точек в прямые и *наоборот*, называемое поляритетом, полезно при создании геометрических алгоритмов. Это связано с тем, что интуитивно нам удобнее работать с точками, чем с прямыми (в $E^2$) или плоскостями (в $E^3$), и мы используем эту возможность в следующих главах.



В частности, пусть $B$ — матрица, определяющая единичную окружность на плоскости $E^2$ (рис. 1.6). В этом случае

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix},$$

поэтому точка $\mathbf{p} = (p_1, p_2, 1)$ отображается в прямую $l$, уравнение которой имеет вид $p_1x_1 + p_2x_2 - x_3 = 0$ (в однородных координатах). Расстояние от $\mathbf{p}$ до начала координат этой плоскости равно $\sqrt{p_1^2 + p_2^2}$, а расстояние от $l$ до этого начала координат равно $1/\sqrt{p_1^2 + p_2^2}$. Значит, в этом частном случае поляритета:

$$\text{расстояние}(\mathbf{p}, \mathbf{0}) \times \text{расстояние}(\text{поляра } (\mathbf{p}), \mathbf{0}) = 1.$$

Отметим также, что при этом поляритете преобразование не связано ни с какими вычислениями, просто по-разному интерпретируется тройка $(p_1, p_2, p_3)$: в одном случае — как координаты $(p_1/p_3, p_2/p_3)$ точки, а в другом — как коэффициенты уравнения прямой $p_1x_1 + p_2x_2 - p_3x_3 = 0$ в однородных координатах.

В следующих главах исследуются интересные приложения этой геометрической двойственности.

## 1.4. Модели вычислений

Главный методологический вопрос, с которым неизбежно сталкиваешься прежде всего при любом исследовании алгоритмов, заключается в тщательном описании принятой *модели вычислений*. Действительно, алгоритм, предложенный для решения определенной задачи, должен быть оценен в терминах его стоимости как некой функции от размера индивидуального экземпляра этой задачи. Принципиальная важность модели вычислений определяется следующим образом:

*Модель вычислений определяет набор допустимых элементарных операций и стоимости этих операций.*

*Элементарные операции* — это такие операции, для каждой из которых мы назначаем *фиксированную* стоимость, хотя эта стоимость неодинакова для разных элементарных операций. Например, если элементарные операции обрабатывают отдельные цифры в числах (как в булевских функциях с двумя двоичными переменными, вычисляемыми на модели машины Тьюринга), то стоимость операции сложения двух целых чисел возрастает с ростом длины операндов, в то время как стоимость этой операции постоянна для модели, в которой операнды имеют фиксиро-

ванную длину (как в любой модели, ориентированной на представление реальных цифровых компьютеров). При выборе модели приходится идти на компромиссы между реальностью и математической строгостью, выбирая схему, которая отражает основные черты используемых математических методов и вместе с тем достаточно проста, чтобы провести тщательный анализ.

Какой же тип модели подходит для геометрических приложений? Чтобы ответить на этот центральный вопрос, нужно тщательно рассмотреть природу решаемых задач.

Как указано в разд. 1.3, главными объектами, которые рассматриваются в вычислительной геометрии, являются точки, принадлежащие пространству $E^d$. Хотя точка считается вектором в декартовых координатах, желательно, чтобы выбор системы координат не слишком влиял на время работы геометрического алгоритма. Это приводит к тому, что модель вычислений должна допускать необходимое преобразование (декартова базиса), причем его стоимость в расчете на одну точку может зависеть от числа измерений, *но не от* числа обрабатываемых точек.

С другой стороны, известно, что множество из $N$ точек в пространстве размерности $d$ может быть выражено по отношению к заданному декартову базису за время, пропорциональное $N$ (напомним, что столько же времени по порядку нужно для «считывания» этих точек как исходных данных), поэтому мы можем с самого начала считать, что эти точки уже заданы в избранном декартовом базисе.

Задачи, возникающие в вычислительной геометрии, относятся к нескольким типам, которые мы для удобства сгруппируем в три следующих класса:

1. *Поиск подмножества.* В задачах этого рода задан набор объектов и требуется найти некое подмножество, которое удовлетворяет определенному условию. Например, поиск ближайшей пары на множестве из $N$ точек или поиск вершин выпуклой оболочки множества. Важная черта задач о выборе подмножества заключается в том, что не нужно создавать никаких новых объектов; решение полностью состоит из элементов, которые заданы на входе.

2. *Вычисление.* Дано множество объектов, нужно вычислить величину некоего геометрического параметра на этом множестве. Допустимые элементарные операции в модели должны быть достаточно мощными, чтобы реализовать это вычисление. Предположим, например, что точки имеют целочисленные координаты. Чтобы найти расстояние между парой точек, надо уметь не только представлять иррациональные числа, но также извлекать квадратные корни. А в ряде задач нам могут понадобиться даже тригонометрические функции.



3. *Распознавание*[1]. Задача распознавания естественным образом связана с задачами «Поиск подмножества» и «Вычисление». В частности,

(1) Если в задаче $\mathscr{A}$ о вычислении требуется найти величину параметра $A$, то в связанной с ней задаче распознавания $D(\mathscr{A})$ требуется дать ответ ДА/НЕТ на вопрос типа: «Верно ли, что $A \geqslant A_0$?», где $A_0$ — константа.

(2) Если в задаче $\mathscr{A}$ о поиске подмножества требуется найти подмножество заданного множества $S$, удовлетворяющее определенному свойству $P$, то в задаче $D(\mathscr{A})$ требуется ответ ДА/НЕТ на вопрос типа: «Верно ли, что $S'$ удовлетворяет $P$?», где $S'$ — подмножество $S$.

Легко заметить, что есть необходимость работать с *действительными* числами (а не только с целыми) и (на реальном компьютере) с соответствующими им приближениями. Однако при создании модели вычислений можно воспользоваться абстрактным понятием, чтобы избежать проблемы, связанной с округлением для приближенного представления действительного числа. Конкретно, возмем *машину с произвольным доступом к памяти* (РАМ), похожую на ту, что описана в [Aho, Hopcroft, Ullman (1974)][2]), но у которой *каждая ячейка памяти способна хранить единственное действительное число. Следующие операции считаются элементарными и обладают единичной стоимостью (единичным временем выполнения)*:

1. Арифметические операции ($+$, $-$, $\times$, $/$).
2. Сравнения двух действительных чисел ($<$, $\leqslant$, $=$, $\neq$, $\geqslant$, $>$).
3. Косвенная адресация памяти (допустимы только целочисленные адреса).

Дополнительные операции (только для тех приложений, где это необходимо):

4. Корни $k$-й степени, тригонометрические функции, EXP, LOG (вообще аналитические выражения).

Эту модель будем называть *вещественнозначной* РАМ. Она хорошо отражает тот тип программ, которые обычно пишутся на алгоритмических языках высокого уровня, таких как Фортран и Алгол, в которых принято считать, что переменные типа REAL имеют неограниченную точность. На этом уровне абстракции можно игнорировать вопросы типа: «Как ввести или вывести действительное число за конечное время?»

[1]) См. [Гэри, Джонсон (1982), с. 27]. — *Прим. перев.*

[2]) В русском переводе этой книги аббревиатура РАМ расшифровывается как «равнодоступная адресная машина». — *Прим. ред.*

Установление нижних оценок для характеристик работы алгоритмов решения определенной задачи — одна из главных целей анализа алгоритмов, поскольку они дают меру для оценки эффективности конкретных алгоритмов. Но это, вообще говоря, очень трудная проблема. Иногда удается установить трудоемкость одной задачи по известной трудоемкости другой задачи, используя метод *преобразования задач*[1]). В частности, предположим, что у нас есть две задачи, $\mathscr{A}$ и $\mathscr{B}$, которые связаны так, что задачу $\mathscr{A}$ можно решить следующим образом:

1. Исходные данные к задаче $\mathscr{A}$ преобразуются в соответствующие исходные данные для задачи $\mathscr{B}$.
2. Решается задача $\mathscr{B}$.
3. Результат решения задачи $\mathscr{B}$ преобразуется в правильное решение задачи $\mathscr{A}$.

В этом случае мы говорим, что задача $\mathscr{A}$ *преобразуема* в задачу $\mathscr{B}$[2]). Если шаги 1 и 3 вышеприведенного преобразования можно выполнить за время $O(\tau(N))$, где, как обычно, $N$ — «размер» задачи $\mathscr{A}$, то скажем, что $\mathscr{A}$ $\tau(N)$-*преобразуема* в $\mathscr{B}$, и запишем это кратко так: $\mathscr{A} \propto_{\tau(N)} \mathscr{B}$. Вообще говоря, преобразуемость не симметричное отношение; в частном случае, когда $\mathscr{A}$ и $\mathscr{B}$ взаимно преобразуемы, мы назовем их *эквивалентными*.

Следующие два самоочевидных утверждения характеризуют мощь метода преобразования в предположении, что это преобразование сохраняет порядок размера задачи.

**Предложение 1** (Нижние оценки методом преобразования). *Если известно, что задача $\mathscr{A}$ требует $T(N)$ времени и $\mathscr{A}$ $\tau(N)$-преобразуема в $\mathscr{B}$* (т. е. $\mathscr{A} \propto_{\tau(N)} \mathscr{B}$), *то задача $\mathscr{B}$ требует не менее $T(N) - O(\tau(N))$ времени.*

**Предложение 2** (Верхние оценки методом преобразования). *Если задачу $\mathscr{B}$ можно решить за время $T(N)$ и задача $\mathscr{A}$ $\tau(N)$-преобразуема в $\mathscr{B}$* (т. е. $\mathscr{A} \propto_{\tau(N)} \mathscr{B}$), *то $\mathscr{A}$ можно решить за время, не превышающее $T(N) + O(\tau(N))$.*

Эту ситуацию графически иллюстрирует рис. 1.7, на котором показано, как нижняя и верхняя оценки переносятся от одной задачи к другой. Этот перенос справедлив, когда $\tau(N) = O(T(N))$, т. е. когда время преобразования не превосходит времени вычисления.

[1]) Этот метод очень часто именуют *сведе́нием*. Поскольку «сведе́ние» предполагает преобразование к более простой задаче (а в данном случае это не так), то мы постараемся обойтись без этого понятия.

[2]) Преобразуемость (или сводимость) обычно определяется по отношению к языкам. В этом случае преобразование результатов не требуется, поскольку в качестве результата распознаватель цепочки выдает нуль или единицу. Для геометрических задач необходима большая гибкость, подкрепленная более общим определением.



Возвращаясь к нашей прежней классификации задач, возьмем некую задачу $\mathscr{A}$ (задачу вычисления или поиска подмножества) и связанную с ней задачу распознавания $D(\mathscr{A})$. Легко видеть, что $D(\mathscr{A}) \propto_{O(N)} \mathscr{A}$, поскольку:

1. Если $\mathscr{A}$ — задача вычисления, то никакого преобразования исходных данных не требуется (т. е. шаг 1 процедуры преобразования не нужен), а решение $\mathscr{A}$ надо просто сравнить за постоянное время $O(1)$ с фиксированной величиной, даваемой $D(\mathscr{A})$.

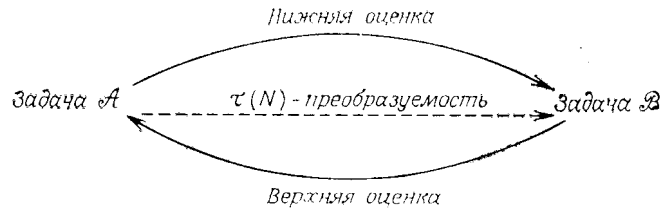



Рис. 1.7. Перенос верхней и нижней оценок между преобразуемыми задачами.

2. Если же $\mathscr{A}$ — задача поиска подмножества, то опять входная информация $S'$ для $D(\mathscr{A})$ подается на вход $\mathscr{A}$ (т. е. шаг 1 не нужен), а решение $\mathscr{A}$ проверяется за время $O(N)$, чтобы убедиться в том, что его мощность совпадает с мощностью $S'$.

Это очень важное рассуждение, поскольку оно показывает, что, когда мы ищем нижние оценки, можно сосредоточить свое внимание на задачах распознавания.

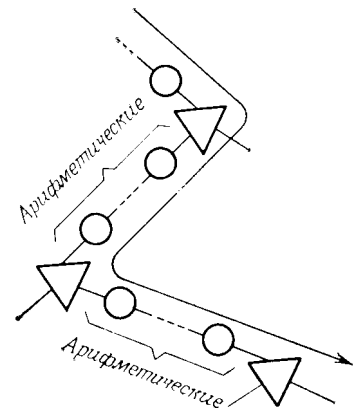



Рис. 1.8. Вычисление как путь в дереве решений.

Когда вещественнозначная RAM исполняет алгоритм для задачи распознавания, ее поведение описывается последовательностью операций двух типов: арифметических и сравнения. В этой последовательности главную роль играют сравнения, поскольку в зависимости от результата каждого сравнения алгоритм выбирает ту или иную ветвь выполнения. Эта ситуация графически показана на рис. 1.8; там каждый кружок представляет собой арифметическую операцию, а каждый треугольник — сравнение. Другими словами, процесс вычислений, исполняемый на RAM, можно рассматривать как путь на дереве с корнем.

Это корневое дерево заключает в себе описание чрезвычайно важной вычислительной модели, называемой «алгебраическим деревом решений», которую мы сейчас формализуем.

*Алгебраическим деревом решений* [Reingold (1972); Rabin (1972); Dobkin, Lipton (1979)] на множестве переменных $\{x_1, \ldots, x_n\}$ называется программа, записанная операторами $L_1, L_2, \ldots, L_t$, следующего типа:

1. $L_r$: Вычислить $f(x_1, \ldots, x_n)$. Если $f$: 0, то перейти к $L_i$, в противном случае к $L_j$ (двоеточие обозначает любую из операций сравнения).
2. $L_s$: Останов и выдача ДА (в задаче распознавания вход принят).
3. $L_t$: Останов и выдача НЕТ (в задаче распознавания вход отвергнут).

В случае 1 $f$ — это *алгебраическая функция* (полином степени *равной порядку* ($f$)). Далее предполагается, что программа не имеет возвратов, т. е. имеет структуру *дерева* $T$, при которой каждый нелистовой узел $v$ описывается выражением

$$f_v(x_1, \ldots, x_n)\colon 0,$$

где $f_v$ — полином от $x_1, \ldots, x_n$, а двоеточие обозначает отношение сравнения. (Заметим, что в этом определении каждый «арифметический отрезок пути» на рис. 1.8 свернут в следующий за ним узел сравнения.) Корень $T$ представляет начальный шаг вычислений, а его листья представляют возможные окончания и содержат все возможные ответы. Без потери общности допустим, что дерево $T$ *двоичное* [1]).

Хотя программа на алгебраическом дереве решений может оказаться намного более громоздкой, чем соответствующая RAM-программа, однако для изучаемых нами классов задач обе программы ведут себя *идентично.* В частности, время работы RAM-программы для худшего случая по меньшей мере пропорционально длине максимального пути от корня в дереве решений. Это подтверждает важность модели дерева решений, поскольку структура дерева достаточно удобна для получения оценок его высоты.

Говорят, что алгебраическое дерево решений имеет *порядок* $d$, если $d$ — это наибольшая из степеней полиномов $f_v(x_1, \ldots, x_n)$ среди всех узлов $v$ из $T$. Модель дерева решений 1-го порядка, или *линейная* модель, является мощным средством

[1]) Заметим, что степень узла $T$ равна числу различных исходов операции сравнения. Допущение о двоичности дерева основано на том факте, что $k$-кратное разветвление может быть заменено на $k-1$ экземпляров 2-кратных разветвлений.



установления нижних оценок для многих задач; мы еще обсудим те весьма тонкие рассуждения, которые будут развиты в последующих главах, в сответствующем контексте (см. разд. 2.2, 4.1.3, 5.2, 8.4). Однако ограничиться рассмотрением только линейных деревьев решений нельзя по двум причинам. Первая и более важная состоит в том, что может случиться, когда любой из известных алгоритмов для данной задачи использует функции со степенью $\geqslant 2$, поэтому нижняя оценка, основанная на модели линейного дерева решений, окажется бесполезной; если же исключить из рассмотрения подобную ситуацию, то возникнет вторая причина — оценка, основанная на модели линейного дерева решений, не применима к пока неизвестным алгоритмам, использующим функции более высоких степеней.

Весьма важные вклады в решение этих вопросов для $d \geqslant 2$ были недавно внесены работами Стили, Яо [Steele, Jao (1982)] и Бен-Ора [Ben-Or (1983)], использующими классические понятия действительной алгебраической геометрии. Их подход основан на следующей очень простой идее: пусть $x_1, x_2, \ldots, x_n$ — параметры задачи распознавания, каждый индивидуальный экземпляр которой может считаться точкой в $n$-мерном евклидовом пространстве $E^n$. Задача распознавания определяет множество точек $W \subseteq E^n$, т. е. она дает ответ ДА тогда и только тогда, когда $(x_1, \ldots, x_n) \in W$ (можно сказать, что *дерево решений* $T$ *решает задачу о принадлежности точки множеству* $W$). Предположим, что нам заранее известно, исходя из природы этой задачи, число $\#(W)$ разделимых связных компонент $W$. Каждое вычисление соответствует какому-то однозначному пути $v_1, v_2, \ldots, v_{l-1}, v_l$ в $T$, где $v_1$ — корень, а $v_l$ — лист; с каждым узлом $v_j$ на этом пути ($j = 1, \ldots, l-1$) связана функция $f_{v_j}(x_1, \ldots, x_n)$, так что $(x_1, \ldots, x_n)$ удовлетворяет ограничению типа

$$f_{v_j} = 0, \quad \text{или} \quad f_{v_j} \geqslant 0, \quad \text{или} \quad f_{v_j} > 0. \tag{1.11}$$

Для приобретения некоторой интуиции рассмотрим вначале частный случай $d = 1$ (линейная модель дерева решений или вычислений), следуя за Добкином и Липтоном [Dobkin, Lipton (1979)]. Доказательство их теоремы следует ниже.

Пусть $W \subseteq E^n$ — множество истинности [1]) для данной задачи распознавания, и пусть $\#W$ обозначает число разделимых связных компонент $W$. Пусть $T$ — (двоичное) линейное дерево решений, реализующее алгоритм $\mathscr{A}$, который проверяет принадлежность $W$. С каждым листом $T$ связана область в $E^n$, и каждый лист является принимающим или отвергающим. А именно пусть

[1]) То есть на выходах из $W$ задача имеет ответ «ДА». — *Прим. перев.*

$\{W_1, \ldots, W_p\}$ — компоненты $W$, $\{l_1, \ldots, l_r\}$ — множество листьев, а $D_j$ — область, связанная с $l_j$ [1]). Лист $l_j$ является:

$$\begin{cases} \text{принимающим, если } D_j \subseteq W; \\ \text{отвергающим в противном случае.} \end{cases}$$

Нижняя оценка для числа листьев $r$ получается путем доказательства того, что $r \geqslant \#(W)$. Действительно, возьмем функцию $Y$: $\{W_1, W_2, \ldots, W_p\} \to \{1, 2, \ldots, r\}$ такую, что $Y(W_i) = \min\{j: j \in \{1, 2, \ldots, r\}$ и $D_j \cap W_i \neq \varnothing\}$. Чтобы получить противоречие, предположим, что существуют два различных подмножества $W_i$ и $W_j$ таких, что $Y(W_i) = Y(W_j) = h$. Поскольку

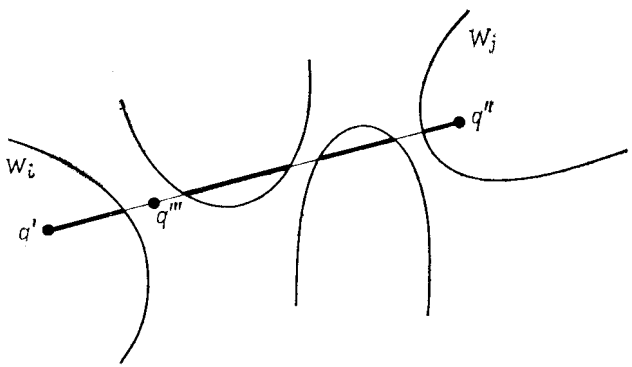



Рис. 1.9. Иллюстрация к доказательству того, что $W_i = W_j$.

алгоритм $\mathscr{A}$ решает вопрос о принадлежности точки $q$ множеству $W_i$, то лист $l_h$ является принимающим. С другой стороны, по определению $Y$ из равенства $Y(W_i) = h$ следует, что $W_i \cap D_h$ непусто. Пусть $q'$ — точка из $W_i \cap D_h$, аналогично $q''$ — точка из $W_j \cap D_h$. Поскольку $T$ — *линейное* дерево решений, то область в $E^n$, соответствующая $D_h$, образована пересечением полупространств, т. е. это *выпуклая* область. Значит, любая выпуклая комбинация точек из $D_h$ принадлежит $D_h$ (разд. 1.3.1). Рассмотрим теперь отрезок $\overline{q'q''}$ (рис. 1.9). Этот отрезок пересекает по крайней мере две компоненты $W$ (он, безусловно, пересекает $W_i$ и $W_j$), и, поскольку эти компоненты разделены, он содержит по крайней мере одну точку $q''' \notin W$. Но так как область $D_h$ выпукла, то весь отрезок $\overline{q'q''}$ принадлежит $D_h$, а, значит, и точка $q'''$ принадлежит внутренности $W$. Получаем противоречие. В итоге $T$ должно иметь не менее $p$ листов. Поскольку двоичное дерево наименьшей высоты с заданным числом листов сбалансировано, то высота $T$ равна не менее чем $\log_2 p = \log_2 \#(W)$. Эти рассуждения приводят к следующей теореме.

[1]) Область входных данных, переводящих алгоритм в лист $l_j$. — *Прим. перев.*



**Теорема 1.1** (Добкин — Липтон). *Любое линейное дерево решений, алгоритм которого решает задачу о принадлежности $W \subseteq E^n$, должно иметь высоту не меньше, чем $\log_2 \#(W)$, где $\#(W)$ — число разделимых связных компонент $W$.*

К сожалению, вышеизложенный метод ограничен алгоритмами линейного дерева вычислений, поскольку он опирается на свойство выпуклости области в $E^n$, связанной с листом этого дерева. Если же максимальная степень полиномов $f_v \geqslant 2$, то это полезное свойство исчезает. Значит, нужны более глубокие представления.

Интуитивно ясно, что, когда используются полиномы более высоких степеней, тогда область, связанная с конкретным листом дерева решений, может состоять из нескольких разделимых компонент в $W$. Если удастся оценить число компонент, связанных с неким листом в терминах высоты этого листа в дереве, то, как мы вскоре увидим, нам удастся оценить высоту $T$.

К счастью, ключ к решению этой трудной проблемы дала утонченная адаптация [Steele, Yao (1982); Ben-Or (1983)] одного классического результата из алгебраической геометрии, независимо полученного Милнором [Milnor (1964)] и Томом [Thom (1965)]. Ниже излагается результат Милнора и Тома.

Пусть $V$ — множество точек (так называемое алгебраическое многообразие) в $m$-мерном декартовом пространстве $E^m$, определенное полиномиальными уравнениями

$$g_1(x_1, \ldots, x_m) = 0; \quad \ldots; \quad g_p(x_1, \ldots, x_m) = 0. \tag{1.12}$$

Тогда, если степень каждого полинома $g_i$ $(i = 1, \ldots, p)$ не превышает $d$, то число $\#(V)$ разделимых связных компонент [1]) $V$ оценивается как:

$$\#(V) \leqslant d \cdot (2d-1)^{m-1}.$$

К сожалению, множество $V$ определено лишь посредством равенств (1.12), в то время как ограничения, связанные с путем в $T$, состоят как из равенств, так и из неравенств. Эта трудность была блестяще преодолена Бен-Ором, который преобразовал ситуацию с деревом $T$ так, что она стала удовлетворять условию теоремы Милнора — Тома, как это будет сейчас показано.

[1]) В действительности Милнор и Том доказали более сильный результат, что $d(2d-1)^{m-1}$ — это верхняя оценка суммы чисел Бетти для множества $V$, в то время как число $\#(V)$ связных компонент $V$ — это только одно из таких чисел (число Бетти 0-го порядка).

Пусть $U \subseteq E^n$ — множество точек, удовлетворяющих следующим ограничениям (здесь $\mathbf{x} = (x_1, \ldots, x_n)$):

$$\begin{cases} q_1(\mathbf{x}) = 0, \ldots, q_r(\mathbf{x}) = 0, \\ p_1(\mathbf{x}) > 0, \ldots, p_s(\mathbf{x}) > 0, \\ p_{s+1}(\mathbf{x}) \geqslant 0, \ldots, p_h(\mathbf{x}) \geqslant 0, \end{cases} \tag{1.13}$$

где $q$ и $p$ — полиномы, $d = \max\{2, \text{степень } (q_i), \text{степень } (p_j)\}$. Заметим, что имеется три типа ограничений: равенства, строгие неравенства и нестрогие неравенства. Пусть $\#(U)$ обозначает число связных компонент множества $U$.

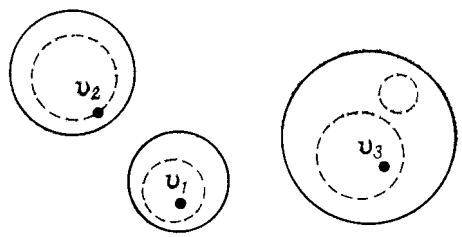



Рис. 1.10. Построение $U_\varepsilon$ по $U$; $U$ ограничено сплошными линиями, $U_\varepsilon$ — штриховыми.

Первый шаг преобразования состоит в замене строгих неравенств нестрогими. Поскольку $\#(U)$ конечно (см. [Milnor (1968)]), положим $\#(U) \triangleq t$ и поместим в каждую компоненту $U$ по точке. Обозначив эти точки $\mathbf{v}_1, \ldots, \mathbf{v}_t$, положим

$$\varepsilon = \min\{p_i(\mathbf{v}_j): \quad i = 1, \ldots, s; \ j = 1, \ldots, t\}.$$

Поскольку каждая точка $\mathbf{v}_j \in U$, то $p_i(\mathbf{v}_j) > 0$ (для $i = 1, \ldots, s$) и $\varepsilon > 0$. Очевидно, что замкнутое множество точек $U_\varepsilon$, определяемое системой

$$\begin{cases} q_1(\mathbf{x}) = 0, \ldots, q_r(\mathbf{x}) = 0, \\ p_1(\mathbf{x}) \geqslant \varepsilon, \ldots, p_s(\mathbf{x}) \geqslant \varepsilon, \\ p_{s+1}(\mathbf{x}) \geqslant 0, \ldots, p_h(\mathbf{x}) \geqslant 0, \end{cases} \tag{1.14}$$

содержится в $U$, $\#(U_\varepsilon) \geqslant \#(U)$, так как все $\mathbf{v}_j$ лежат в разных связных компонентах $U_\varepsilon$ (см. в качестве интуитивного примера рис. 1.10).

Второй шаг преобразования состоит во введении искусственных переменных $y_j$ для всех $j = 1, \ldots, h$ (т. е. по одной для каждого из нестрогих неравенств), чтобы $E^n$ рассматривалось как такое подпространство в $E^{n+h}$, где

$$\begin{cases} q_1(\mathbf{x}) = 0, \ldots, q_r(\mathbf{x}) = 0, \\ p_1(\mathbf{x}) - \varepsilon - y_1^2 = 0, \ldots, p_s(\mathbf{x}) - \varepsilon - y_s^2 = 0, \\ p_{s+1}(\mathbf{x}) - y_{s+1}^2 = 0, \ldots, p_h(\mathbf{x}) - y_h^2 = 0. \end{cases} \tag{1.15}$$



Очевидно, что множество $U^*$ всех решений (1.15) представляет собой алгебраическое многообразие в $E^{n+h}$, определенное полиномами степени, не превышающей $d$ и удовлетворяющее условиям теоремы Милнора — Тома; следовательно, число $\#(U^*)$ связных компонент $U^*$ оценивается следующим образом:

$$\#(U^*) \leqslant d(2d-1)^{n+h-1}.$$

Заметим, что $U_\varepsilon$ — это проекция $U^*$ на $E^n$; поскольку проецирование является непрерывной функцией, то $\#(U_\varepsilon) \leqslant \#(U^*)$. Вспоминая, что $\#(U) \leqslant \#(U_\varepsilon)$, мы заключаем:

$$\#(U) \leqslant \#(U_\varepsilon) \leqslant \#(U^*) \leqslant d \cdot (2d-1)^{n+h-1},$$

что и является искомым результатом. В самом деле, если (1.13) есть множество ограничений, полученных при прохождении пути в $T$ от корня к листу, то число неравенств $h$ по величине не превосходит длины этого пути. Отсюда следует, что лист на этом пути связан не более чем с $d(2d-1)^{n+h-1}$ связными компонентами из множества $W$ истинности для данной задачи. Если $h^*$ — высота $T$ (длина наибольшего пути от корня до листа), то $T$ имеет не более $2^{h^*}$ листов; поскольку каждый лист связан не более чем с $d(2d-1)^{n+h^*-1}$ связными компонентами из $W$, то

$$\#(W) \leqslant 2^{h^*} \cdot d \cdot (2d-1)^{n+h^*-1},$$

или, что то же самое,

$$h^* \geqslant \frac{\log_2 \geqslant (W)}{1 + \log_2(2d-1)} - \frac{n \log_2(2d-1)}{1 + \log_2(2d-1)} - \frac{\log_2 d - \log_2(2d-1)}{1 + \log_2(2d-1)}. \tag{1.16}$$

Сформулируем этот важный результат в виде теоремы.

**Теорема 1.2** [1]. *Пусть $W$ — множество в декартовом пространстве $E^n$, а $T$ — алгебраическое дерево решений фиксированного порядка $d$ ($d \geqslant 2$), которое решает задачу о принадлежности $W$. Если $h^*$ — высота $T$, а $\#(W)$ — число разделимых связных компонент $W$, то $h^* = \Omega(\log \#(W) - n)$.*

Теорема 1.2 включает и случай $d = 1$, поскольку при любом фиксированном $d \geqslant 2$ допустимы и полиномы меньших степеней (простым обнулением коэффициентов при высших степенях). По существу, использование нелинейных полиномов в качестве функций принятия решений не изменяет природу этой задачи радикально; просто наибольшая высота деревьев вычислений уменьшается на мультипликативную константу, зависящую от максимального порядка $d$. Последняя теорема — краеугольный камень в построениях нижних оценок, которые будут представлены в следующих главах.

[1]) В действительности Бен-Ор доказал несколько более сильный результат, независимый от $d$.

ГЛАВА 2

# Геометрический поиск

Чтобы описать поиск в простейшей абстрактной форме, представим себе, что у нас есть некий набор данных (именуемый файлом) и некоторый новый элемент данных (именуемый образцом). Поиск — это установление связи между образцом и файлом. К вспомогательным, но в принципе не чисто поисковым операциям можно отнести: включение образца в файл, удаление образца из файла, если он был там, и т. п. Как указал Кнут [Knuth (1973)], поиск сводится к определению позиции соответствующей записи (или записей) в данном наборе записей.

Хотя все виды поиска обладают рядом общих базисных черт, его особая геометрическая форма позволяет ставить вопросы со своими неповторимыми нюансами. Во-первых, потому что в геометрических приложениях файлы это не просто «коллекции», как в ряде других областей информатики. Чаще они отображают более сложные структуры, такие как многоугольники, полиэдры и т. п. Хотя случай, скажем, набора отрезков на плоскости выглядит обманчиво бесструктурным, на самом деле каждому отрезку сопоставлены его концы, координаты которых косвенно связаны между собой метрической структурой этой плоскости.

Тот факт, что координаты точек представимы действительными числами, часто ведет к тому, что результатом поиска может оказаться не элемент файла, соответствующий образцу, а скорее *положение* последнего относительно файла. Это и составляет второе отличие геометрического поиска от обычного.

## 2.1. Введение в геометрический поиск

В данной главе описываются основные методы геометрического поиска, которые будут использованы в последующих главах для решения более трудных задач. Поисковое сообщение, в соответствии с которым ведется просмотр файла, обычно именуется *запросом*. От типа файла и от набора допустимых запросов будут сильно зависеть организация первого и алгоритмы обработки последних. Один конкретный пример позволит убедиться, насколько важен этот аспект задачи.



Пусть имеется набор геометрических данных и нужно узнать, обладают ли они определенным свойством (скажем, выпуклостью). В простейшем случае, когда этот вопрос возникает единожды, было расточительством проводить предобработку в надежде ускорить прохождение последующих запросов. Назовем разовый запрос такого типа *уникальным*. Однако будут и запросы, обработка которых повторяется многократно на том же самом файле. Такие запросы назовем *массовыми*.

В последнем случае, возможно, стоит расположить информацию в соответствии с некоторой структурой, облегчающей поиск. Однако это можно проделать, только затратив некоторый ресурс, и анализ надо сосредоточить на четырех различных мерах его оценки:

1. *Время запроса.* Сколько времени необходимо как в среднем, так и в худшем случае для ответа на один запрос?
2. *Память.* Сколько памяти необходимо для структуры данных?
3. *Время предобработки.* Сколько времени необходимо для организации данных перед поиском?
4. *Время корректировки.* Указан элемент данных. Сколько времени потребуется на его включение в структуру данных или удаление из нее?

Различные варианты затрат времени запроса, времени предобработки и памяти хорошо продемонстрированы на следующем примере решения задачи *регионального поиска* [Knuth (1978), т. 3] [1]), которая часто возникает в географических приложениях и при управлении базами данных:

**Задача П.1** (РЕГИОНАЛЬНЫЙ ПОИСК — ПОДСЧЕТ). Даны $N$ точек на плоскости. Сколько из них лежит внутри заданного прямоугольника, стороны которого параллельны координатным осям? То есть сколько точек $(x, y)$ удовлетворяют неравенствам $a \leqslant x \leqslant b$, $c \leqslant y \leqslant d$ для заданных $a$, $b$, $c$ и $d$ (рис. 2.1)?

Очевидно, что уникальный региональный запрос может быть обработан (оптимально) за линейное время, так как надо только проверить каждую из $N$ точек, чтобы увидеть, удовлетворяет ли она неравенствам, задающим прямоугольник. Аналогично необходима линейная затрата памяти, так как следует запомнить только $2N$ координат. Нет никаких затрат на предобработку, а

[1]) Эта задача будет рассмотрена в полном объеме в разд. 2.3.

время корректировки для новой точки равно константе. Какую структуру данных можно использовать для ускорения обработки массовых запросов? Кажется, что слишком трудно найти такое упорядочение точек, чтобы любой новый прямоугольник мог быть легко с ним согласован. Мы не можем также решить эту задачу наперед для *всех* возможных прямоугольников, так как их число бесконечно. Следующее решение является примером использования *метода локусов* в геометрических задачах: запросу ставится в соответствие точка в удобном для поиска

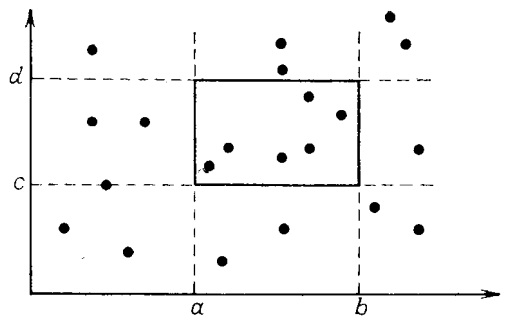



Рис. 2.1. Региональный запрос. Сколько точек внутри прямоугольника?

пространстве, а это пространство разбивается на области (*локусы*) [1]), в пределах которых ответ не изменяется. Другими словами, если считать эквивалентными два запроса, на которые поступают одинаковые ответы, то каждая область разбиения пространства поиска соответствует одному классу эквивалентности запросов.

Прямоугольник сам по себе — неудобный объект; мы предпочитаем работать с точками. Это означает, например, что можно заменить запрос с прямоугольником четырьмя подзадачами, по одной на каждую из его вершин, и совместить их решения для получения окончательного ответа. В этом случае подзадача, связанная с вершиной $p$, состоит в определении числа точек $Q(p)$ исходного множества, которые удовлетворяют неравенствам $x \leqslant x(p)$ и $y \leqslant y(p)$, т. е. числа точек в левом нижнем квадранте, определяемом вершиной $p$ (рис. 2.2).

Понятие, с которым мы встретились здесь, — это *векторное доминирование* (см. также разд. 4.1.3). Говорят, что точка (вектор) **v** *доминирует* над **w**, тогда и только тогда, когда для всех индексов $i$ верно условие $v_i \geqslant w_i$. На плоскости точка **v** доминирует над **w** тогда и только тогда, когда **w** лежит в левом

[1]) Понятию «локус» соответствует устаревший термин «геометрическое место точек», употребляющийся сейчас лишь в учебной литературе. — *Прим. перев.*



нижнем квадранте, определяемом **v**. Итак, $Q(p)$ — число точек, над которыми доминирует $p$. Связь между доминированием и региональным поиском видна на рис. 2.3. Число точек

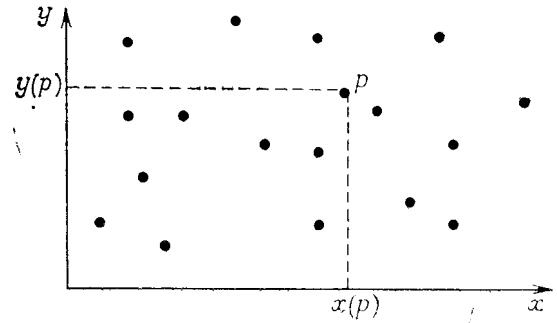



Рис. 2.2. Сколько точек «юго-западнее» $p$?

$N(p_1p_2p_3p_4)$ в прямоугольнике $p_1p_2p_3p_4$ определяется следующим образом:

$$N(p_1p_2p_3p_4) = Q(p_1) - Q(p_2) - Q(p_4) + Q(p_3). \qquad (2.1)$$

Это следует из комбинаторного правила включения — исключения [Liu (1968)]: над всеми точками прямоугольника, несомненно, доминирует вершина $p_1$. Нужно исключить такие точки, над которыми доминируют $p_2$ и $p_4$, но это приведет к тому, что часть точек будет исключена дважды, а именно те, над которыми доминируют и $p_2$, и $p_4$, но как раз эти точки лежат в левом нижнем квадранте вершины $p_3$.

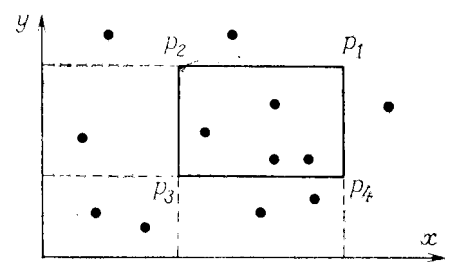



Рис. 2.3. Региональный поиск в форме четырех запросов о доминировании.

Итак, задача регионального поиска сведена к задаче обработки запросов о доминировании для четырех точек. Свойство, облегчающее эти запросы, заключается в том, что на плоскости существуют области удобной формы, внутри которых число доминирования $Q$ является константой.

Предположим, что из наших точек на оси $x$ и $y$ опущены перпендикуляры, а полученные линии продолжены в бесконечность. Они создают решетку из $(N+1)^2$ прямоугольников, как показано на рис.2.4.

Для всех точек $p$ в любом из таких прямоугольников (ячеек) $Q(p)$ — константа. Это означает, что доминантный поиск есть

просто ответ на вопрос: в какой ячейке прямоугольной решетки лежит заданная точка? На этот вопрос особенно легко ответить. После упорядочения исходных точек по обеим координатам останется только выполнить два двоичных поиска (по одному для каждой оси), чтобы найти ячейку, содержащую нашу точку. Значит, время запроса будет равным только $O(\log N)$. К сожалению, имеется $O(N^2)$ ячеек, поэтому будет нужна квадратичная память. Нужно, конечно, вычислить число доминирования

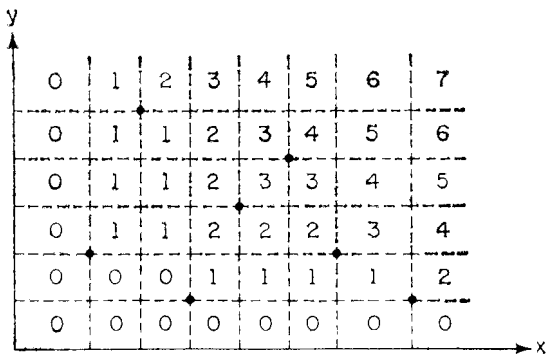



Рис. 2.4. Прямоугольная решетка для доминантного поиска.

для каждой ячейки. Это можно легко проделать для любой из ячеек за время $O(N)$, что приводит к общей затрате времени $O(N^3)$ на предобработку; однако для менее наивного подхода [1]) время предобработки можно снизить до $O(N^2)$.

Если кто-то сочтет затраты памяти и времени на предобработку для вышеизложенного метода чрезмерными, то он может

*Таблица 1*

| Запрос | Память | Предобработка | Примечание |
|---|---|---|---|
| $O(\log N)$ | $O(N^2)$ | $O(N^2)$ | Изложенный метод |
| $O(\log^2 N)$ | $O(N \log N)$ | $O(N \log N)$ | [2]) |
| $O(N)$ | $O(N)$ | $O(N)$ | Без предобработки |

[1]) См. работу по региональному поиску [Bentley, Shamos (1977)].

[2]) Этот результат, полученный также в [Bentley, Shamos (1977)], будет показан в разд 2.3.4.



исследовать другие подходы. Обычно эти исследования демонстрируют общее правило, известное из практики построения алгоритмов, а именно взаимозависимость между различными мерами эффективности. Задача регионального поиска не составляет исключения; в таблице I показаны затраты при комбинировании ресурсов.

Из предыдущих рассуждений вытекают две главные модели для задач геометрического поиска:

1. *Задачи локализации*, когда файл представляет собой разбиение геометрического пространства на области, а запрос является точкой. Локализация состоит в определении области, содержащей запрошенную точку.

2. *Задачи регионального поиска*, когда файл содержит набор точек пространства, а запрос есть некая стандартная геометрическая фигура, произвольно перемещаемая в этом пространстве (типичный запрос в 3-мерном пространстве это шар или брус). Региональный поиск состоит в извлечении (задачи *отчета*) или в подсчете числа (задачи *подсчета*) всех точек внутри запросного региона (области).

Хотя методы, используемые для решения этих двух типов задач, близки — мы только что видели, как региональный поиск может быть сведен к задаче о локализации точки, — удобно демонстрировать их отдельно. Остаток этой главы посвящен данному вопросу.

## 2.2. Задачи локализации точки

### 2.2.1. Общие соображения. Простые случаи

Задачи локализации точки можно также с полным основанием назвать *задачами о принадлежности точки*. В самом деле, утверждение «точка $p$ лежит в области $R$» синонимично утверждению «точка $p$ принадлежит области $R$». Трудоемкость решения этой задачи, безусловно, будет существенно зависеть от природы пространства и от способа его разбиения.

Для современного состояния вычислительной геометрии типично то, что плоская задача хорошо изучена, но весьма мало известно про случай $E^3$ и даже еще менее — про случаи большего числа измерений.

Даже в самом простом случае плоскости тип разбиения, на котором ведется поиск, определяет для этой задачи сложность, оцениваемую тремя основными мерами, указанными в предыдущем разделе. Рассмотрим сначала разбиения плоскости — или *планарные подразбиения*, как их обычно называют, — образованные прямолинейными отрезками; затем мы откажемся от этого допущения, которое выглядит ограничивающим. Однако

прежде, чем изложение будет продолжено, необходимо упомянуть об одной технической детали: рассматриваемые нами планарные подразбиения таковы, что после удаления из плоскости границ областей остающиеся открытые множества связны. Это нужно для того, чтобы пару точек, принадлежащих одной области, можно было соединить кривой, не содержащей граничных точек области.

Первой приходит в голову задача, в которой плоскость разбита на две области, одна из которых бесконечна, а другая является многоугольником $P$. Очевидно, что $P$ — *простой* многоугольник; это следует из условия разбиения плоскости на две области и из теоремы Жордана для многоугольников (см. разд. 1.3.1). Итак, можно сформулировать:

**Задачу П.2** (ПРИНАДЛЕЖНОСТЬ МНОГОУГОЛЬНИКУ). Даны простой многоугольник $P$ и точка $z$; определить, находится ли $z$ внутри $P$.

Повторим еще раз, что трудоемкость этой задачи зависит от того, обладает ли $P$ помимо простоты еще какими-нибудь полезными свойствами. Интуитивно выпуклый многоугольник выглядит более простым объектом. Поэтому рассмотрим:

**Задачу П.3** (ПРИНАДЛЕЖНОСТЬ ВЫПУКЛОМУ МНОГОУГОЛЬНИКУ). Даны выпуклый многоугольник $P$ и точка $z$. Находится ли $z$ внутри $P$?

Можно сразу же продемонстрировать ее решение для случая уникального запроса, причем этот результат будет справедлив и для невыпуклых многоугольников.

**Теорема 2.1.** *Принадлежность точки $z$ внутренней области простого $N$-угольника $P$ можно установить за время $O(N)$ без предобработки.*

*Доказательство.* Проведем через точку $z$ горизонталь $l$ (рис. 2.5). По теореме Жордана внешняя и внутренняя области $P$ хорошо определены. Если $l$ не пересекает $P$, то $z$ — внешняя точка. Поэтому пусть $l$ пересекает $P$, и рассмотрим вначале случай, когда $l$ не проходит ни через одну из вершин $P$. Пусть $L$ — число точек пересечения $l$ с границей $P$ слева от $z$. Поскольку $P$ ограничен, левый конец $l$ лежит вне $P$. Будем двигаться вдоль $l$ от $-\infty$ направо вплоть до $z$. На самом левом пересечении $l$ с границей $P$ мы попадем внутрь $P$, на следующем пересечении выйдем наружу и т. д. Поэтому $z$ лежит внутри тогда и только тогда, когда $L$ нечетно. Теперь рассмотрим вырожденный случай, когда $l$ проходит через вершины $P$. Бесконечно малый поворот $l$ вокруг $z$ против часовой стрелки не изменит классификации (внутри/вне) точки $z$, но устранит вырожденность. Итак, вообразив реализацию этого бесконечно малого поворота, мы увидим: если обе вершины ребра принадлежат $l$, то это ребро следует отбросить; если же ровно одна вершина ребра лежит на $l$, то пересечение будет учтено, когда эта вершина с большой



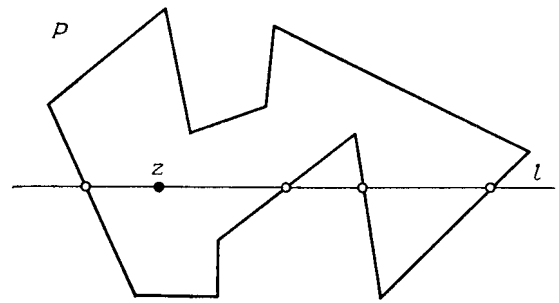



Рис. 2.5. Решение задачи о принадлежности точки простому многоугольнику при уникальном запросе. Есть только одно пересечение $P$ линией $l$ слева от $z$, поэтому $z$ внутри многоугольника.

ординатой, и игнорируется в противном случае. Резюмируя, получаем следующий алгоритм:

```
begin L := 0;
      for i := 1 until N do if ребро (i) не горизонтально
          then if (ребро (i) пересекает l нижним концом
                   слева от z)
                   then L := L + 1;
      if (L нечетно) then z внутри else z снаружи
end.
```

Очевидно, что этот простой алгоритм выполняется за время $O(N)$.

Для массовых запросов сначала рассмотрим случай, когда $P$ — выпуклый многоугольник. Предлагаемый метод использует выпуклость $P$, а именно свойство, что вершины выпуклого многоугольника упорядочены по полярным углам относительно любой внутренней точки. Такую точку $q$ можно легко найти; например, можно взять центр масс (центроид) треугольника, образованного любой тройкой вершин $P$. Теперь рассмотрим $N$ лучей, исходящих из точки $q$ и проходящих через вершины многоугольника $P$ (рис. 2.6).

Эти лучи разбивают плоскость на $N$ клиньев. Каждый клин разбит на две части одним из ребер $P$. Одна из этих частей лежит целиком внутри $P$, другая — целиком снаружи. Считая $q$ началом полярных координат, мы можем отыскать тот клин, где

лежит точка $z$, проведя один раз двоичный поиск, поскольку лучи следуют в порядке возрастания их углов. После нахождения клина остается только сравнить $z$ с тем единственным ребром из $P$, которое разрезает этот клин, и решить, лежит ли $z$ внутри $P$.

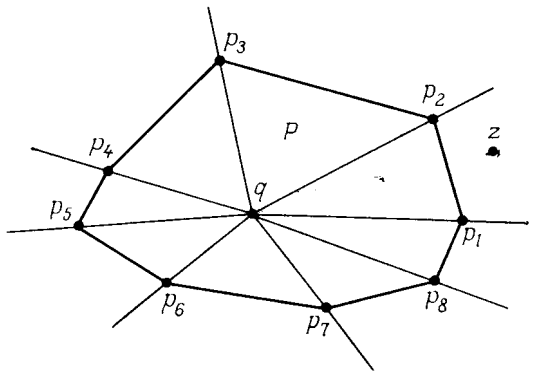



Рис. 2.6. Разбиение на клинья для задачи о принадлежности выпуклому многоугольнику. 1. Двоичным поиском находим, что $z$ лежит в клине $p_1qp_2$. 2. Определяя, по какую сторону ребра $p_1p_2$ лежит $z$, находим, что $z$ снаружи.

Предобработка в вышеописанном способе решения состоит в поиске $q$ как центра масс трех вершин из $P$ и размещении вершин $p_1, p_2, \ldots, p_N$ в структуре данных, пригодной для двоичного поиска (например, в векторе). Очевидно, это можно проделать за время $O(N)$ для заданной последовательности $p_1$, $p_2, \ldots, p_N$. Теперь сосредоточим внимание на процедуре поиска:

**Процедура** ПРИНАДЛЕЖНОСТЬ ВЫПУКЛОМУ МНОГОУГОЛЬНИКУ

1. Дана пробная точка $z$. Определяем методом двоичного поиска клин, в котором она лежит. Точка $z$ лежит между лучами, определяемыми $p_i$ и $p_{i+1}$, тогда и только тогда, когда угол $(zqp_{i+1})$ положительный, а угол $(zqp_i)$ отрицательный [1]).
2. Если $p_i$ и $p_{i+1}$ найдены, то $z$ — внутренняя точка тогда и только тогда, когда угол $(p_ip_{i+1}z)$ отрицательный.

[1]) Заметим, что определению знака угла $(p_1p_2p_3)$ соответствует вычисление определителя матрицы третьего порядка, образованной координатами этих точек. Конкретно, если положить $p_i = (x_i, y_i)$, то этот определитель, равный $\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$, дает удвоенную *ориентированную площадь* треугольника $(p_1p_2p_3)$, где знак плюс будет тогда и только тогда, когда обход $(p_1p_2p_3)$ ориентирован против часовой стрелки. Итак, $(p_1p_2p_3)$ соответствует левому повороту тогда и только тогда, когда этот определитель положителен.



**Теорема 2.2.** *Время ответа на запрос о принадлежности точки выпуклому $N$-угольнику равно $O(\log N)$ при затрате $O(N)$ памяти и $O(N)$ времени на предварительную обработку.*

Какое же свойство выпуклого многоугольника позволяет провести быстрый поиск? Чтобы можно было применить двоичный поиск, вершины должны быть упорядочены по углу относительно некоторой точки. Очевидно, что выпуклость является только достаточным условием обладания этим свойством. На самом же деле существует более обширный класс простых многоугольников, включающий в себя и выпуклые многоугольники, который обладает этим свойством: это класс *звездных* многоугольников (см. разд. 1.3.1). Действительно, звездный многоугольник $P$ (рис. 2.7) содержит по меньшей мере одну точку $q$ такую, что отрезок $qp_i$ лежит целиком внутри многоугольника $P$ для любой вершины $p_i$ из $P$, $i = 1, \ldots, N$.

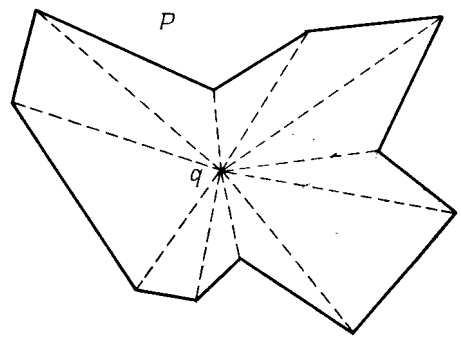



Рис. 2.7. Звездный многоугольник.

Для определения принадлежности точки звездному многоугольнику можно непосредственно использовать предыдущий алгоритм, если найден соответствующий центр $q$, служащий основой поиска. Множество подходящих центров внутри $P$ было определено в разд. 1.3.1 как *ядро* $P$. Построение ядра простого $N$-угольника лежит за рамками этой главы; мы вернемся к этой задаче в разд. 7.2.6, где доказан неожиданный результат, что ядро может быть найдено за время $O(N)$. А сейчас предположим, что ядро $P$ известно (и непусто); поэтому можно выбрать центр, относительно которого строятся клинья, необходимые для поиска. Получаем следующую теорему:

**Теорема 2.3.** *Время ответа на запрос о принадлежности точки звездному $N$-угольнику равно $O(\log N)$ при затрате $O(N)$ памяти и $O(N)$ времени на предварительную обработку.*

Теперь можно обратить внимание на простые многоугольники общего вида, которые будем называть обыкновенными. Существует иерархия свойств, строго упорядоченная отношением «быть подмножеством»:

$$\text{ВЫПУКЛОСТЬ} \subset \text{ЗВЕЗДНОСТЬ} \subset \text{ОБЫКНОВЕННОСТЬ}. \quad (2.2)$$

Мы только что видели, что задача о принадлежности звездному многоугольнику асимптотически ничуть не сложнее задачи о принадлежности выпуклому многоугольнику. Но что можно сказать об обыкновенном случае? Один из подходов к этой задаче подсказан тем, что каждый простой многоугольник есть объединение некоторого числа многоугольников специального вида — таких, как звездные или выпуклые, или в конечном итоге треугольников. К сожалению, минимальная мощность $k$ такой декомпозиции может сама оказаться равной $O(N)$ [1]). Поэтому данный подход сводится к преобразованию простого $N$-угольника в $N$-вершинный плоский граф. В связи с этим кажется, что задача принадлежности обыкновенному простому многоугольнику не легче, чем, по-видимому, более общая задача о локализации точки в планарном подразбиении, хотя и неизвестно доказательство их эквивалентности. Поэтому мы сосредоточимся на последней из этих задач.

### 2.2.2. Локализация точки на планарном подразбиении

В главе 1 упоминалось, что планарный граф всегда может быть уложен на плоскости так, что его ребра перейдут в прямолинейные отрезки. Графы, уложенные подобным образом, будут называться *плоскими прямолинейными графами* (ППЛГ). Любой ППЛГ определяет, вообще говоря, подразбиение плоскости; если в ППЛГ нет вершин со степенью $<2$, то можно непосредственно убедиться в том, что все ограниченные области этого подразбиения — простые многоугольники. Без потери общности здесь и далее граф предполагается связным.

Чтобы вести поиск в подобной структуре, можно попытаться разбить каждую область на такие многоугольники, для которых поисковая операция относительно проста. Не принимая во внимание на некоторое время сложности получения желаемой декомпозиции — это, безусловно, может оказаться не простым делом, — заметим, что успех в локализации точки зависит от возможности быстро сузить множество компонент декомпозиции, которые следует просмотреть. С другой стороны, скорейшие из известных методов поиска основаны на *дихотомии*, иначе говоря, на двоичном поиске. Легко убедиться после небольшого размышления, что с точки зрения затрат времени на запрос возможность использовать двоичный поиск важнее, чем минимизация размера исследуемого множества, в связи с логарифмической зависимостью времени запроса от этого размера. Отсюда

[1]) Это очевидно, если компонентами разбиения являются треугольники, поскольку $k = N - 3$. Но даже для звездных компонент $k$ может достигать величины $\lceil N/3 \rceil$ [Chvatal (1975)].



видно, что потенциально успешный метод локализации точки должен обладать следующими чертами: исходное планарное подразбиение должно преобразовываться в новое, такое, что каждый из составляющих его многоугольников пересекает небольшое и ограниченное число (возможно, ровно одну) исходных областей, и, кроме того, такое, что к нему применим двоичный поиск. Другими словами, главная идея состоит в том, чтобы *создать новые геометрические объекты, допускающие двоичный поиск*, и в таком виде она была сформулирована Добкином и Липтоном [Dobkin, Lipton (1976)]. Все известные методы, описываемые ниже, по существу, пронизаны этой идеей.

#### 2.2.2.1. Метод полос

Пусть задан ППЛГ $G$. Проведем горизонтальные прямые через каждую его вершину, как показано на рис. 2.8. Они разделяют плоскость на $N + 1$ горизонтальных полос. Если провести сортировку этих полос по координате $y$ как часть предобработки, то появится возможность найти ту полосу, в которой лежит пробная точка $z$, за время $O(\log N)$.

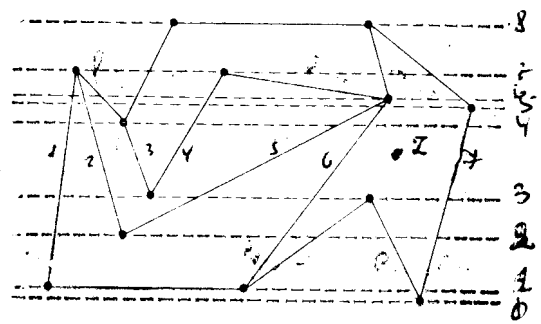



Рис. 2.8. Вершины ППЛГ определяют горизонтальные полосы.

Теперь рассмотрим пересечение одной из полос с графом $G$. Оно состоит из отрезков ребер графа $G$. Эти отрезки определяют трапеции [1]). Поскольку $G$ есть плоская укладка планарного графа, то его ребра пересекаются между собой только в вершинах, а так как каждая вершина лежит на границе полосы, то отрезки ребер внутри полос не пересекаются (рис. 2.9).

Поэтому данные отрезки можно полностью упорядочить слева направо и использовать двоичный поиск для определения той трапеции, в которую попала точка $z$, потратив на это $O(\log N)$ времени. Отсюда получаем оценку времени запроса для худшего случая, равную $O(\log N)$.

[1]) Такие трапеции, очевидно, могут вырождаться в треугольники, как показано на рис. 2.9.

Осталось только узнать, сколько ресурсов ушло на предварительную подготовку и запоминание ППЛГ. При наивном подходе кажется, что нужно сортировать все отрезки внутри каждой полосы. Кроме того, поскольку каждая полоса может содер-

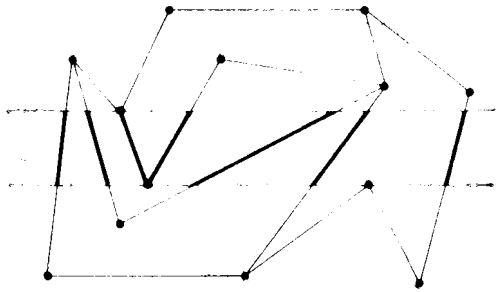


Рис. 2.9. Отрезки ребер графа внутри полосы не пересекаются.

жать $O(N)$ таких отрезков, то кажется, что должны потребоваться затраты: $O(N^2 \log N)$ — для времени и $O(N^2)$ — для памяти. Теперь покажем, как сократить время предобработки до $O(N^2)$. Однако (в данном алгоритме) ничего нельзя сделать для сокращения памяти, поскольку существуют такие ППЛГ, которые требуют квадратичной памяти (рис. 2.10).

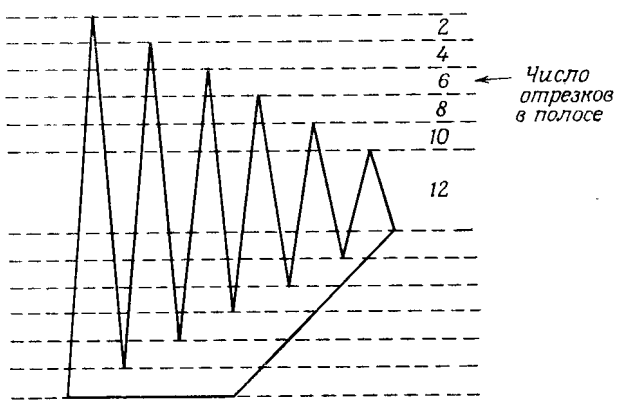



Рис. 2.10. Суммарное число отрезков в полосах может достигать $O(N^2)$.

Заметим, что если ребро ППЛГ проходит через несколько полос, то эти полосы следуют одна за другой. Это наблюдение и является тем ключом, позволяющим сократить время предобработки, поскольку можно воспользоваться методом заметания плоскости (см. разд. 1.2.2). Напомним, что алгоритм плоского заметания характеризуется двумя основными структурами данных: (1) *списком точек событий*, т. е. последовательностью



позиций, назначаемых для заметающей прямой; (2) *статусом заметающей прямой*, т. е. описанием пересечения заметающей прямой с заметаемым геометрическим объектом. В нашем случае, если плоское заметание проводить, скажем, снизу вверх, то моментальным статусом заметающей прямой будет упорядоченная слева направо последовательность ребер графа, пересекающих ту полосу, где находится заметающая прямая. Эта последовательность, т. е. порядок ребер слева направо, не изменяется внутри данной полосы, но изменяется на границе со следующей полосой, когда достигается $v$ — новая вершина ППЛГ. Ясно, что в этой точке рёбра, оканчивающиеся в $v$, удаляются и заменяются теми, которые начинаются в $v$. Статус заметающей прямой можно хранить в форме дерева, сбалансированного по высоте (т. е. 2-3 дерева), которое, как хорошо известно, допускает реализацию операций ВСТАВИТЬ и УДАЛИТЬ за время, пропорциональное логарифму его размера. Добавим, что заметающая прямая сканирует полосы в восходящем порядке, т. е. список точек событий — это просто перечисленные снизу вверх вершины ППЛГ. В каждой точке события статус заметающей прямой корректируется и запоминается; ясно, что этот статус есть просто последовательность отрезков внутри вышележащей полосы. С точки зрения затраты ресурсов работа состоит во вставлении и удалении каждого из ребер графа — со стоимостью $O(\log N)$ на одну операцию — и в запоминании статуса; на первое уйдет $O(N \log N)$ времени, поскольку по теореме Эйлера в $N$-вершинном планарном графе $O(N)$ ребер, а последнее потребует $O(N^2)$ памяти, как указано выше. Алгоритм предобработки будет интуитивно понятнее, если считать ребра ориентированными снизу вверх. Вершины, упорядоченные по возрастанию $y$, запоминаются в массиве ВЕРШИНА; $B[i]$ — множество ребер, инцидентных ВЕРШИНЕ[$i$] снизу (входящие ребра) и упорядоченных по углу против часовой стрелки; $A[i]$ — множество ребер, исходящих из ВЕРШИНЫ($i$) и упорядоченных по часовой стрелке. Ребра запоминаются в сбалансированном по высоте дереве $L$. Предлагается следующий алгоритм:

```
        procedure ПРЕДОБРАБОТКА-ДЛЯ-ЛОКАЛИЗАЦИИ-
                  ТОЧКИ-НА-ПЛОСКОСТИ
begin   ВЕРШИНА[1 : 2N] := вершины G, упорядоченные по
        возрастанию y;
        L := ∅;
        for i := 2 until N do
            begin УДАЛИТЬ(B[i]) из L;
                  ВСТАВИТЬ(A[i]) в L;
                  Запомнить L
            end
end.
```

Итак, сформулируем теорему:

**Теорема 2.4.** *Локализацию точки в $N$-вершинном планарном подразбиении можно реализовать за время $O(\log N)$ с использованием $O(N^2)$ памяти, если $O(N^2)$ времени ушло на предобработку.*

Хотя данный метод обладает оптимальной оценкой времени запроса, однако его время предобработки и — в еще большей степени — его затраты памяти чрезмерны и для многих приложений недопустимы. Было высказано предположение [Shamos (1975a)], что затраты памяти можно сократить до $O(N)$, хотя это может потребовать увеличения времени запроса до $O(\log^2 N)$. Вскоре эта гипотеза подтвердилась, что и будет показано ниже. Этот почти оптимальный метод стал основой для дальнейших существенных продвижений [Edelsbrunner, Guibas, Stolfi (1985)].

### 2.2.2.2. Метод цепей

В то время как в методе полос эффективность поиска достигается благодаря декомпозиции исходного подразбиения на трапеции, в методе цепей, предложенном Ли и Препаратой [Lee, Preparata (1978)], эта же цель достигается благодаря использованию в качестве базовой компоненты нового типа многоугольников, называемых *монотонными* и определяемых ниже.

Ключевым в этом методе является определяемое здесь понятие «цепи».

**Определение 2.1.** *Цепью* $C = (u_1, \ldots, u_p)$ называется ППЛГ с вершинами $\{u_1, \ldots, u_p\}$ и ребрами $\{(u_i, u_{i+1}): i = 1, \ldots, p-1\}$.

Другими словами, цепью в силу данного определения является плоская укладка теоретико-графовой цепи, которую иногда называют также *ломаной линией* (рис. 2.11(a)).

Рассмотрим планарное подразбиение, определяемое ППЛГ $G$. Предположим, что в $G$ найдена цепь $C$ (подграф $G$) одного из следующих типов: (1) $C$ является циклом или (2) оба конца $u_1$ и $u_p$ из $C$ лежат на границе бесконечной области. В последнем случае дополним $C$ с обоих концов полубесконечными параллельными ребрами. Теперь цепь каждого типа делит исходное подразбиение на две части. Более того, мы скоро увидим, что процедура определения того, по какую сторону от $C$ лежит пробная точка $z$, именуемая процедурой *дискриминации $z$ относительно $C$*, довольно проста. Далее, если удастся выбрать $C$ так, чтобы указанные части были сравнимы по сложности, то, рекурсивно повторяя деления, мы получим метод локализации точки, который требует логарифмического числа дискриминаций. Поэтому сразу же возникают два вопроса: (1) Какова стоимость процедуры дискриминации точки относительно произвольной цепи и существуют ли такие цепи, дискриминация относительно которых проста? (2) Какова трудоемкость поиска подходящей разделяющей цепи?



Рассматривая первый вопрос, легко убедиться, что дискриминация относительно произвольной цепи есть, по существу, та же

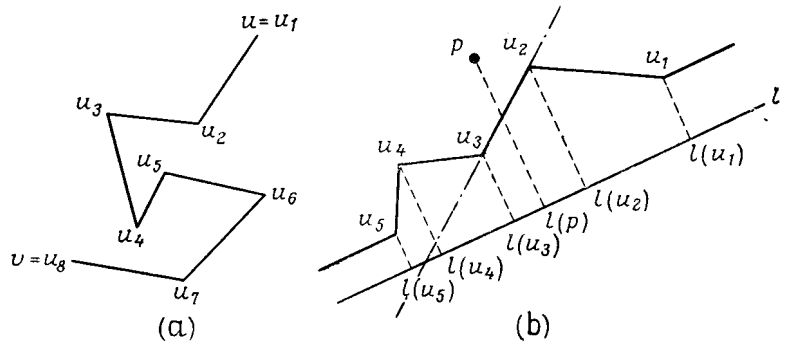



Рис. 2.11. Примеры цепей: (a) — общего вида; (b) — монотонная по отношению к прямой $l$.

самая задача, что и проверка принадлежности обыкновенному многоугольнику. Поэтому необходимо поискать более ограниченный класс цепей. Один из таких классов задается следующим определением.

**Определение 2.2.** Цепь $C = (u_1, \ldots, u_p)$ называется *монотонной* по отношению к прямой $l$, если любая прямая, ортогональная к $l$, пересекает $C$ ровно в одной точке.

Другими словами, любая монотонная цепь $C$ принадлежит к определенному выше типу (2), а ортогональные проекции $\{l(u_1), \ldots, l(u_p)\}$ вершин $C$ на $l$ упорядочены вдоль $l$ следующим образом: $l(u_1), \ldots, l(u_p)$.

Одна из подобных цепей показана на рис. 2.11(b). Очевидно, что проекцию $l(z)$ заданной пробной точки $z$ на $l$ можно локализовать методом двоичного поиска в единственном из интервалов $(l(u_i), l(u_{i+1}))$. Затем единственная линейная проверка покажет, по какую сторону от прямой, несущей ребро $u_iu_{i+1}$, лежит пробная точка $z$. Поэтому дискриминация произвольной точки относительно $p$-вершинной цепи реализуется за время $O(\log p)$ [1]).

[1]) Заметим, что монотонную цепь можно считать предельным случаем звездного многоугольника.

Такая эффективность побуждает использовать монотонные цепи при локализации точек. Предположим, что существует некое множество $\mathscr{C} = \{C_1, \ldots, C_r\}$ цепей, монотонных относительно одной и той же прямой $l$ и обладающих следующими дополнительными свойствами:

**Свойство 1.** *Объединение всех элементов $\mathscr{C}$ содержит заданный ППЛГ $G$*[1]).

**Свойство 2.** *Для любой пары цепей $C_i$ и $C_j$ из $\mathscr{C}$ те узлы $C_i$, которые не являются узлами $C_j$, лежат по одну сторону от $C_j$.*

Такое множество $\mathscr{C}$ называется *полным множеством монотонных цепей графа $G$*. Заметим, что, согласно свойству 2, цепи из полного множества упорядочены. Следовательно, можно применить к $\mathscr{C}$ метод дихотомии (т. е. двоичный поиск), в котором элементарной операцией вместо простого сравнения чисел теперь будет дискриминация точки относительно цепи. Итак, если у нас $r$ цепей в $\mathscr{C}$ и в самой длинной цепи $p$ вершин, то поиск займет в наихудшем случае $O(\log r \cdot \log p)$ времени.

Но остается самый важный вопрос: можно ли построить полное множество монотонных цепей для произвольного ППЛГ $G$? Ответ на этот вопрос отрицательный. Однако мы покажем, что ППЛГ допускает построение полного множества монотонных цепей, если удовлетворяет довольно слабому ограничению. Кроме того, мы покажем, что произвольный ППЛГ можно легко преобразовать в такой граф, к которому применима процедура построения цепей. Эта процедура создает несколько новых «искусственных» областей, которые не мешают, однако, эффективному решению задачи локализации точки.

Следующее определение выражает упомянутое выше слабое ограничение:

**Определение 2.3.** Пусть $G$ — ППЛГ с множеством вершин $\{v_1, \ldots, v_N\}$, где вершины индексированы так, что $i < j$ тогда и только тогда, когда $y(v_i) < y(v_j)$, или $y(v_i) = y(v_j)$ и $x(v_i) > x(v_j)$. Вершина $v_j$ называется *регулярной*, если существуют такие целые $i < j < k$, что $(v_i, v_j)$ и $(v_j, v_k)$ — ребра $G$. Говорят, что ППЛГ $G$ *регулярен*, если каждая его вершина $v_j$ регулярна при $1 < j < N$ (т. е. за исключением двух крайних вершин $v_1$ и $v_N$).

На рис. 2.12 приведен пример регулярного графа. Сейчас мы покажем, что регулярный граф распадается на полное множество цепей, монотонных относительно оси $y$. (Далее прямая $l$ будет считаться осью $y$ на плоскости.)

[1]) Заметим, что конкретное ребро $G$ может принадлежать более чем одной цепи из $\mathscr{C}$.



Чтобы подтолкнуть интуицию, представим, что ребро $(v_i, v_j)$ ориентировано от $v_i$ к $v_j$, если $i < j$. Поэтому можно говорить о множествах $\mathrm{IN}(v_j)$ и $\mathrm{OUT}(v_j)$ соответственно входящих и исходящих ребер для вершины $v_j$. Предположим, что ребра в $\mathrm{IN}(v_j)$ упорядочены по углу против часовой стрелки, а ребра в $\mathrm{OUT}(v_j)$ упорядочены по часовой стрелке. По предположению о регулярности оба эти множества не пусты для каждой из некрайних вершин. Отсюда видно, что для любой $v_j$ $(j \neq 1)$ можно построить $y$-монотонную цепь от $v_1$ до $v_j$. Это тривиально для $j = 2$. Допустим, что данное утверждение верно для всех $k < j$. Поскольку $v_j$ регулярна, то по определению 2.3 существует такое $i < j$, что $(v_i, v_j)$ — ребро $G$. Но по предположению индукции существует цепь $C$ от $v_1$ до $v_i$, монотонная относительно оси $y$; ясно, что сцепление $C$ с $(v_i, v_j)$ — тоже монотонная цепь. Для завершения доказательства необходимо показать, что выполнены свойства 1 и 2, упомянутые выше. Пусть *вес* $W(e)$ ребра $e$ — это число цепей, которым принадлежит $e$. Кроме того, введем следующие обозначения:

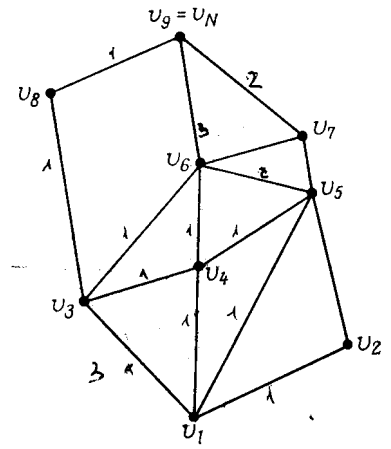



Рис. 2.12. Пример регулярного ППЛГ.

$$W_{\mathrm{IN}}(v) = \sum_{e \in \mathrm{IN}(v)} W(e); \quad W_{\mathrm{OUT}}(v) = \sum_{e \in \mathrm{OUT}(v)} W(e).$$

Тогда надо только показать, что веса ребер можно выбрать такими, что:

(1) каждое ребро обладает положительным весом;
(2) для любого $v_j$ $(j \neq 1, N)$, $W_{\mathrm{IN}}(v_j) = W_{\mathrm{OUT}}(v_j)$.

Условие (1) устанавливает, что каждое ребро принадлежит по крайней мере одной цепи (т. е. свойство 1), а условие (2) гарантирует, что $W_{\mathrm{IN}}(v_j)$ цепей проходят через $v_j$ и их можно выбрать так, чтобы они не пересекались (свойство 2). Реализация условия $W_{\mathrm{IN}} = W_{\mathrm{OUT}}$ является решением достаточно известной классической потоковой задачи [Even (1979)] и может быть достигнуто за два прохода по графу $G$. Полагая $W(e) = 1$ для каждого ребра $e$, в первом проходе — от $v_1$ до $v_N$ — мы получим $W_{\mathrm{IN}}(v_i) \leqslant W_{\mathrm{OUT}}(v_i)$ для всех некрайних $v_i$. При втором проходе — от $v_N$ до $v_1$ — мы получим $W_{\mathrm{IN}}(v_i) \geqslant W_{\mathrm{OUT}}(v_i)$, т. е. иско-

мое выражение. Обозначая $v_{IN}(v) = |IN(v)|$ и $v_{OUT}(v) = |OUT(v)|$, имеем алгоритм

**procedure** БАЛАНСИРОВКА-ПО-ВЕСУ-В-РЕГУЛЯРНОМ-ППЛГ
**begin for** каждого ребра $e$ **do** $W(e) := 1$ (* инициализация *)
**for** $i := 2$ **until** $N - 1$ **do**
**begin** $W_{IN}(v_i) :=$ сумма весов ребер, входящих в $v_i$;
$d_1 :=$ крайнее слева ребро, исходящее из $v_i$;
**if** $(W_{IN}(v_i) > v_{OUT}(v_i))$ **then** $W(d_1) := W_{IN}(v_i) - v_{OUT}(v_i) + 1$
**end** (* первый проход *);
**for** $i := N - 1$ **until** 2 **do**
**begin** $W_{OUT}(v_i) :=$ сумма весов ребер, исходящих из $v_i$;
$d_2 :=$ крайнее слева ребро, входящее в $v_i$;
**if** $(W_{OUT}(v_i) > W_{IN}(v_i))$ **then** $W(d_2) := W_{OUT}(v_i) - W_{IN}(v_i) + W(d_2)$
**end** (* второй проход *)
**end.**

Очевидно, что этот алгоритм требует времени, линейно зависящего от числа ребер и вершин графа $G$. Данную процедуру можно изменить так, чтобы построение $\mathscr{C}$, т. е. назначение ребер цепям, проводилось при реализации второго прохода. Для краткости опустим эти детали и покажем на рис. 2.13(a) — (d) конфигурацию веса ребер после инициализации, первого и второго проходов соответственно и чертеж цепей $\mathscr{C}$ для ППЛГ $G$ с рис. 2.12. Этим заканчивается доказательство того, что регулярный ППЛГ покрывается полным множеством монотонных цепей.

Перед проведением анализа работы этого алгоритма важно показать, как произвольный ППЛГ преобразуется в регулярный. Рассмотрим такую нерегулярную вершину $v$ графа $G$, которая не имеет, скажем, исходящих ребер (рис. 2.14). Горизонтальная прямая, проходящая через $v$, протыкает в общем случае пару ребер $e_1$ и $e_2$ графа $G$, ближайших к $v$ слева и справа. (Заметим, что, поскольку $v$ — некрайняя вершина, по меньшей мере один из этих проколов должен существовать.) Пусть $v_i$ — верхняя вершина ребер $e_i$ $(i = 1, 2)$, и пусть $v^*$ — та из вершин $\{v_1, v_2\}$, которая имеет меньшую ординату (например, $v_2$). Тогда отрезок $vv^*$ не пересекает ребер $G$ [1]) и, следовательно, может быть добавлен к ППЛГ, тем самым «регуляризуя» вершину $v$. Данное наблюдение позволяет использовать метод, именуемый вертикальным плоским заметанием (см. разд. 1.2.2). А именно заметаем граф сверху вниз, чтобы регуляризовать вершины, не имеющие исходящих ребер, а затем снизу вверх, чтобы регуляризовать вершины другого типа. В первом случае списком точек событий является последовательность вершин $v_N, v_{N-1}, \ldots, v_1$. Структура статуса заметающей прямой (которая будет реализована деревом, сбалансированным по высоте) задается упорядоченным слева направо списком номеров ребер ППЛГ, пересекающихся с этой заметающей прямой, и, кроме того, с каждым интервалом между этими ребрами связана одна вершина (с минимальной ординатой в этом интервале). В процессе заметания для каждой встреченной вершины $v$ реализуем следующие операции: (1) локализуем $v$ (по абсциссе) в одном из интервалов в структуре данных статуса; (2) корректируем эту структуру статуса; (3) если $v$ нерегулярна, добавляем ребро от $v$ до той вершины, которая связана с интервалом, определенным в операции (1). Построение более формального алгоритма мы оставляем для самостоятельного упражнения. Заметим, что *регуляризация* $N$-вершинного ППЛГ осуществима за время $O(N \log N)$ благодаря возможности начальной сортировки ординат его вершин и локализации $N$ вершин в структуре данных

[1]) Это утверждение, вообще говоря, неверно. Контрпример получается, если другая нерегулярная вершина $v_3$, не имеющая *входящих* ребер, находится внутри треугольника $(v, v_2, v_4)$. Здесь $v_4$ — точка пересечения ребра $e_2$ с горизонтальной прямой, проходящей через точку $v$ (см. рис. 2.14). Однако это наблюдение не опровергает теорему 2.5, изложенную ниже. Алгоритм регуляризации нуждается при этом в небольшом изменении. — *Прим. перев.*



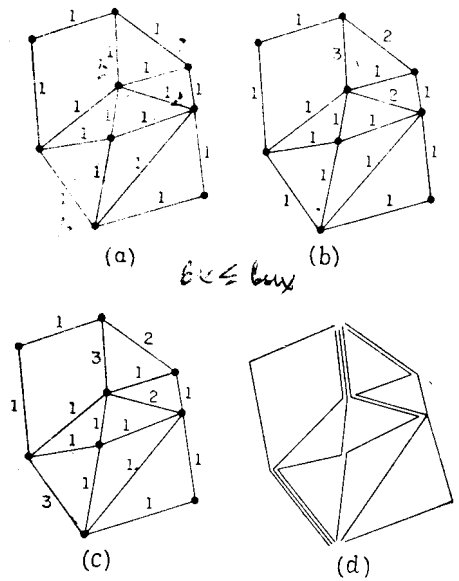



Рис. 2.13. Построение множества $\mathscr{C}$ для ППЛГ с рис. 2.12. Метки ребер являются их весами после: (a) инициализации; (b) первого прохода; (c) второго прохода. Полученное множество цепей изображено на (d).

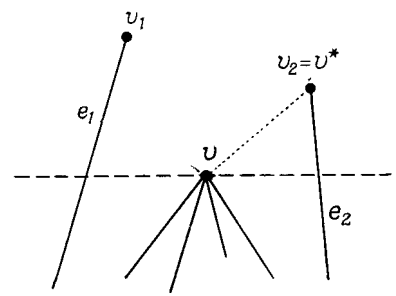



Рис. 2.14. Пример типичной нерегулярной вершины $v$.

статуса за время $O(\log N)$ для каждой вершины. Сформулируем этот факт как теорему для будущих ссылок.

**Теорема 2.5.** *$N$-вершинный ППЛГ можно регуляризовать за время $O(N \log N)$ с затратой $O(N)$ памяти.*

На рис. 2.15 показан результат регуляризации ППЛГ из рис. 2.8.

Теперь, после того как мы определили, что метод цепей применим к любому ППЛГ, — возможно, после его регуляризации — проанализируем его эффективность. Верхняя оценка времени поиска для наихудшего случая $O(\log p \cdot \log r) = O(\log^2 N)$ уже была установлена. Причем эта оценка достижима, поскольку существует $N$-вершинный ППЛГ с $O(\sqrt{N})$ цепями, каждая с $O(\sqrt{N})$ ребрами (рис. 2.16(a) для $N = 16$).

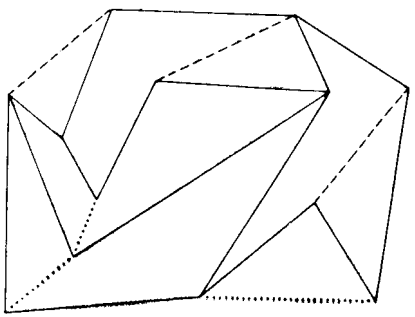


Рис. 2.15. Регуляризованный ППЛГ. Штриховыми линиями изображены ребра, добавленные при заметании сверху вниз, а пунктирными — при заметании снизу вверх.

Рассмотрим ППЛГ, изображенный на рис. 2.16(b). Этот граф содержит в своем полном множестве $N/2$ цепей, в каждой из которых $N/2$ ребер. На первый взгляд этот пример внушает беспокойство, так как в отношении затрат памяти данный метод, казалось бы, демонстрирует обескураживающую зависимость $O(N^2)$! Однако дела совсем не так плохи, если обратить внимание на то, как в алгоритме используются цепи. Безусловно, цепи используются в схеме двоичного поиска. Алгоритм двоичного поиска на полностью упорядоченном множестве $S$ индуцирует естественную иерархию на $S$, представленую двоичным деревом с корнем; процесс поиска соответствует проходу в этом дереве от корня к листу. Если естественным образом пронумеровать цепи, скажем, слева направо, то ребро $e$, принадлежащее более чем одной цепи, будет принадлежать всем элементам множества (интервалу) последовательных цепей. Теперь предположим, что цепи сопоставлены узлам дерева двоичного поиска. Если ребро $e$ принадлежит нескольким цепям какого-то интервала, то существует единственный элемент $C^*$ в этом интервале, который является общим предком [1]) для всех остальных элементов из этого

[1]) Согласно стандартной терминологии, вершина $v_i$ в корневом дереве $T$ является *предком* или *предшественником* вершины $v_j$, если существует путь от корня $T$, содержащий и $v_i$, и $v_j$, причем $v_i$ ближе к корню.



интервала в дереве поиска. Пусть $C$ — любой из этих остальных элементов; тогда дискриминации точки $z$ относительно $C$ пред-

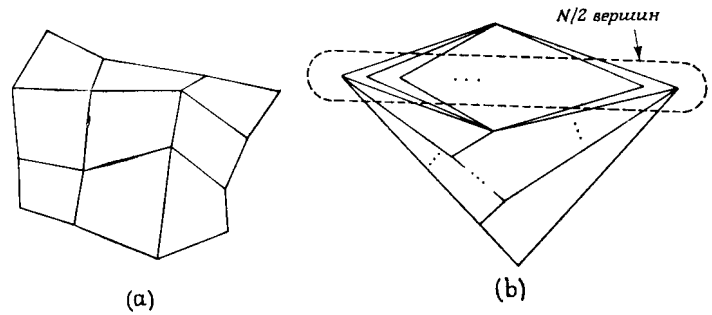



Рис. 2.16. Примеры худших случаев ППЛГ.

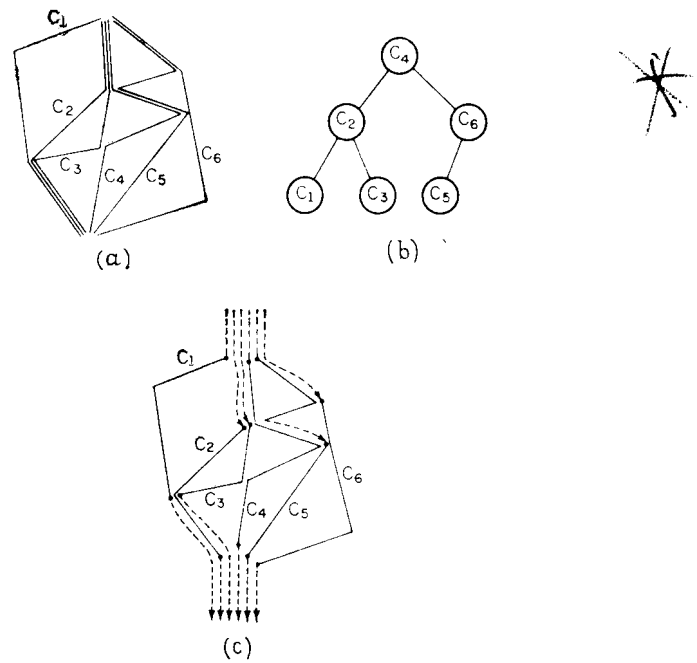



Рис. 2.17. Полное множество монотонных цепей (a), на котором ведется поиск в соответствии с иерархией, выраженной двоичным деревом (b). Обходные указатели показаны штриховыми линиями (c).

шествует — в схеме двоичного поиска — дискриминация $z$ относительно $C^*$. Следовательно, ребро $e$ можно отнести только к цепи $C^*$, и на самом деле оно будет отнесено к самой верхней в иерархии цепи из числа тех, которым оно принадлежит. При-

мер такого отнесения иллюстрирует рис. 2.17. Отметим наличие обходных указателей на рис. 2.17(c); однако их число не превосходит числа ребер, отсюда следует, что полную структуру данных для поиска можно разместить в $O(N)$ ячейках памяти.

Что же касается предобработки, то по-прежнему предположим, что ППЛГ задан в виде структуры данных РСДС, описанной в разд. 1.2.3.2. Поскольку кольцо ребер, инцидентных любой вершине $v$ графа $G$, можно получить за время, линейно связанное с ее степенью, то построение множеств $\mathrm{IN}(v)$ и $\mathrm{OUT}(v)$ для всех $v$ осуществимо за время $O(N)$.

Процедура балансирования по весу, как мы видели, также проходит за время $O(N)$. Отмечая, что возможный предварительный регуляризующий проход потребует еще $O(N \log N)$ времени, заключаем, что задача предобработки займет $O(N \log N)$ времени. Резюмируем:

**Теорема 2.6.** *Локализацию точки в $N$-вершинном планарном подразбиении можно реализовать за время $O(\log^2 N)$ с использованием $O(N)$ памяти при затратах $O(N \log N)$ времени на предобработку.*

В начале разд. 2.2.2.2 было отмечено, что в методе цепей используется новый тип простого многоугольника. Определим эти многоугольники:

**Определение 2.4.** Простой многоугольник называется *монотонным*, если его границу можно разбить на две цепи, монотонные относительно одной и той же прямой линии.

Монотонные многоугольники являются весьма интересными геометрическими объектами. Из результатов данного раздела непосредственно следует, что задачу о принадлежности точки монотонному многоугольнику с $N$ вершинами можно решить за время $O(\log N)$; позднее мы увидим, что и триангуляцию монотонного многоугольника, и определение факта монотонности простого многоугольника можно осуществить за оптимальное время $O(N)$.

### 2.2.2.3. Оптимальные методы: метод планарного сепаратора и метод детализации триангуляции

Существует ли метод со временем поиска $O(\log N)$, использующий менее чем квадратичную память? Эта знаменитая задача довольно долго не была решена. Она получила положительное решение благодаря замечательной конструкции, принадлежащей Липтону и Тарьяну [Lipton, Tarjan (1977a, 1977b, 1980)]. Следует обратить внимание на слова «существует ли метод». В действительности этот метод столь громоздок, а оценки



его эффективности столь откровенно и щедро опираются на «большое $O$», что Липтон и Тарьян сами высказали следующее предостережение [Lipton, Tarjan (1977b)]: «Мы не считаем этот алгоритм практичным, но его существование заставляет думать, что может найтись практичный алгоритм с временно́й оценкой $O(\log N)$ и с оценкой памяти $O(N)$». Описание их удивительно искусного метода выходит за рамки данной книги. Вместо этого рассмотрим два недавно предложенных оптимальных метода, дающих ответы на ожидания Липтона и Тарьяна. Первый метод — детализация триангуляции Киркпатрика [Kirkpatrick

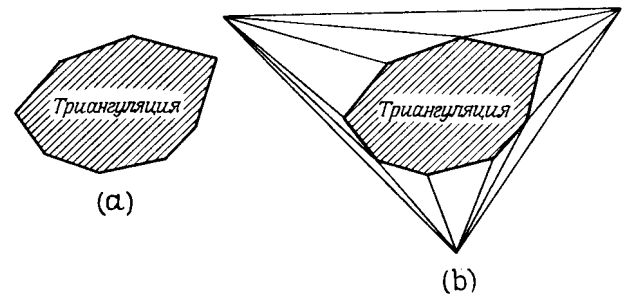



Рис. 2.18. Заданная триангуляция (a) и та же триангуляция внутри охватывающего ее треугольника (b).

(1983)] — может стать практичным алгоритмом; он описан в данном разделе. Второй метод [Edelsbrunner, Guibas, Stolfi (1985)] является дальнейшим улучшением метода цепей и будет кратко описан в разд. 2.4.

Метод «детализации триангуляции» принадлежит Киркпатрику. Предполагается, что $N$-вершинный ППЛГ является триангуляцией (см разд. 1.3.1); в том случае, если он не триангуляция, его можно преобразовать в нее за время $O((N \log N)$ с помощью простого алгоритма, который будет описан в разд. 6.2.2. Напомним, что триангуляция на множестве вершин $V$ на плоскости есть ППЛГ с максимальным числом ребер, не превышающим $3|V|-6$ (по формуле Эйлера) Кроме того, по причинам, которые скоро станут ясными, триангуляцию удобно окружить треугольной границей путем построения охватывающего ее треугольника и триангулирования области между этими двумя объектами (рис. 2.18). С учетом этого замечания все триангуляции будут считаться обладающими границей с тремя вершинами и ровно с $3|V|-6$ ребрами.

Пусть задана $N$-вершинная триангуляция $G$, и предположим, что строится *последовательность триангуляций* $S_1, S_2, \ldots, S_{h(N)}$, $S_1 = G$, а $S_i$ получается из $S_{i-1}$ по следующим правилам:

*Шаг* (1). Удалим некоторое множество независимых (т. е. несмежных) неграничных вершин $S_{i-1}$ и инцидентные им ребра. (От выбора этого множества, который будет описан ниже, самым непосредственным образом зависит эффективность алгоритма.)

*Шаг* (2). Вновь триангулируем многоугольники, образовавшиеся в результате удаления вершин и ребер.

Таким образом, $S_{h(N)}$ не имеет внутренних вершин (т. е. состоит из одного треугольника).

Заметим, что все триангуляции $S_1$, $S_2$, ..., $S_{h(N)}$ имеют одну общую границу, поскольку на шаге (1) мы удаляем только внутренние вершины. Треугольники, главные объекты этого метода, будем обозначать буквой $R$, снабжая ее индексами. Треугольник $R_j$ может появляться во многих триангуляциях, однако условимся, что $R_j$ *принадлежит триангуляции* $S_i$ (обозначая $R_j \in {}^* S_i$), если $R_j$ создан на шаге (2) при построении $S_i$.

Теперь построим структуру данных $T$ для поиска; узлам этой структуры соответствуют треугольники. (Для краткости узел часто будет называться «треугольник $R_j$» вместо «узел, представляющий треугольник $R_j$».) Структура $T$, топологию которой представляет направленный ациклический граф, определяется следующим образом: от треугольника $R_k$ к треугольнику $R_j$ проводится дуга, если при построении $S_i$ после $S_{i-1}$ мы имеем:

1) $R_j$ удаляется из $S_{i-1}$ на шаге (1);
2) $R_k$ создается в $S_i$ на шаге (2);
3) $R_j \cap R_k \neq \varnothing$.

Очевидно, что треугольники из $S_1$ не имеют исходящих дуг в $T$, и только они обладают этим свойством.

Для ясности удобно изобразить $T$ в рассмотренном виде, т. е. помещая его узлы в горизонтальные строки, каждая из которых соответствует какой-нибудь триангуляции. Последовательность триангуляций показана на рис. 2.19(a); кружком обведены вершины, которые удалены на данном шаге. Соответствующая структура $T$ показана на рис. 2.19(b).

После построения $T$ легко понять, как происходит локализация точки. Элементарной операцией является «принадлежность треугольнику», которая, очевидно, выполняется за время $O(1)$. Начальный шаг состоит в локализации пробной точки $z$ относительно $S_{h(N)}$. Затем идем вниз по пути в $T$ и неизбежно остановимся в одном из треугольников, принадлежащих $S_1$. Построение данного пути происходит следующим образом: если достигнут какой-то узел на этом пути (т. е. $z$ локализована в соответствующем ему треугольнике), то проверяется принадлежность точки $z$ всем потомкам этого узла; поскольку $z$ принадлежит ровно одному из потомков, то в путь добавляется еще одна дуга. Этот поиск можно также рассматривать как последовательную локализацию $z$ в триангуляциях $S_{h(N)}$, $S_{h(N)-1}$, ..., $S_1$. То, что $S_{i-1}$ является результатом детализации $S_i$, объясняет наименование самого метода.



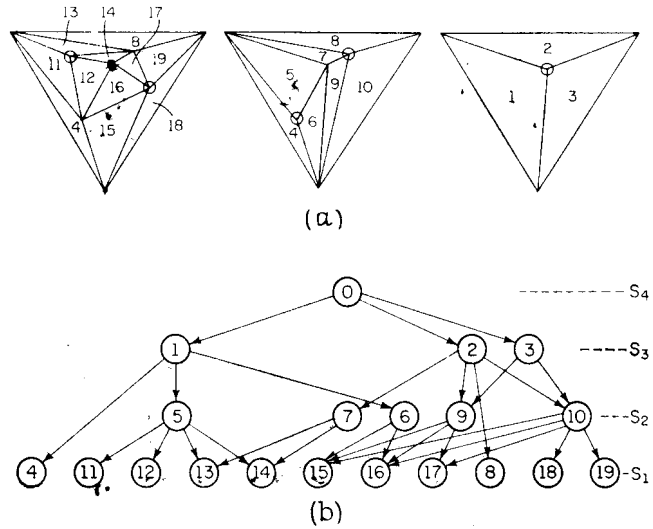



Рис. 2.19. Последовательность триангуляций (a) и соответствующий ей направленный ациклический граф поиска (b).

Более строго, предположим, что все потомки узла $v$ из $T$ собраны в список $\Gamma(v)$, и пусть ТРЕУГОЛЬНИК($v$) обозначает треугольник, отнесенный к узлу $v$. Теперь мы получили следующий поисковый алгоритм:

```
        procedure ЛОКАЛИЗАЦИЯ-ТОЧКИ
begin if(z ∉ ТРЕУГОЛЬНИК (корень)) then печать
        "z лежит в бесконечной области"
      else begin v := корень;
                 while (Γ(v) ≠ ∅) do
                       for каждый u ∈ Γ(v) do if
                       (z ∈ ТРЕУГОЛЬНИК(u)) then v := u;
                 печать v
           end
end.
```

Как упоминалось ранее, от выбора множества вершин триангуляции, которые будут удалены при построении $S_i$ по $S_{i-1}$,

существенно зависит эффективность метода. Предположим, что можно выбрать это множество так, чтобы выполнялись следующие свойства (здесь через $N_i$ обозначено число вершин в $S_i$):

**Свойство 1.** $N_i = \alpha_i N_{i-1}$, *где* $\alpha_i \leqslant \alpha < 1$ *для* $i = 2, \ldots, h(N)$.

**Свойство 2.** *Каждый треугольник* $R_j \in S_i$ *пересекается не более чем с* $H$ *треугольниками из* $S_{i-1}$, *и наоборот.*

Свойство 1 немедленно влечет за собой следствие, что $h(N) \leqslant \lceil \log_{1/\alpha} N \rceil = O(\log N)$, поскольку при переходе от $S_{i-1}$ к $S_i$ удаляется по меньшей мере фиксированная доля вершин. Из свойств 1 и 2, вместе взятых, следует, что память для $T$ равна $O(N)$. Действительно, заметим, что эта память используется для хранения узлов и указателей на их потомков. Из теоремы Эйлера о плоских графах следует, что $S_i$ содержит $F_i < 2N_i$ треугольников. Число узлов $T$, представляющих треугольники из $S_i$, не превосходит $F_i$ (только те треугольники, которые действительно «принадлежат» $S_i$, появляются на соответствующем ярусе $T$). Отсюда следует, что общее число узлов в $T$ меньше, чем

$$2(N_1 + N_2 + \ldots + N_{h(N)}) \leqslant 2N_1(1 + \alpha + \alpha^2 + \ldots$$
$$\ldots + \alpha^{h(N)-1}) < \frac{2N}{1-\alpha}$$

(заметим, что $N_1 = N$ по определению). Что касается памяти, используемой под указатели, то по свойству 2 каждый узел имеет не более $H$ указателей; поэтому не более $2NH/(1-\alpha)$ указателей появится в $T$. Это доказывает последнее утверждение.

Теперь надо показать, что существует такой критерий выбора множества удаляемых вершин, при котором выполняются свойства 1 и 2. Вот этот критерий: «Удалить несмежные вершины со степенью меньше $K$» (здесь $K$ — целое число, которое будет тщательно подобрано). Порядок просмотра и, если необходимо, удаления этих вершин произволен: начинаем с любой из них, помечаем ее соседей (они не могут удаляться) и продолжаем, пока еще остаются непомеченные вершины.

Введение данного критерия — это удачный прием. Выполнение свойства 2 обеспечивается тривиально. Поскольку удаление вершины со степенью меньше $K$ приводит к образованию многоугольника с числом ребер менее $K$, то каждый из удаленных тругольников пересекает не более $K - 2 \triangleq H$ новых треугольников. Для проверки выполнения свойства 1 необходимо воспользоваться некоторыми особенностями плоских графов. Хотя есть возможность провести более детальный анализ, приводящий к лучшим результатам, но следующие соображения достаточны для доказательства необходимого нам утверждения.



(1) Из формулы Эйлера для плоских графов, в частном случае триангуляции, ограниченной тремя ребрами, следует, что число вершин $N$ и число ребер $e$ связаны соотношением

$$e = 3N - 6.$$

(2) Пока в триангуляции есть внутренние вершины (в противном случае задача тривиальна), степень каждой из трех граничных вершин не меньше трех. Поскольку существует $3N - 6$ ребер, а каждое ребро инцидентно двум вершинам, то сумма степеней всех вершин меньше $6N$. Отсюда сразу следует, что не менее $N/2$ вершин имеет степень меньше 12. Следовательно, пусть $K = 12$. Пусть $\nu$ — это число выбранных вершин: поскольку каждой из них инцидентно не более $K - 1 = 11$ ребер, а три граничные вершины не выбираются, то мы имеем

$$\nu \geqslant \left\lfloor \tfrac{1}{12}\left(\tfrac{N}{2} - 3\right) \right\rfloor.$$

Следовательно, $\alpha \cong 1 - 1/24 < 0{,}959 < 1$, что доказывает справедливость свойства 1. Безусловно, наш грубый анализ дает оценку $\alpha$, непрактично близкую к 1, и оценка данного метода, полученная в результате этого анализа, вероятно, хуже, чем оценка, которую покажут численные эксперименты.

**Теорема 2.7.** *Локализацию точки в* $N$*-вершинном планарном подразбиении можно осуществить за время* $O(\log N)$ *с использованием* $O(N)$ *памяти, если* $O(N \log N)$ *времени ушло на предобработку.*

Другое доказательство этой теоремы, основанное на уточнении метода цепей, кратко описано в разд. 2.4.

### 2.2.2.4. Метод трапеций

Хотя вопрос о существовании оптимального метода локализации точки теоретически уже решен, а в литературе уже представлены и практически оптимальные алгоритмы, описанные в предыдущем разделе, но более привлекательным методом может оказаться такой, асимптотическое поведение которого в худшем случае, возможно, и неоптимально, о чем говорилось в разд. 1.2.1. Другими словами, желаемой целью может оказаться компромиссный подход, основанный на прямой процедуре поиска при, возможно, немного неоптимальной затрате памяти. Сейчас будет описан метод трапеций, который, по-видимому, удовлетворяет этим требованиям [Preparata (1981); Bilardi, Preparata (1981)] [1]).

Существуют несколько путей неформального изложения идеи

[1]) Экспериментальные наблюдения, опубликованные в [Edahiro et al. (1983)], подтвердили эти ожидания.

этого метода. Возможно, наиболее естественный путь — считать его развитием метода полос, принадлежащего Добкину и Липтону и описанного в разд. 2.2.2.1. Как указано ранее, этот метод имеет чрезвычайно простую процедуру поиска — $O(\log N)$, но использует досадно много памяти — $O(N^2)$ в худшем случае. Последнее обстоятельство имеет место благодаря возможному наличию «длинных» ребер в ППЛГ; длинным считается ребро, пересекающее несколько полос и, следовательно, разбиваемое на такое же число фрагментов. Интересно, что длинные ребра — проклятие метода полос — могут стать преимуществом в методе трапеций. В самом деле, длинное ребро $e$ так удобно разрезает ППЛГ, что горизонтальные полосы по каждую сторону от $e$ образуют полностью независимые структуры (в то время как в методе Добкина — Липтона $e$ находится внутри полосы). В результате плоскость будет разбита на трапеции, как определено ниже.

**Определение 2.5.** *Трапеция* имеет две горизонтальные стороны и может иметь две, одну или нуль боковых сторон, причем боковые стороны, если они есть, являются ребрами ППЛГ (или их частями) и никакое другое ребро ППЛГ не пересекает обе ее горизонтальные стороны.

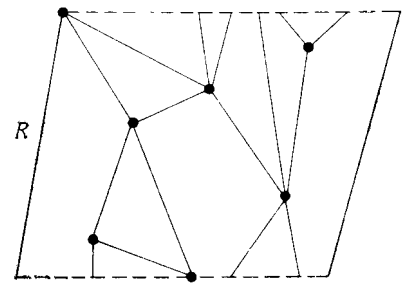
Поиск осуществляется путем локализации пробной точки в последовательности вложенных трапеций, до тех пор пока эта последовательность не закончится такой трапецией, внутри которой нет ребер ППЛГ или их фрагментов. Будет показано, что в этом методе ребро разбивается не более чем на $2\log_2 N$ фрагментов в худшем случае, что существенно улучшает оценку памяти, сохраняя $O(\log N)$ — оценку времени поиска метода Добкина — Липтона (с незначительным ухудшением константы).

В методе трапеций не требуется, чтобы граф был триангулирован; на самом деле позднее мы увидим, что не требуется даже прямолинейности его ребер. Однако на некоторое время допустим, что имеется ППЛГ $G$. Множество вершин $V$ графа $G$ упорядочено по возрастанию их ординат, а множество его ребер $E$ упорядочено в соответствии со следующим отношением (частичного порядка) «$\prec$»:

**Определение 2.6.** Даны два ребра $e_1$ и $e_2$ из $E$; запись $e_1 \prec e_2$ (читается «$e_1$ левее $e_2$») обозначает, что существует горизонталь, пересекающая оба ребра, и точка ее пересечения с $e_1$ будет левее соответствующей точки на $e_2$. (Позднее (в примечании 1, с. 87) мы обсудим, как алгоритмически получить это упорядочение.)

Основной механизм, строящий структуру данных поиска, обрабатывает по одной трапеции и стремится разбить ее на максимально возможное число более мелких трапеций Это делается путем разрезания трапеции $R$ на «нижний» и «верхний» слои $R_1$ и $R_2$ горизонталью, проходящей через ту вершину ППЛГ, чья ордината является медианой множества ординат вершин внутри $R$. Каждый из этих слоев $R_1$ и $R_2$, хотя и является геометрической трапецией, удовлетворяет не всем условиям определения 2.5, поскольку некоторые ребра ППЛГ могут пересекать одновременно обе его горизонтальные стороны; такие ребра назовем *накрывающими*. Теперь каждое ребро, накрывающее один из слоев [1]), определяет его последующий разрез.





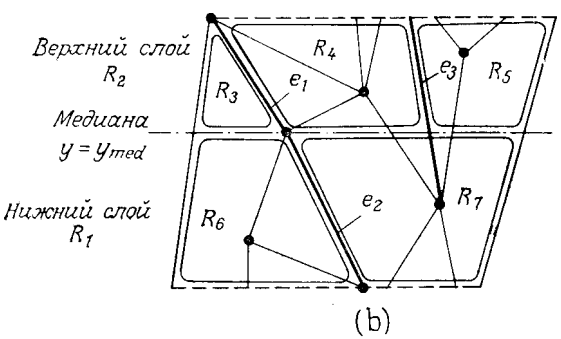




Рис. 2.20. Трапеция $R$ (a), разбитая на трапеции $R_3$, $R_4$, $R_5$, $R_6$ и $R_7$, как показано на (b).

Проделываются следующие операции. После определения медианы $y = y_{\text{med}}$ трапеции $R$ (рис. 2.20) цепочка ребер ППЛГ, пересекающих $R$, просматривается слева направо и разделяется на две цепочки, относящиеся к слоям $R_1$ и $R_2$ соответственно.

[1]) Никакое ребро не может накрывать и $R_1$, и $R_2$, иначе оно накрывало бы трапецию $R$, что противоречит определению.

Всякий раз, когда встречается накрывающее ребро, оно становится правой боковой границей новой трапеции, которая может обрабатываться независимо. Если внутренность трапеции пуста (ребра ППЛГ не пересекают ее), то ее обработка тривиальна.

Структура данных поиска, соответствующая трапеции с рис. 2.20, изображена на рис. 2.21. Каждой трапеции $R$ соответствует дерево двоичного поиска $T(R)$, каждый узел которого

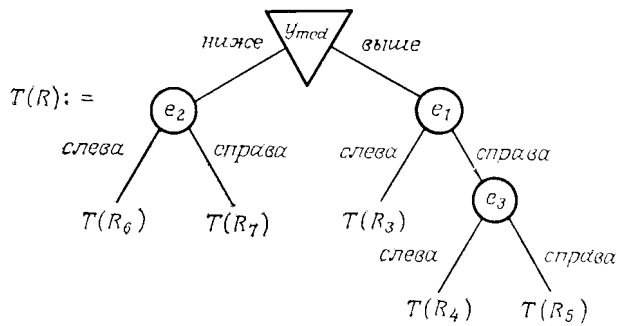



Рис. 2.21. Структура данных поиска, соответствующая трапеции с рис. 2.20.

связан с линейной проверкой. Удобно различать узлы двух типов: $\nabla$-узлы, если соответствующая проверка проводится относительно горизонтали, и O-узлы, если эта проверка ведется относительно прямой, несущей ребро ППЛГ. Очевидно, что корень $T(R)$ всегда будет $\nabla$-узлом. Заметим, что любому $\nabla$-узлу соответствует единственная вершина ППЛГ, а именно такая вершина, ордината которой является медианой множества ординат вершин ППЛГ в данной трапеции. Поскольку две крайние вершины ППЛГ не участвуют в разбиении трапеции, то число $\nabla$-узлов в дереве поиска будет равно $N-2$.

Более формально, процедура-функция ТРАПЕЦИЯ, строящая дерево поиска, получает на входе: $E$ — цепочку ребер ППЛГ, $V$ — последовательность вершин внутри $R$, упорядоченную по возрастанию их ординат, и $y$ — интервал $I$ трапеции $R$. В процедуре используются вспомогательные операции, такие как «поиск медианы» и «балансировка», которые будут описаны позже; $E_1$, $E_2$, $U_1$ и $U_2$ — рабочие списки этого алгоритма.

```
1  function ТРАПЕЦИЯ(E,V,I)
2  begin if(V = ∅) then return Λ (* лист дерева поиска *)
3        else begin E1 := E2 := V1 := V2 := U := ∅;
4                   y_med := медиана множества ординат V;
5                   I1 := [min(I), y_med]; I2 := [y_med, max(I)];
6                   repeat e ⇐ E;
7                          for i = 1,2 do
8                                 begin if(e имеет конец p
                                          внутри R_i)
                                        then
9                                       begin E_i ⇐ e;
10                                            V_i := V_i ∪ {p}
                                        end;
11                                    if(e накрывает R_i)
                                        или (e = Λ) then
12                                       begin U_i ⇐ ТРАПЕ-
                                                ЦИЯ(E_i, V_i, I_i);
13                                           if(e ≠ Λ)
                                               then U_j ⇐ e;
14                                           E_i := V_i := ∅
                                       end
                                end
15                 until e = Λ;
16                 новый(w); (* создание нового ∇-узла w,
                   корня T(R) *)
17                 Y[w] := y_med; (* дискриминация узла w *)
18                 ЛДЕРЕВО[w] := БАЛАНС(U1);
19                 ПДЕРЕВО[w] := БАЛАНС(U2);
                   (* функция БАЛАНС принимает на входе
                   чередующуюся последовательность
                   из деревьев и ребер и организует их
                   в сбалансированное дерево *)
20                 return ДЕРЕВО[w]
       end
  end.
```



В этом алгоритме выделяются три основных действия:

1) определение медианы множества ординат вершин в $R$;

2) разбиение нижнего и верхнего слоев на трапеции и получение для каждого слоя $R_i$ ($i = 1, 2$) цепочки $U_i$, состоящей из ребер и деревьев (в нашем примере $U_2 = T(R_3)e_1T(R_4)e_3T(R_5)$, а $U_1 = T(R_6)e_2T(R_7)$). На это уходит бóльшая часть всей работы, и делается она в цикле *repeat* в строках 6—15;

3) балансировка двух цепочек $U_1$ и $U_2$ (строки 18 и 19).

Первое действие можно выполнить в принципе за время $O(|V|)$ с помощью классического алгоритма поиска медианы. Однако существует более простой и прямой путь. Вершины трапеции располагаются в массиве по возрастанию ординат. (Медиану этого массива можно найти за одно обращение.) Модификация процедуры ТРАПЕЦИЯ происходит следующим образом. Последовательность ребер просматривается *дважды*. При первом проходе каждая вершина просто помечается именем той трапеции, к которой она отнесена, непосредственно используя эти пометки, строится массив вершин для каждой порождаемой

трапеции. При втором проходе выполняется цикл *repeat* (за исключением строки 10).

Вспомогательная функция БАЛАНС организует из чередующейся цепочки, состоящей из деревьев и ребер, $U = T_1e_1T_2e_2 \ldots e_{h-1}T_h$ — сбалансированное дерево. Каждое дерево $T_j$ имеет вес $W(T_j) \geqslant 0$, равный числу вершин ППЛГ, содержащихся в соответствующей трапеции. Вес $U$ равен $W(U) = \sum_{j=1}^{r} W(T_j)$. Функция БАЛАНС работает следующим образом:

1. Если $W(U)=0$ (т. е. $U = e_1e_2 \ldots e_{h-1}$), то сделать из этих ребер сбалансированное дерево. Иначе:

2.1. Найти такое целое $r$, чтобы $\sum_{j=1}^{r-1} W(T_j) < W(U)/2$, а $\sum_{j=1}^{r} W(T_j) \geqslant W(U)/2$.

2.2. Построить дерево, подобное изображенному на рис. 2.22(a), где $U' = T_1e_1 \ldots T_{r-1}$, $U'' = T_{r+1}e_{r+1} \ldots T_h$, а $\tau' =$ БАЛАНС$(U')$, $\tau'' =$ БАЛАНС$(U'')$. (Заметим, что $W(U') < W(U)/2$ и $W(U'') \leqslant W(U)/2$.)

Очевидно, что процедура БАЛАНС требует $O(h \log h)$ времени. Действительно, разделение $U$ на $(U', T_r, U'')$ потребует $O(h)$ времени, а затем последуют два рекурсивных вызова этой же процедуры, применяемых к половинам исходной цепочки.

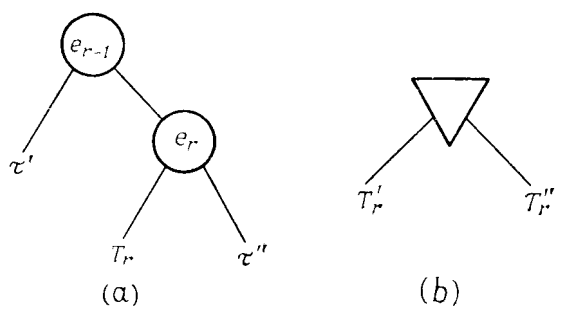



Рис. 2.22. Пример операции балансирования.

Но еще интереснее анализ высоты сбалансированных деревьев. Заметим для начала, что $h$ — число деревьев в цепочке $U$, меньшее $N$, где $N$, как и ранее, равно числу вершин в ППЛГ. В самом деле, из рис. 2.23 видно, что $h$ — число новых трапеций, на единицу большее числа ребер, накрывающих исходную трапецию. Последнее число ограничено сверху числом ребер, которые можно пересечь одной прямой линией. Теперь заметим, что эти ребра можно считать «диагоналями» простого многоугольника (ограниченного на рис. 2.23 штриховой линией). Если у этого многоугольника $s$ вершин, то в нем не более $s-3$ диагоналей (потому что эти диагонали определяют триангуляцию данного многоугольника). Поскольку $s \leqslant N$ (числа вершин ППЛГ), то получаем $h \leqslant s-2 \leqslant N-2$. Теперь можно доказать следующую теорему.



**Теорема.** *Для заданной цепочки* $U = T_1e_1 \ldots e_{h-1}T_h$ *высота* $\delta(U)$ *дерева, построенного с помощью процедуры БАЛАНС, не превосходит* $3\log_2 W(U) + \lceil \log_2 N \rceil + 3$.

*Доказательство.* Доказательство проведем индукцией по $W(U)$. Начнем с $W(U)=1$. В этом случае, процедура БАЛАНС

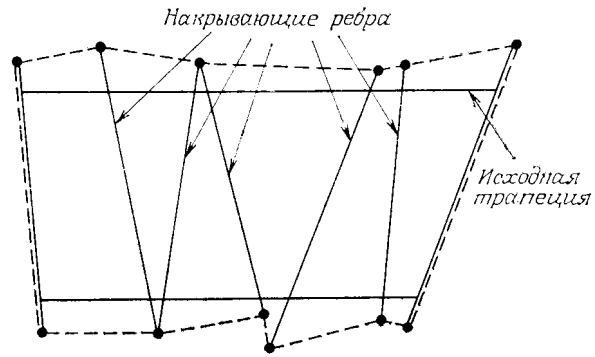



Рис. 2.23. Иллюстрация к доказательству того, что $h < N$ ($h$ — число новых трапеций).

дает такое дерево, как на рис. 2.22(a), у которого $\tau'$ и $\tau''$, возможно, пусты, а $T_r$ — дерево, приведенное на рис. 2.22(b). Здесь $T_r'$ и $T_r''$ содержат не более чем по $N$ штук O-узлов. Поэтому высота этого дерева не превосходит $\lceil \log_2 N \rceil + 3$, и теорема верна. Полагая $W(U) = K$ (целому положительному числу), допустим, что теорема справедлива для всех весов, меньших $K$. Будем различать два случая:

*Случай 1.* $W(T_r) \leqslant K/2$. Поскольку верно также, что $W(U')$, $W(U'') \leqslant K/2$, то по предположению индукции $\delta(U')$, $\delta(U'')$, $\delta(T_r) \leqslant 3\log_2 K/2 + \lceil \log_2 h \rceil + 3 \leqslant 3\log_2 K - 3 + \lceil \log_2 N \rceil + 3$; так что конфигурация на рис. 2.22(a) обеспечивает индуктивный переход.

*Случай 2.* $W(T_r) > K/2$. Поскольку $W(U')$, $W(U'') \leqslant K/2$, то доводы для $\tau'$ и $\tau''$ будут теми же, что и в случае 1. Относительно $T_r$ заметим, что оно соответствует трапеции. Поэтому под действием процедуры ТРАПЕЦИЯ $T_r$ получает структуру, показанную на рис. 2.22(b), где $T_r'$ и $T_r''$ сами являются сбалансированными деревьями, и в силу медианного разбиения $W(T_r')$,

$W(T_r'') \leqslant W(T_r)/2 \leqslant K/2$. Следовательно, для некоторого $h'' < N$ имеем $\delta(T_r')$, $\delta(T_r'') \leqslant 3 \log W(T_r)/2 + \lceil \log_2 h'' \rceil + 3 \leqslant 3 \log K - 3 + \lceil \log_2 N \rceil + 3$, что и завершает доказательство.

Поскольку для ППЛГ верно, что $W(U) \leqslant N$, то высота дерева поиска для ППЛГ не превосходит $4\lceil \log_2 N \rceil + 3$.

Наконец, рассмотрим цикл *repeat* в процедуре ТРАПЕЦИЯ. Для каждого просматриваемого ребра (т. е. взятого из списка $E$) эта процедура осуществляет константный объем вычислений в строке 6 и в строках либо 8—10, либо 12—14. Следовательно,

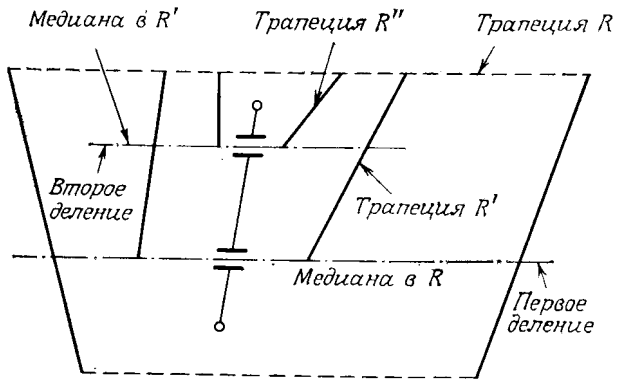



Рис. 2.24. Иллюстрация механизма деления ребра.

общая работа, проделанная в этом цикле, пропорциональна суммарному числу фрагментов ребер, порождаемых данным алгоритмом. Теперь оценим это число сверху. Обращаясь к рис. 2.24, видим, что ребро делится впервые тогда, когда его концы лежат в разных слоях, определяемых медианой трапеции $R$. После этого деления верхний фрагмент ребра попадает в трапецию $R'$. Если медиана $R'$ делит $e$ (худший случай), то $e$ накрывает нижний слой в $R'$, а следующее его деление может произойти только в верхнем слое $R'$. Аналогичные соображения применимы к фрагменту $e$, попавшему в нижний слой $R$. Поэтому, выражая число делений верхнего фрагмента ребра $e$ как функцию $c(s')$ от числа $s'$ вершин в $R'$, мы получаем грубую оценку: $c(s') \leqslant 1 + c(s'')$, где $s''$ — число вершин в трапеции $R''$, как показано на рис. 2.24. Поскольку $s'' < s'/2$, мы получаем $c(s') \leqslant \log_2 s'$, а поскольку $s' < N/2$, то общее число делений $e$ не превосходит $2 \log_2 N - 2$. Эта оценка доказывает два утверждения. Первое: общая работа, производимая в цикле *repeat*, пропорциональна $O(N \log N)$, поскольку имеется $O(N)$ ребер, а каждое ребро разделено на не более чем $O(\log N)$ фрагментов. Поскольку обе задачи (поиска медиан и балансировки) требуют в сумме $O(N \log N)$ времени, то все построение структуры данных поиска завершается за время $O(N \log N)$. Второе утверждение таково: поскольку структура данных поиска имеет O-узел для каждого фрагмента ребра и $\nabla$-узел для каждой вершины, то используемая память также оценивается через $O(N \log N)$ [1]). Подводя итог, имеем:



**Теорема 2.8.** *Точку можно локализовать в N-вершинном планарном подразбиении менее чем за* $4 \log_2 N$ *проверок, с использованием* $O(N \log N)$ *памяти и такого же времени предобработки.*

**Примечание 1.** Ранее мы отложили обсуждение вопроса о построении упорядочения множества ребер $E$, согласованного с отношением частичного порядка «$\prec$», задаваемого определением 2.6. Для достижения этой цели можно употребить метод,

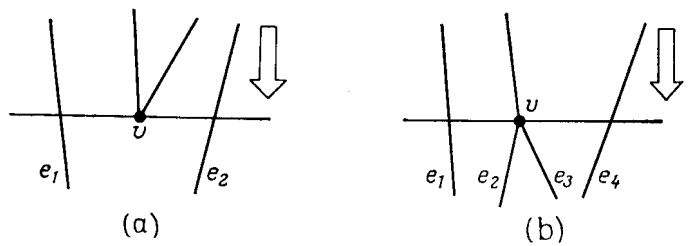



Рис. 2.25. Пример вычисления отношения «$\ll$» как результата обработки вершины $v$. В случае (a) порождается пара $(e_1, e_2)$; в случае (b) порождаются пары $(e_1, e_2)$, $(e_2, e_3)$ и $(e_3, e_4)$.

являющийся простой модификацией плокого заметания, описанного в разд. 2.2.2.2 для «регуляризации» плоского графа.

В процессе заметания сверху вниз вершины графа $G$ просматриваются, а статус заметающей прямой корректируется так, как описано выше. Побочным результатом этого заметания является регистрация смежности всех ребер, получаемая при их вставлении или удалении. Отношение смежности, выражаемое в виде упорядоченной пары ребер (левое, правое) (рис. 2.25), является транзитивным сведением «$\ll$» искомого частичного упорядочения. Очевидно, что если граф $G$ имеет $N$ вершин, то отношение «$\ll$» можно вычислить за время $O(N \log N)$. Если известно отношение «$\ll$», то искомое упорядочение можно

[1]) В худшем случае затраты памяти действительно составляют $O(N \log N)$. Однако для ППЛГ специального вида [Bilardi, Preparata (1981)] требуется памяти в среднем $O(N)$, что заставляет предполагать аналогичное поведение в среднем и для ППЛГ общего вида.

получить за время $O(N)$ стандартным методом топологической сортировки [Knuth (1973) [1])].

**Примечание 2.** Описанный метод не ограничивается лишь ППЛГ. В самом деле, прямолинейные отрезки можно заменить другими кривыми, если будут выполнены следующие два условия: (1) эти кривые — однозначные функции от одной избранной координаты (скажем, от $y$); (2) дискриминацию точки относительно любой из таких кривых можно проделать за константное время. Например, эти условия, очевидно, выполняются для дуг окружностей и других коник, которые не имеют горизонтальных касательных к неконцевым точкам. Применение метода Киркпатрика (разд. 2.2.2.3) в подобной ситуации представляет очевидные трудности, поскольку требуется перетриангулировать многоугольник с криволинейными ребрами.

## 2.3. Задачи регионального поиска

### 2.3.1. Общие замечания

Как было показано в разд. 2.1, задачи регионального поиска можно рассматривать как двойственные, в определенном смысле, к задачам локализации точки, которые уже были рассмотрены.

Здесь файлом будет считаться набор записей, каждая из которых идентифицирована упорядоченным $d$-плексом «ключей» $(x_1, x_2, \ldots, x_d)$. Естественно считать $d$-плекс ключей точкой в $d$-мерном декартовом пространстве. На таком файле можно допустить некоторые действия пользователя, или *запросы*. Однако в данном контексте мы будем иметь дело исключительно с *региональными запросами*. Запрос определяет область (регион) в $d$-мерном пространстве, а результатом поиска является отчет о подмножестве точек файла, содержащихся в этой запрашиваемой области. Другой, более ограниченной целью поиска является простой подсчет числа точек в запрашиваемой области. (Единый подход к этим двум типам задач получится, если каждую запись сопоставить с одним из элементов коммутативной полугруппы с операцией «*» и считать, что запрос реализует операцию «*» над всеми записями в своем интервале. Тогда в режиме отчета каждой записи сопоставлено ее имя, а «*» есть операция объединения подмножеств; в режиме подсчета каждой записи сопоставлено целое число 1, а «*» есть обычное сложение [Fredman (1981)].)

Представление в декартовом пространстве является абстракцией для множества очень важных приложений, часто именуемых «поиском по многим ключам» [Knuth (1973), v. 3]. Например, отдел кадров компании может пожелать узнать число работников в возрасте от 30 до 40 лет, зарабатывающих от 27 000 до 34 500 долларов в год. Помимо данного выдуманного примера приложения подобного рода возникают в географии, статистике [Loftsgaarden, Queensberry (1965)] и автоматизации проектирования [Lauther (1978)].

Возвращаясь к нашей абстрактной задаче, заметим, что запросы уникального типа не представляют интереса, ибо их обработка непременно сведется к какому-нибудь виду исчерпывающего перебора в пространстве. Поэтому сосредоточимся на массовых запросах, и тогда потребуются изначальные затраты на предобработку и запоминание, чтобы в долгосрочном плане получать выигрыш во времени поиска. В этом смысле нашей конечной целью является решение данной задачи в ее самом широком обобщении, т. е. умение работать в пространстве произвольной размерности и на множестве произвольных областей запроса как по форме, так и по размеру. В то время как вопрос о размерности пространства можно адекватно разрешить (за определенную вычислительную плату, разумеется), к сожалению, бóльшая часть того, что известно сегодня относительно природы запроса, связано с гиперпрямоугольными областями. (Гиперпрямоугольная область — это декартово произведение интервалов на различных координатных осях [1]). Хотя случай гиперпрямоугольной области и очень важен, он, несомненно, не охватывает всех желамых форм. Например, поиск всех точек на ограниченном расстоянии $d$ от пробной точки $z$, называемый *поиском на ограниченном расстоянии*, порождает сферическую область поиска (т. е. гиперсферу радиусом $d$ с центром в $z$). Как будет показано ниже, методы, успешно применяемые в гиперпрямоугольном случае, неприменимы в случае сферы. Если для сферы когда-нибудь и удастся достичь сравнимой эффективности, то получена она будет, вероятно, на основе совершенно иных принципов. На самом деле, имеются кое-какие недавние продвижения [Chazelle, Cole, Preparata, Yap (1984)] в задаче поиска на ограниченном расстоянии при $d = 2$ (задача *кругового регионального поиска*), основанные на понятии «близости»;

[1]) См. русское издание: Кнут, т. 1, с. 323. — *Прим. перев.*

[1]) Выбранная так область поиска часто называется *ортогональным запросом*, возможно, потому, что гиперплоскости, ограничивающие гиперпрямоугольник, ортогональны координатным осям. Иногда гиперпрямоугольник называют *прямоугольным* (rectilinear); употребляя слово, вероятно, подсказанное формой путей в плоской $L_1$-метрике (см. гл. 8). Однако нам кажется неудачным такое использование термина «прямоугольный» (дискуссия на эту тему будет продолжена в разд. 8.2). (Терминологические вопросы, которые здесь возникли, специфичны для английского языка и не имеют прямых аналогий в русском. — *Ред.*).

они будут представлены в гл. 5 и 6. Эти рассуждения, строго говоря, приложимы только к такому режиму работы, когда поисковые операции не получают доступа ни к каким другим элементам файла, кроме содержащихся внутри области запроса. Если такое ограничение снять, то сферическую область можно адекватно аппроксимировать семейством гиперпрямоугольных областей; хотя подобный подход может оказаться неудачным при оценке поведения в худшем случае, однако в «реальных», практических ситуациях он может работать весьма хорошо (например, метод $k$-$D$-дерева из разд. 2.3.2).

Общая черта всех методов, которые будут рассмотрены в настоящей главе, состоит в том, что используемые структуры данных будут *статическими*, т. е. они не будут модифицироваться после их построения. Очевидно, отсюда следует, что все составляющие их элементы — в нашем случае это множество точек в $d$-мерном пространстве — должны быть заданы заранее. Соответствующие *динамические* структуры данных, т. е. такие структуры, которые допускают вставку и удаление своих элементов, по-видимому, будут значительно более сложными; их изучение находится до сих пор в весьма активной стадии и не будет детально обсуждаться в данной книге, за исключением нескольких коротких замечаний и библиографических ссылок в разд. 2.4.

Теперь стоит поискать критерий для сравнения разных методов. Естественно в качестве критерия выбрать нижнюю оценку для некоторых мер эффективности этих методов. Однако, перед тем как двинуться дальше, стоит неформально проанализировать взаимосвязь разных способов оценки.

Как обычно, предположим, что файл является фиксированным набором $S$ из $N$ записей. Ответом на запрос будет $S'$-подмножество $S$. Один из возможных подходов состоит в предварительном вычислении ответов на все допустимые запросы. Поскольку допустимые ответы составляют конечное множество (как множество всех подмножеств конечного множества $S$), то бесконечное множество запросов разбивается на конечное число классов эквивалентности (два запроса эквивалентны, если на них поступает одинаковый ответ). Тогда обработка запроса свелась бы к его сравнению со стандартным представителем соответствующего класса эквивалентности, что в свою очередь немедленно дало бы ответ. Этот метод прямого доступа (он будет обсуждаться в разд. 2.3.3) продемонстрировал бы весьма простую обработку запроса, достигнутую за счет огромных затрат памяти и времени предобработки. Однако из двух последних ресурсов память важнее времени предобработки; в самом деле, время предобработки это вполне допустимая однократная затрата, но память является долгосрочной затратой, поскольку непригодна для других целей на протяжении всей жизни запросной системы. Поэтому стало традицией, если и не пренебрегать вовсе, то по крайней мере не переоценивать затраты на предобработку, а вместо этого концентрировать внимание на времени обработки запроса и объеме памяти для файла. (Впрочем, во многих приложениях время предобработки и память оцениваются функциями одного порядка, поэтому бывает достаточно оценить лишь последнюю.) Вышеописанный метод характеризуется малым временем запроса и большой памятью; понятно, что уменьшение требований к памяти достигается ценой увеличения времени запроса; итак, мы столкнулись с взаимозависимостью между двумя мерами эффективности. В соответствии с современной практикой будем характеризовать метод регионального поиска парой: (память файла, время запроса).



Понятие «время запроса», в свою очередь, заслуживает некоторого обсуждения. Работа, затрачиваемая на обработку запроса, естественно, зависит не только от размера файла $N$ и от подмножества $S' \subset S$, содержащегося внутри запрашиваемого региона, но и от типа запроса (т. е. от природы ранее упомянутой операции «$*$» на полугруппе). В самом деле, ответом на запрос в режиме подсчета всегда будет единственное целое число ($k$ — мощность $S'$), в то время как ответом на запрос в режиме отчета будут имена всех $k$ элементов $S'$. В принципе можно различать два вида действий при обработке запроса:

1. *Поиск*, т. е. действия, приводящие к элементам искомого подмножества (обычно последовательность сравнений).
2. *Выборка*, т. е. действия по составлению ответа на запрос (для запросов в режиме отчета — это фактическое извлечение искомого подмножества).

Анализ запросов в режиме подсчета удобен тем, что всю вычислительную работу вбирает в себя «поиск»; при этом следует ожидать, что время запроса для худшего случая будет функцией $f(N, d)$, зависящей от размера файла и числа измерений. Намного сложнее случай запросов в режиме отчета, поскольку вычислительная работа теперь является комбинацией поиска и выборки, и та ее часть, которая связана с выборкой, ограничена снизу числом $k$ — мощностью $S'$. Если потребовать, чтобы при обработке запроса обеспечивался доступ *только* к элементам $S'$, то ясно, что время на поиск этих элементов будет одинаковым для запросов обоих типов. Если же разрешить доступ к супермножеству, немного превосходящему искомое множество $S'$, то появится возможность сократить затраты памяти, а часть стоимости поисковой работы «отнести на счет» задачи выборки; в этом и состоит основная суть метода *фильтрующего поиска* [Chazelle (1983c)], который будет обсуждаться в конце данной

главы [1]). Итак, в общем случае верхняя оценка времени запроса в режиме отчета будет выражена в форме $O(f(N, d) + k \cdot g(N, d))$; эти два слагаемых следует понимать как соответственно «время поиска» и «время выборки», если допустимо в точности выбираемое множество, или как времена запроса для «малого» ($k = O(f(N, d))$) и «большого» ($f(N, d) = o(k)$) выбираемых множеств в более общем случае.

Относительно нижних оценок сразу же заметим, что $\Omega(k)$ — это тривиальная нижняя оценка времени выборки, поскольку длина результата пропорциональна $k$. Что же касается времени поиска, то обратимся к модели двоичного дерева решений, которая, как хорошо известно, подсчитывает число сравнений, необходимых для доступа к элементам множества $S' \subset S$ внутри запросного региона (заметим, что здесь мы считаем доступными только те элементы, которые необходимо выбрать). Это число сравнений ограничено снизу величиной $\log_2 Q(S)$, где $Q(S)$ — число различных подмножеств $S$, получаемых как ответы на запросы. Обращаясь к рис. 2.26, рассмотрим следующее множество [Bentley, Maurer (1980)]: $S$ — множество всех точек, имеющих одну и только одну ненулевую целочисленную координату в интервале $[-a, a]$; ясно, что $N = 2ad$. Затем выберем независимо по каждой оси координат два любых целых числа в интервалах соответственно $[-a, -1]$ и $[1, a]$; они и определят регион запроса, для которого подмножество связанных с ним точек непусто и отлично от всех других подмножеств. Поскольку вышеупомянутый выбор можно осуществить $a^{2d} = (N/(2d))^{2d}$ способами, то нижняя оценка числа двоичных решений равна $\Omega(\log(N/(2d))^{2d}) = \Omega(d \log N)$. Хотя более тонкий комбинаторный анализ [Saxe (1979)] может дать более точную оценку числа ортогональных регионов, обладающих различными выбираемыми множествами точек, однако и простые доводы, приведенные выше, дали нам нижнюю оценку времени поиска $\Omega(d \log N)$. К сожалению, как отметил Люкер [Lueker (1978)], модель дерева решений, по-видимому, не адекватна для данной задачи, поскольку для больших $d$ все известные методы показывают экспоненциальную, а не линейную зависимость времени поиска от $d$ (т. е. верхняя граница имеет вид $O((\log N)^d)$). Этот вопрос был поставлен Фредменом [Fredman (1981)], который доказал нижнюю оценку времени поиска $\Omega((\log N)^d)$ для *динамических* запросных систем (оценивая сложность операции «*» на полугруппе, которая была введена ранее в настоящем разделе); однако не совсем ясно, как результат Фредмена можно приложить к рассматриваемой здесь ситуации. Наконец, заметим, что $\Omega(Nd)$ — тривиальная нижняя оценка памяти, занятой под структуру данных поиска.

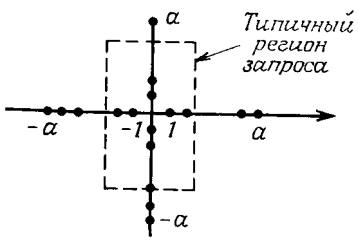



Рис. 2.26. Пример множества $S$ при $d = 2$.

[1]) Существует принципиальное отличие фильтрующего поиска от того метода поиска, в котором заданный регион аппроксимируется регионами других типов (например, сферический регион — гиперпрямоугольными). Хотя оба эти метода не относятся к типу «точного доступа», при фильтрующем поиске размер доступного множества не более чем на мультипликативную *константу* отличается от размера выбираемого множества.



Остаток этой главы будет посвящен в основном алгоритмам для задачи типа «отчета», а не типа «подсчета» (за иключением упражнения 4). Заметим, что алгоритм первого типа можно тривиально использовать для решения задачи второго типа; однако обратное преобразование в общем случае невозможно, т. е. могут быть развиты специфические эффективные методы для решения задачи типа «подсчета». Для всех описываемых алгоритмов время поиска выражается в форме $O(f(N, d) + k)$.

В следующих разделах показано несколько достаточно интересных и тонких методов, предложенных для задачи регионального поиска. Будет пролит свет на те трудности, которые до сих пор не позволяют построить оптимальные алгоритмы и которые непосредственно определяют соотношение между двумя важными ресурсами — временем запроса и объемом памяти.

Чтобы соответствующим образом подготовиться к изложению существующих методов, нам придется начать с простейшего примера регионального поиска, который при совершенной своей тривиальности содержит существенные черты данной задачи: речь пойдет об одномерном региональном поиске.

Множество из $N$ точек на оси $x$ представляет файл, а запросным регионом является отрезок $[x', x'']$ (называемый $x$-регионом). Методом, дающим эффективный (оптимальный) региональный поиск, является *двоичный поиск*, т. е. дихотомия искомого упорядоченного множества. В самом деле, ясно, что $x'$ — левый конец $x$-региона — локализуется на оси $x$ дихотомией; на этом завершается поиск, а для осуществления выборки надо пройти ось $x$ в направлении вырастания $x$ вплоть до достижения правого конца $x''$. Структурой данных, обеспечивающей указанное действие, является *прошитое двоичное дерево*, т. е. такое сбалансированное двоичное дерево, листья которого дополнительно связаны в списке, отражающем порядок абсцисс; дерево и список обрабатываются на фазах соответственно поиска и выборки. Заметим, что описанный метод оптимален как по времени запроса $\theta(\log N + k)$, так и по памяти $\theta(N)$.

В последующем обсуждении различных подходов к $d$-мерному региональному поиску прежде всего будем рассматривать простейший случай (т. е. $d = 2$) для выявления таких существенных черт каждого метода, которые свободны от бремени размерности. После достижения указанной цели мы будем стремиться там, где это уместно, к обобщению на произвольное число измерений.

### 2.3.2. Метод многомерного двоичного дерева ($k$-$D$-дерева)

Мы только что видели чрезвычайную важность дихотомии при разработке оптимального метода для одномерного регионального поиска. Поэтому естественно попытаться обобщить дихотомию на двумерный случай, где файл является набором из $N$ точек, лежащих на плоскости $(x, y)$. То, что делается при дихотомии, есть не что иное, как последовательное разрезание региона (неважно, конечного или бесконечного) на две части. В случае двух измерений всю плоскость можно считать бесконечным прямоугольником, который будет разрезан сначала на две полуплоскости прямой, параллельной одной из осей, скажем оси $y$. Затем каждая из этих полуплоскостей может разрезаться еще раз прямой, параллельной оси $x$, и т. д., меняя на каждом шаге направление разрезающей линии, например от $x$ к $y$. Как выбирать разрезающие линии? В точном соответствии с тем же принципом, который используется при обычной дихотомии, т. е. руководствуясь принципом получения приблизительно равного числа элементов (точек) по каждую сторону от разреза. Таким образом мы пришли к правомерному обобщению дихотомии и, более того, к основной идее метода *многомерного двоичного дерева* [Bentley (1975)].

Более строго, назовем (обобщенным) *прямоугольником* такую область на плоскости, которая определена декартовым произведением $[x_1, x_2] \times [y_1, y_2]$ $x$-интервала $[x_1, x_2]$ и $y$-интервала $[y_1, y_2]$, включая предельные случаи, когда в любой комбинации допускается: $x_1 = -\infty$, $x_2 = \infty$, $y_1 = -\infty$, $y_2 = \infty$. Поэтому мы будем считать прямоугольниками также неограниченные (с одной или двух сторон) полосы, любой квадрант, или даже всю плоскость.

Процесс разбиения $S$ путем разрезания плоскости лучше всего иллюстрировать в сочетании с построением двумерного двоичного дерева $T$. С каждым $v$ узлом $T$ мы неявно связываем прямоугольник $\mathcal{R}(v)$ (он определен выше) и подмножество $S(v) \subseteq S$ точек, лежащих внутри $\mathcal{R}(v)$; явно же, т. е. как фактические параметры этой структуры данных, свяжем с $v$ одну избранную точку $P(v)$ из $S(v)$ и «секущую прямую» $l(v)$, проходящую через $P(v)$ и параллельную одной из координатных осей.



Этот процесс начинается с определения корня $T$. С $\mathcal{R}$(корень) соотносится вся плоскость, и полагается, что $S$(корень) $= S$; затем определяется точка $p \in S$, такая что $x(p)$ — медиана множества абсцисс точек из $S$(корень), и полагается, что $P$(корень) $= p$, а с $l$(корень) соотносится прямая с уравнением $x = x(p)$. Точка $p$ разбивает $S$ на два множества приблизительно равной мощности, назначенных потомкам корня. Процесс

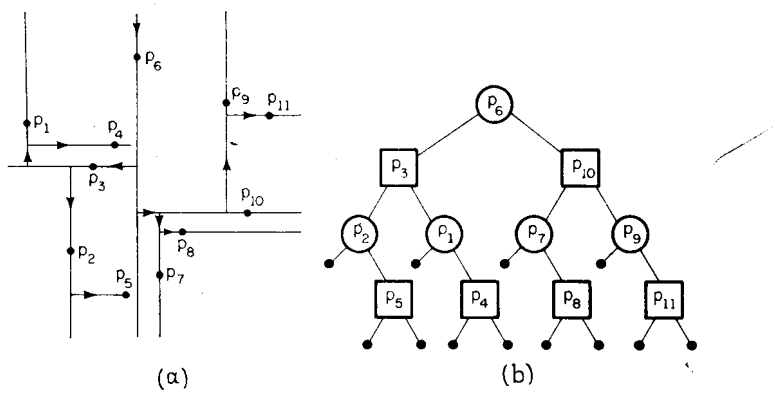



Рис. 2.27. Иллюстрация метода поиска с помощью двумерного двоичного дерева. Разбиение плоскости (а) моделируется деревом (b). Поиск начинается с проведения вертикали через $P_6$. На рис. (b) приняты следующие графические обозначения: круглые узлы соответствуют вертикальным разрезам, квадратные — горизонтальным, точки — листьям дерева.

дробления прекращается, когда обнаружен прямоугольник, не содержащий внутри точек; соответствующий ему узел является листом дерева $T$.

Данный метод проиллюстрирован на примере с рис. 2.27 для множества из $N = 11$ точек. Узлы трех разных типов обозначены различными графическими символами: кружками — нелистовые узлы с вертикальной линией разреза, квадратами — нелистовые узлы с горизонтальной линией разреза, точками — листья. Такая структура данных часто называется *2-D-деревом* (аббревиатура выражения «двумерное двоичное дерево поиска»).

Изучим теперь использование структуры данных типа 2-$D$-дерева для регионального поиска. Алгоритмической схемой будет метод «разделяй и властвуй» в чистом виде. Действительно, рассмотрим взаимное расположение прямоугольной области $\mathcal{R}(v)$, связанной с $v$ узлом $T$, и некоего прямоугольного региона $D$ такого, что $\mathcal{R}(v)$ и $D$ имеют непустое пересечение. Область $\mathcal{R}(v)$ разрезана на два прямоугольника $R_1$ и $R_2$ прямой $l(v)$, проходящей через $P(v)$. Если $D \cap \mathcal{R}(v)$ полностью содержится

в $R_i$ ($i = 1, 2$), то поиск продолжается с единственной парой типа (область, регион), а именно с $(R_i, D)$. Если же область $D \cap \mathscr{R}(v)$ разрезана прямой $l(v)$, то $l(v)$ имеет непустое пересечение с $D$, а значит, $D$ может содержать $P(v)$. Поэтому сначала проверим, находится ли $P(v)$ внутри $D$, и если да, то поместим эту точку в выбираемое множество; затем продолжим поиск с двумя парами типа (область, регион), а именно с $(R_1, D)$ и $(R_2, D)$. Этот процесс поиска прекращается при достижении любого листа.

Более строго любой узел $v$ дерева $T$ характеризуется тройкой параметров $(P(v), t(v), M(v))$. Точка $P(v)$ уже была определена. Два других параметра вместе определяют прямую $l(v)$, а именно $t(v)$ определяет горизонтальность или вертикальность $l(v)$; в первом случае $l(v)$ — это прямая $y = M(v)$ [1]), во втором случае — это прямая $x = M(v)$ [2]). Данный алгоритм накапливает выбираемые точки в списке $U$ — внешнем по отношению к процедуре и первоначально пустом. Обозначая через $D = [x_1, x_2] \times [y_1, y_2]$ запросный регион, получаем, что поиск на дереве $T$ осуществляется вызовом процедуры ПОИСК(корень($T$),$D$), имеющей следующее описание:

```
procedure ПОИСК(v, D);
begin if(t(v)=вертикаль) then [l,r] := [x1,x2] else
                               [l,r] := [y1,y2];
      if(l ≤ M(v) ≤ r) then if(P(v) ∈ D) then U ⇐ P(v);
      if(v ≠ лист) then
         begin if(l < M(v)) then ПОИСК(ЛСЫН[v],D);
               if(M(v) < r) then ПОИСК(ПСЫН[v],D)
         end
end.
```

На рис. 2.28(a) показан пример регионального поиска для файла, приведенного на рис. 2.27(a). В частности, на рис. 2.28(b) показан обход узлов $T$, осуществленный описанной выше процедурой поиска. Заметим, что только в тех узлах, где поисковый обход раздваивается (таких, как $p_6$, $p_3$, $p_2$, $p_4$, $p_{10}$, $p_7$, $p_8$), производится проверка принадлежности точки региону. Звездочкой (*) помечены те узлы, в которых такая проверка успешна, и соответствующая точка выбрана. Фактически выбранное множество — это $\{p_3, p_4, p_8\}$.

С точки зрения оценки сложности 2-$D$-дерево использует оптимальную память $\theta(N)$ (по узлу на точку из $S$). Кроме того, его можно построить за оптимальное время $\theta(N \log N)$ следующим образом. Вертикальный разрез множества $S$ производится в результате вычисления медианы множества $x$-координат точек из $S$ за время $O(|S|)$ (с использованием алгоритма из работы [Blum et al. (1973)]) и путем формирования разбиения $S$ с такой же оценкой времени; аналогично и для горизонтального разреза. Итак, за время $O(N)$ исходное множество разбивается, в результате чего получаются полуплоскости, в каждой из которых по $N/2$ точек; тривиально получаем рекуррентное соотношение для времени $T(N)$ работы алгоритма построения дерева:

$$T(N) \leqslant 2T(N/2) + O(N),$$

что и дает обещанную оценку этого времени. В более прямой реализации этого построения обходятся без сложного алгоритма

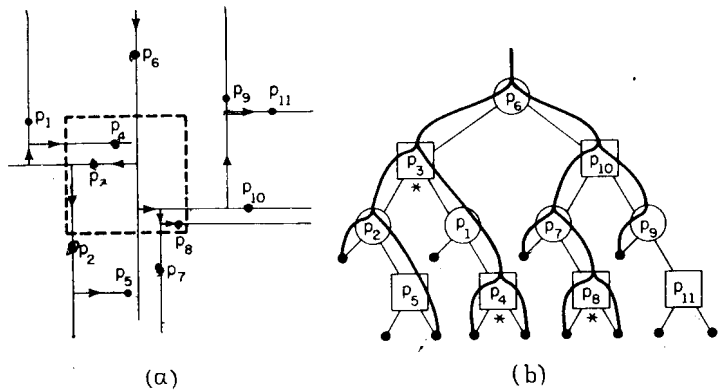



Рис. 2.28. Пример регионального поиска на ранее заданном файле. Часть (b) показывает узлы, фактически пройденные при поиске.

поиска медианы. Поскольку в только что описанном методе фактически производится сортировка множеств $x$- и $y$-координат путем рекурсивного поиска медианы, то мы можем прибегнуть к предварительной сортировке для формирования упорядоченных массивов абсцисс и ординат точек из $S$, именуемых соответственно $x$- и $y$-массивами. Это займет $O(N \log N)$ времени. Первый разрез производится выборкой (за константное время) медианы $x$-массива путем маркировки тех точек, чьи абсциссы, скажем, меньше этой медианы (за время $O(N)$). Маркировка позволяет быстро сформировать два отсортированных подмассива, и процесс для них рекуррентно повторяется.

Анализ времени запроса для худшего случая намного более сложен. И не удивительно, что этот анализ был разработан Ли и Вонгом [Lee, Wong (1977)] значительно позже того, как были впервые предложены многомерные двоичные деревья. Очевидно, что время запроса пропорционально *общему числу* узлов в $T$,

[1]) Здесь $M(v) = y(P(v))$. — *Прим. перев.*
[2]) Здесь $M(v) = x(P(v))$. — *Прим. перев.*

посещаемых поисковым алгоритмом, поскольку в каждом узле этот алгоритм затрачивает константное время. Очевидно, что это время потрачено в узле $v$ не зря, если точка $P(v)$ выбирается (*продуктивный* узел); иначе, этот узел считается *непродуктивным*. Анализ, проделанный Ли и Вонгом, направлен на построение ситуации худшего случая, т. е. он соответствует максимальному поддереву посещенных узлов, из которых все непродуктивны. Как указано ранее, каждый узел $v$ из $T$ соответствует

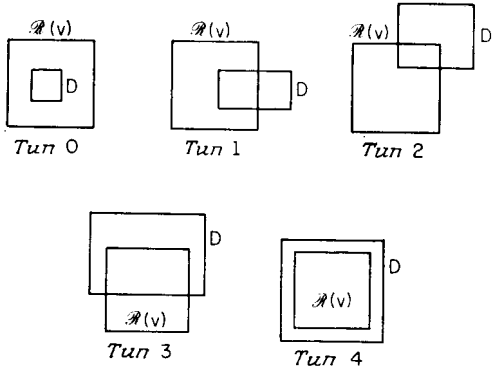



Рис. 2.29. Примеры различных типов пересечений $D$ с $\mathscr{R}(v)$, когда $D \cap \mathscr{R}(v) \neq \varnothing$.

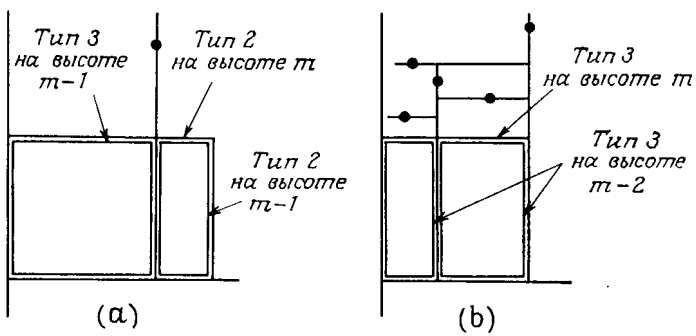



Рис. 2.30. Самовоспроизводящиеся ситуации, соответствующие непродуктивным узлам в $T$.

некоему обобщенному прямоугольнику $\mathscr{R}(v)$. Пересечения запросного региона $D$ и подобного обобщенного прямоугольника $\mathscr{R}(v)$ могут быть отнесены к разным «типам» в зависимости от числа сторон $\mathscr{R}(v)$, имеющих непустые пересечения с $D$. В частности, если обозначить это число через $i$, то говорят, что пересечение имеет тип $i$ для $i = 0, 1, 2, 3, 4$ (рис. 2.29). Единственный тип пересечений, который всегда продуктивен — это тип 4; все остальные могут оказаться непродуктивными. Ограничимся частными случаями пересечений типа 2 и 3. (Заметим, что пересечения типов 2 и 3 могут начать появляться только на узлах, удаленных от корня на два и три уровня соответственно.) Рассматривая рис. 2.30(a), можно легко построить ситуацию, когда пересечение типа 2 на высоте $m$ непродуктивно и порождает одно пересечение типа 2 и одно пересечение типа 3 на высоте $(m-1)$ (оба они могут быть сделаны непродуктивными). Аналогично, как показано на рис. 2.30(b), при том же самом ограничении на типы непродуктивных узлов можно построить ситуацию, когда одно пересечение типа 3 на высоте $m$ в $T$ порождает пару пересечений типа 3 на высоте $(m-2)$. Итак, обозначая через $U_i(m)$ число пройденных непродуктивных узлов в поддереве высотой $m$, чей корень имеет тип $i$ ($i = 2, 3$), получаем рекуррентные соотношения:

$$\begin{cases} U_2(m) = U_2(m-1) + U_3(m-1) + 1, \\ U_3(m) = 2U_3(m-2) + 3. \end{cases} \tag{2.3}$$

Результат же таков: для $U_2(m)$ и $U_3(m)$ оценка одинакова — $O(\sqrt{N})$.

Заключаем, что для файла мощности $N$ в худшем случае время запроса составит $O(\sqrt{N})$, даже если выбранное множество окажется пустым. Этот негативный результат для худшего случая противоречит хорошей оценке, полученной методом имитационного моделирования [Bentley (1975)] и подтвержденной эвристическим рассуждением [Bentley, Stanat (1975)].

Обобщение вышеописанного метода на случай $d$ измерений получается непосредственно. Здесь мы будем работать с секущими гиперплоскостями, ортогональными координатным осям, а все, что необходимо определить, — это критерий, по которому следует выбирать ориентации этих секущих гиперплоскостей. Если координаты обозначены через $x_1, x_2, \ldots, x_d$, то одним из возможных критериев будет *циклический сдвиг* на последовательности $(1, 2, \ldots, d)$ индексов координат. Результирующее разбиение $d$-мерного пространства моделируется двоичным деревом с $N$ узлами, которое называется *многомерным двоичным деревом* или *k-D-деревом*[1]. Анализ эффективности также можно обобщить на случай $d$ измерений, однако этого здесь мы делать не будем. Подведем итоги в следующей теореме:

[1]) Выражение «$k$-$D$-дерево» — это принадлежащая Д. Кнуту аббревиатура для «$k$-мерного дерева двоичного поиска». Оно не слишком выразительно, но уже вошло в современный профессиональный жаргон.

**Теорема 2.9.** *С помощью метода k-D-дерева региональный поиск на d-мерном множестве (при $d \geqslant 2$) из N точек можно провести за время $O(dN^{1-1/d} + k)$ с использованием $\theta(dN)$ памяти, если затратить $\theta(dN \log N)$ времени на предобработку. Поэтому метод k-D-дерева дает $(dN, dN^{1-1/d})$-алгоритм.*

Оптимальные оценки по памяти и времени предобработки, к сожалению, не компенсируют обескураживающую оценку времени поиска для худшего случая. Поэтому неудивительно, что в качестве альтернатив были предложены другие методы. Эти методы будут описаны ниже.

### 2.3.3. Метод прямого доступа и его варианты

Неэффективность метода *k-D*-дерева для худшего случая является важным мотивом для исследования методов, обладающих удовлетворительными оценками времени поиска, возможно, за счет ухудшения двух других обсуждавшихся оценок эффективности. Мы начнем с самого простого метода, а затем последовательно будем его улучшать.

Как упоминалось в разд. 2.3.1, самый грубый способ минимизации времени поиска состоит в предварительном вычислении ответов на все возможные при региональном поиске запросы. Поэтому возникает искушение заявить, что время поиска можно сократить до $O(1)$, поскольку единственное обращение к памяти завершило бы поиск. Это, конечно, приводит к постановке головоломного вопроса о кажущемся противоречии с нижней оценкой, полученной в разд. 2.3.1. Скоро мы поймем, что есть одна загвоздка, и все встанет на свои места.

Следуя подходу, введенному в разд. 2.1, предположим, что дано множество $S$ из $N$ точек на плоскости, и проведем горизонтальную и вертикальную прямые через каждую из этих точек. Разумеется, эти прямые разобьют плоскость на $(N+1)^2$ прямоугольных ячеек. Положим, что в регионе $D = [x_1, x_2] \times [y_1, y_2]$ точка $(x_1, y_1)$ принадлежит ячейке $C_1$, а точка $(x_2, y_2)$ — ячейке $C_2$. Если теперь переместить как угодно любую из этих точек, оставляя ее, однако, внутри соответствующей ячейки, то очевидно, что искомое множество (т. е. подмножество $S$, содержащееся внутри $D$) не изменится. Другими словами, пары ячеек формируют классы эквивалентности относительно результатов регионального поиска. Следовательно, число различных регионов, которые нам следовало бы рассмотреть, ограничено сверху числом

$$\binom{N+1}{2} \times \binom{N+1}{2} = O(N^4).$$



Если предварительно вычислить выбираемое множество для каждой из этих пар ячеек (затратив при этом $O(N^5)$ памяти), то мы получили бы схему, в которой поиск заканчивается после единственного обращения к файлу. Однако для достижения этой цели произвольный регион $D$ нужно сопоставить паре ячеек, или, что то же самое, произвольная точка должна быть сопоставлена одной из ячеек. Очевидно, что это можно проделать двумя двоичными поисками на множествах абсцисс и ординат точек из $S$. На эту «нормализацию», однако, уйдет $O(\log N)$ времени, которое, будучи добавленным к $O(1)$ времени на единственное обращение к файлу, разрешает кажущееся противоречие, встреченное нами ранее. Заметим в заключение, что данный метод дает на плоскости $(N^5, \log N)$-алгоритм, слишком обременительный для практического использования из-за затрат памяти.

Чтобы найти пути сокращения памяти, заметим, что одномерный поиск, несмотря на наши безуспешные усилия обобщить его, является оптимальным по обеим мерам (времени поиска и памяти). Это замечание заставляет предположить возможность улучшения вышеописанного «метода ячеек». В самом деле, можно объединить одномерный региональный поиск с локализацией региона методом прямого доступа [Bentley, Maurer (1980)]. Конкретно, для заданного региона $D = [x_1, x_2] \times [y_1, y_2]$ можно реализовать прямой доступ по координате $x$ с последующим одномерным поиском по координате $y$. Для этого требуется, чтобы ко всем различимым упорядоченным парам абсцисс точек из $S$ ($x$-регионы) был обеспечен прямой доступ; каждая такая пара $(x', x'')$ в свою очередь связана с двоичным деревом поиска на ординатах точек, содержащихся в полосе $x' \leqslant x \leqslant x''$. Теперь удобно преобразовать исходную задачу, заменяя каждое действительное значение координаты на ее порядковый номер во множестве координат. Такое преобразование, именуемое *нормализацией*, осуществляется путем операции сортировки на каждом из множеств координат. После нормализации этих данных любой $x$-регион преобразуется за время $O(N \log N)$ в пару целых чисел $(i, j)$ из отрезка $[1, N+1]$ $(i < j)$. Такая пара соответствует адресу в массиве указателей прямого доступа [1]), который отсылает к двоичному дереву с $(j - i)$ листьями. Поэтому общий объем памяти пропорционален

$$\sum_{i=1}^{N} \sum_{j=i+1}^{N+1} (j - i) = [(N+1)^3 - (N+1)]/6 = O(N^3).$$

[1]) Например, если этот массив начинается с адреса $A$, то по паре $(i, j)$ можно однозначно вычислить адрес: $A - 2 + (2N(i-1) + i - i^2)/2 + j$.

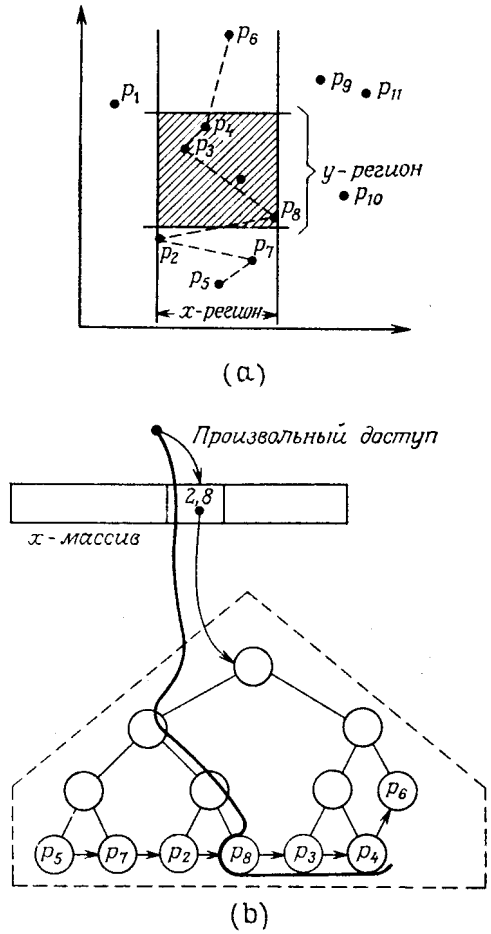



Рис. 2.31. Региональный поиск прямым доступом. Адрес $x$-интервала локализуется методом произвольного доступа после нормализации. По этому адресу находится указатель к двоичному дереву поиска.

Заметим, что, хотя память удалось сократить с $O(N^5)$ до $O(N^3)$, время поиска осталось равным $O(\log N)$, поскольку и нормализация регионов, и одномерный региональный поиск используют $O(\log N)$ времени. Ситуация, характерная для последнего примера, проиллюстрирована на рис. 2.31 (заметим, что точки индексированы в порядке возрастания абсцисс). В массиве $x$ имеется доступ к региону [2, 8], заданному нормализованными абсциссами (рис. 2.31(b)); отсюда указатель направляет поиск к двоичному дереву. Там поиск, *как таковой*, заканчивается локализацией точки $P_8$, после чего производится выборка последовательности $(P_8, P_3, P_4)$.



У нас все еще мало оснований, чтобы быть довольными вышеописанным методом, и не только потому, что затрата $O(N^3)$ памяти остается весьма дорогой платой, но также потому, что в многомерном случае трюк с использованием одномерного поиска можно проделать лишь однажды, тем самым снижая используемую память только в $N^2$ раз. И на самом деле непосредственный анализ показывает, что если этот подход обобщить на случай $d$ измерений, то будет использовано $O(N^{2d-1})$ памяти.

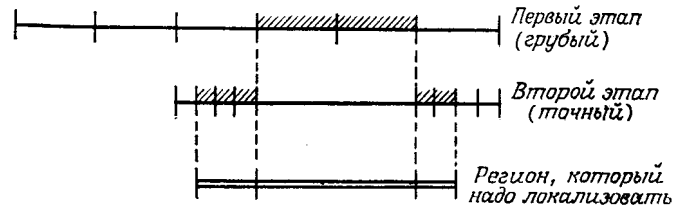



Рис. 2.32. Пример двухэтапной схемы метода прямого доступа.

Очевидно, что именно фаза прямого доступа в этой двухфазной схеме виновна в большой затрате памяти. Поэтому улучшения нужно искать именно в этой фазе. Бентли и Маурер предложили многоэтапный подход [Bentley, Maurer (1980)] [1]), который будет продемонстрирован здесь для случая двух этапов. Идея состоит в том, чтобы последовательно использовать грубую сетку и мелкую сетку (рис. 2.32). Все координаты нормализуются, и грубая сетка состоит из отрезков длиной $k$, при этом на каждом таком отрезке существует мелкая сетка, состоящая из отрезков единичной длины. Отсюда видно, что произвольный регион разбивается не более чем на три части, одна из которых целиком принадлежит грубой сетке, а две других — мелкой сетке (рис.2.32). Каждой сетке соответствует описанный выше массив указателей прямого доступа. В частности, если грубая сетка состоит из $N^\alpha$ шагов $(0 < \alpha \leqslant 1)$, то на грубой сетке всего будет $O(N^{2\alpha})$ отрезков, а для соответствующей структуры данных потребуется $O(N^{2\alpha}) \times N = O(N^{1+2\alpha})$ памяти. Аналогично каждая мелкая сетка состоит из $N^{1-\alpha}$ шагов и всего будет $N^\alpha$ таких сеток, поэтому общая память для них составит величину порядка $N^\alpha \times N^{(1-\alpha)2} \times N^{1-\alpha} = N^{3-2\alpha}$. Минимального значения, равного $O(N^2)$, память достигает при $\alpha = 1/2$ без

[1]) Структуры данных, которые будут введены, первоначально назывались многоуровневыми $k$-регионами.

потери логарифмической оценки для времени регионального поиска. Фактически время поиска возрастает примерно в 3 раза, и мы получаем $O(N^2, \log N)$-алгоритм.

Теперь можно обобщить этот подход, используя последовательность из $l$ сеток возрастающей мелкости, для дальнейшего сокращения памяти при сохранении, однако, оценки времени поиска в пределах $O(l \log N)$. Этот относительно простой анализ приводит к следующей теореме:

**Теорема 2.10.** *Для любого* $\varepsilon$ *такого, что* $0 < \varepsilon < 1$, *региональный поиск на двумерном N-точечном файле можно провести с помощью* $(N^{1+\varepsilon}, \log N)$*-алгоритма, основанного на многоэтапном методе прямого доступа.*

Самый интересный вывод, который можно извлечь из предшествовавшего обсуждения методов прямого доступа, состоит в следующем. Одномерный региональный поиск можно сделать оптимальным. Поэтому следует добиваться преобразования многомерной задачи в серию одномерных задач. Схема прямого доступа — это один из путей реализации такого преобразования. В принципе в двумерном случае произвольный отрезок разбивается на не более чем *три* стандартных отрезка (и в $d$-мерном случае точно так же в конечном итоге получается некое *фиксированное* число стандартных отрезков). Данный подход содержит в себе некоторые идеи метода дерева регионов, который будет описан ниже [1]).

### 2.3.4. Метод дерева регионов и его варианты

Эффективность по времени поиска в методе прямого доступа получена благодаря построению набора «стандартных» отрезков. Выигрыш в памяти при переходе от одноэтапной схемы к двухэтапной получен благодаря сильному сокращению числа таких стандартных отрезков. Возникает стремление к дальнейшему следованию этой линии, т. е. к созданию схемы, минимизирующей число стандартных отрезков. В этом и состоит главная идея метода *регионального дерева.*

Рассмотрим множество на оси $x$, состоящее из $N$ абсцисс, нормализованных до целых чисел из интервала $[1, N]$ по их величине. Эти $N$ абсцисс определяют $N-1$ элементарных отрезков $[i, i+1]$ для $i = 1, 2, \ldots, N-1$. Средством, которое будет

[1]) Однако хронологически метод дерева регионов был создан немного ранее методов прямого доступа [Bentley (1979)]. Тем не менее с чисто идейной стороны его можно считать развитием последних методов. Первые идеи, связанные с понятием дерева регионов, можно найти в работе [Bentley, Shamos (1977)].



использовано для реализации разбиения произвольного отрезка $[i, j]$, является *дерево отрезков*, введенное в разд. 1.2.3.1. Повторим просто для удобства читателя, что произвольный отрезок, концы которого принадлежат множеству из $N$ заданных абсцисс, может быть разбит деревом отрезков $T(1, N)$ на не более $2\lceil \log_2 N \rceil - 2$ стандартных отрезков. Каждый стандартный отрезок отнесен к одному из узлов $T(1, N)$, а те узлы, которые определяют разбиение отрезка $[i, j]$, называются *узлами отнесения* для $[i, j]$.

После такого напоминания не составит труда воспользоваться деревом отрезков при региональном поиске. Начнем, как обычно, с двумерного случая. Дерево отрезков $T(1, N)$ используется при поиске по $x$-координате. Этот поиск определяет уникальное множество узлов (узлов отнесения). Каждый такой узел $v$ соответствует множеству из $(E[v] - B[v])$ абсцисс (определения даны в разд. 1.3.2.1), т. е. множеству из $(E[v] - B[v])$ точек на плоскости. Ординаты этих точек образуют обычное прошитое двоичное дерево для регионального поиска в $y$-направлении. В результате построена новая структура данных, именуемая *деревом регионов*; его первичной структурой является дерево отрезков, заданных на абсциссах точек исходного множества $S$. В каждом узле этого дерева имеется указатель на прошитое двоичное дерево поиска (вторичные структуры). Для нашего постоянного примера эта структура проиллюстрирована на рис. 2.33.

Обобщение на $d$ измерений можно провести весьма естественно. Пусть в $d$-мерном пространстве с координатными осями $x_1, x_2, \ldots, x_d$ задано множество $S$ из $N$ точек. Эти координаты будут обрабатываться в следующем порядке: сначала $x_1$, затем $x_2$ и т. д. Предположим также, что все значения координат нормализованы. Дерево регионов строится рекурсивно следующим образом:

(1) Первичное дерево отрезков $T^*$ соответствует множеству $\{x_1(p)\colon p \in S\}$. Для каждого узла $v$ из $T^*$ обозначим через $S_d(v)$ — множество точек из $S$, проецирующихся на отрезок $[B[v], E[v]]$ по координате $x_1$. Определим $(d-1)$-мерное множество:

$$S_{d-1}(v) \triangleq \{(x_2(p), \ldots, x_d(p))\colon p \in S_d(v)\}.$$

(2) Узел $v$ из $T^*$ имеет указатель на дерево регионов для $S_{d-1}(v)$.

Перед тем как приступить к анализу эффективности данного метода, заметим, что дерево отрезков можно считать не только инструментом для разбиения любого отрезка на логарифмическое число кусков, но также и способом применения метода «разделяй и властвуй» к региональному поиску.

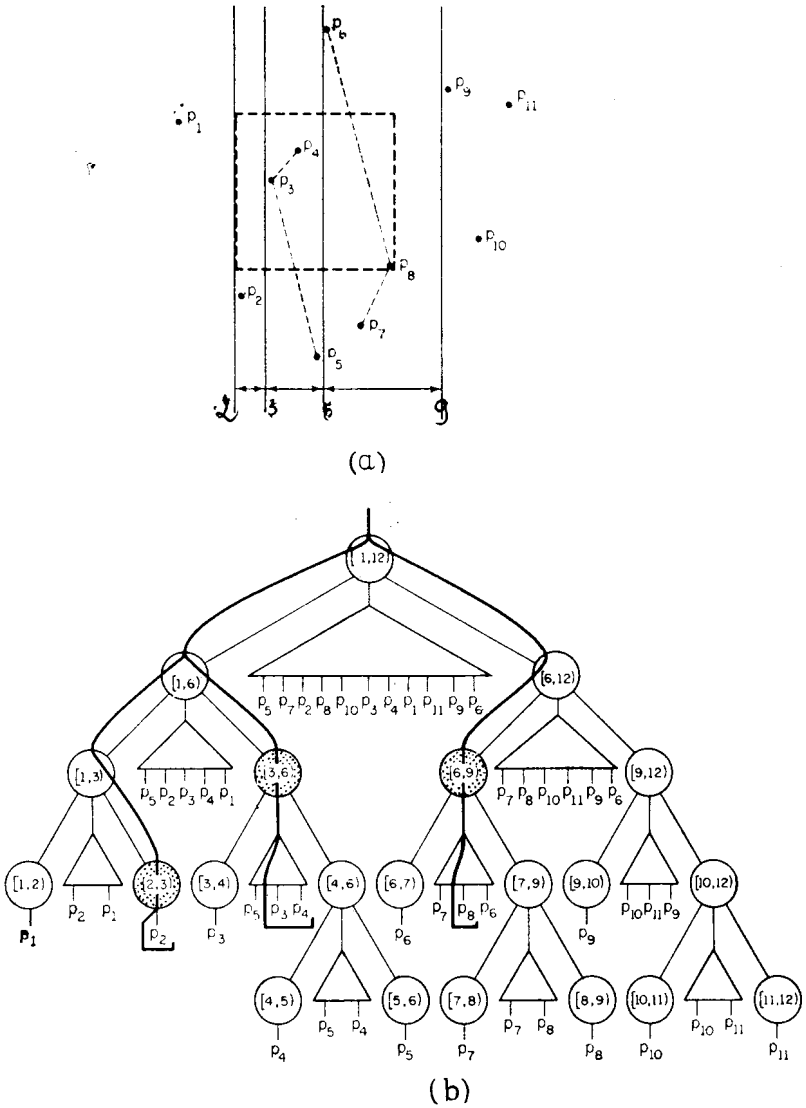



Рис. 2.33. Иллюстрация применения метода дерева регионов к нашему постоянному примеру. Исходный регион разбит тремя стандартными отрезками (a). Поисковые действия показаны на рис. (b).

Переходя теперь к анализу сложности метода, заметим, во-первых, что при $d \geqslant 2$ каждый узел отнесения из проекции запросного региона на ось $x_1$ (их всего $O(\log N)$) порождает отдельную $(d-1)$-мерную задачу регионального поиска. В частности, узел $v$ порождает поиск на $n(v) \triangleq E[v] - B[v]$ точках. Обозначая через $Q(N, d)$ время поиска для файла из $N$ штук $d$-мерных точек, получаем простое рекуррентное соотношение:

$$Q(N, d) = O(\log N) + \sum_{\substack{v \in \text{множеству узлов} \\ \text{отнесения из проекции} \\ \text{запросного региона} \\ \text{на ось } x_1}} Q(n(v), d-1). \qquad (2.4)$$



Первый член в правой части этого уравнения появился из-за поиска на первичном дереве отрезков, а второй член объясняется наличием $(d-1)$-мерных подзадач. Поскольку существует не более $2\lceil \log_2 N \rceil - 2$ узлов отнесения и $n(v) \leqslant N$ (очевидно), то получаем

$$Q(N, d) = O(\log N)\, Q(N, d-1).$$

Поскольку $Q(N, 1) = O(\log N)$ для двоичного поиска, то получаем оценку времени поиска:

$$Q(N, d) = O((\log N)^d). \qquad (2.5)$$

Обозначая через $S(N, d)$ память, занятую деревом регионов, получаем рекуррентное соотношение

$$S(N, d) = O(N) + \sum_{\substack{\text{по всем узлам } v \\ \text{первичного дерева}}} S(n(v), d-1), \qquad (2.6)$$

где первый член в правой части соответствует памяти, занятой первичным деревом, а второй член соответствует всем $(d-1)$-мерным деревьям. Оценка второго члена очень проста, если $N$ есть степень 2; в противном случае будет использована столь же эффективная простая аппроксимация. Существует не более двух узлов $v$ с $n(v) = 2^{\lceil \log_2 N \rceil - 1}$, не более четырех — с $n(v) = 2^{\lceil \log_2 N \rceil - 2}$ и т. д. Поэтому эту сумму можно оценить сверху следующим образом:

$$\sum_{\substack{\text{по всем узлам } v \\ \text{первичного дерева}}} S(n(v), d-1) \leqslant \sum_{i=0}^{\lceil \log_2 N \rceil} 2^{\lceil \log_2 N \rceil - i} \cdot S(2^i, d-1).$$

Учитывая эту аппроксимацию и замечая, что $S(N, 1) = O(N)$ — память под прошитое двоичное дерево, получаем решение

$$S(N, d) = O(N \log^{d-1} N). \qquad (2.7)$$

Для оценки времени предварительной обработки можно следовать той же самой схеме рассуждений; оставим это в качестве весьма простого упражнения. В самом деле, в этом случае для оценки эффективности применимо рекуррентное соотношение, идентичное (2.6), со всеми его следствиями. Теперь можно подвести итог предшествовавшему обсуждению в следующей теореме:

**Теорема 2.11.** *Региональный поиск на d-мерном файле из N точек можно провести* $(N\log^{d-1}N,\ \log^{d}N)$*-алгоритмом, если затратить* $O(N\log^{d-1}N)$ *времени на предварительную обработку. Этот алгоритм основан на методе дерева регионов.*

В частности, при $d=2$ эти меры — время запроса, память и время предварительной обработки — выглядят так: $O(\log^2 N+k)$, $O(N\log N)$ и $O(N\log N)$ [1]) соответственно. Достигнута весьма привлекательная оценка памяти, хотя и при некотором увеличении времени поиска.

Прежде чем согласиться с таким соотношением оценок, которое, вероятно, внутренне присуще этой задаче, стоит еще раз осмыслить вопрос о возможной потере эффективности процесса поиска. Уиллард [Willard (1978)] и Люкер [Lueker (1978)], именно так и поступив, независимо друг от друга предложили вариант схемы, в которой достигается оптимальное время поиска для двух измерений. Теперь обсудим эту идею.

Каждый узел $v$ в первичном дереве отрезков связан со *списком* $Y(v)$ (а не с двоичным деревом!) тех точек, которые проецируются на отрезок этого узла. Этот список упорядочен так, чтобы ординаты указанных точек не убывали. Чтобы осуществить выборку, необходимо найти первый элемент подсписка, который надлежит выбрать. В схеме метода дерева регионов с этой целью делается двоичный поиск для каждого узла $v$ из первичного дерева. Если рассмотреть два потомка узла $v$, мы увидим, что $Y(\text{ЛСЫН}[v])$ и $Y(\text{ПСЫН}[v])$ образуют разбиение $Y(v)$. Предположим, теперь, что $p$ — начальный (крайний слева) элемент $Y(v)$, подлежащий выборке, — известен. Без потери общности положим, что если $p\in Y(\text{ЛСЫН}[v])$, то существует единственный элемент $p'$ в $Y(\text{ПСЫН}[v])$ такой, что $y(p')$ — наименьшее число, для которого $y(p)\leqslant y(p')$. Следовательно, пара указателей от $p$ к $Y(v)$ однозначно определяет соответствующую пару начальных элементов в $Y$-списках для потомков $v$.

Теперь относительно просто можно сделать выводы. В дереве регионов можно заменить каждое прошитое двоичное дерево его листовым списком для всех узлов, кроме корня. В качестве же корневой структуры сохранится прошитое двоичное дерево, поскольку оно необходимо для реализации поиска за логарифмическое время. Для каждого нелистового узла в дереве отрезков хранится пара указателей от каждого из элементов его $Y$-списка к паре $Y$-списков его потомков. Этот метод иногда называют расслоением, а модифицированное дерево регионов можно называть расслоенным деревом регионов. Использование модифицированного дерева регионов для случая нашего постоянного примера показано на рис. 2.34.

[1]) Заметим, что метод дерева регионов можно очевидным образом адаптировать для режима подсчета точек и получить следующие оценки для трех мер сложности: $O(\log^2 N)$, $O(N\log N)$ и $O(N\log N)$. Это подтверждает один из результатов, приведенных в разд. 2.1 (табл. I).



Затраты памяти не изменились, поскольку и прошитое двоичное дерево с $s$ листьями, и список с $s$ записями и парой внешних указателей на каждую запись требуют по $O(s)$ памяти. Однако время поиска сократилось до $O(\log N)$, поскольку двоичный поиск в корне осуществляет начальную локализацию левого конца $y$-региона. Эта структура указателей реализует за константное время вычисление левого конца для каждого узла, встреченного при «проходе» дерева отрезков. При обобщении этой схемы на случай большего числа измерений сохраняется эффект уменьшения оценки времени поиска в $\log N$ раз. Итак, получаем:

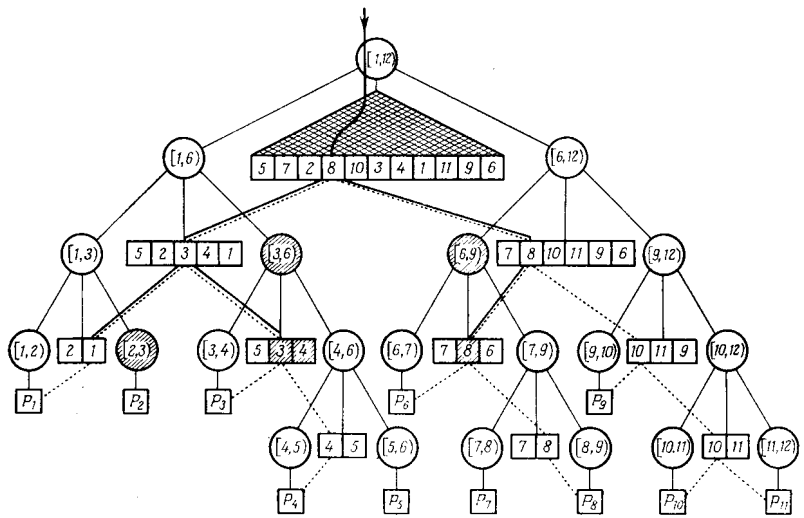



Рис. 2.34. Иллюстрация обхода модифицированного дерева регионов для нашего постоянного примера. Показана только часть указателей в $Y$-списках.

**Теорема 2.12.** *Региональный поиск в d-мерном файле (при* $d\geqslant 2$*) из N точек можно реализовать* $(N\log^{d-1}N,\ \log^{d-1}N)$*-алгоритмом, если затратить* $(N\log^{d-1}N)$ *времени на предварительную обработку. Этот алгоритм основан на модификации алгоритма Уилларда — Люкера для дерева регионов, известной также как расслоенное дерево регионов.*

## 2.4. Замечания и комментарии

Со времени появления статьи Добкина и Липтона был достигнут заметный прогресс в решении задачи о локализации точки. Как указывалось в разд. 2.2, все представленные методы формируют новые геометрические объекты, помогающие при поиске. Например, в методе полос плоскость разбивается на элементарные пустые трапеции, обладающие парой горизонтальных сторон; два этапа двоичного поиска определяют одну-единственную трапецию. В методе детализации триангуляции путем перетриангуляции частей ППЛГ создается удобное малое множество *новых* треугольников, на котором легко проводить поиск. В методе трапеций ведется эффективное обследование исходного графа путем создания трапеций, обладающих парой горизонтальных сторон, которые быстро сужают область поиска. Наконец, в методе цепей из плоского прямолинейного графа выделяется множество цепей ломаных линий, разбивающих плоскость на монотонные многоугольники, на которых легко вести поиск. Хотя этот метод почти оптимален, его можно улучшить и получить оптимальный и при том весьма простой алгоритм: это улучшение основано на удачном сочетании метода цепей, структуры указателей Уилларда — Люкера и «фильтрующего поиска». Ниже будет кратко описан фильтрующий поиск.

*Фильтрующий поиск* [Chazelle [1983c)] [1]) это один из общих методов решения задач поиска в режиме отчета, основанный на вычислительно разумной идее о том, что можно позволить доступ к супермножеству искомого множества, если при этом получается выигрыш вычислительных ресурсов (таких как память или время поиска), при условии что мощность супермножества превосходит не более чем заданную (малую) мультипликативную константу — мощность искомого множества. Другими словами, это супермножество является «черпаком», который выбирают, а затем «фильтруют», чтобы выделить искомое множество; поэтому «фильтрующий поиск» можно не без оснований называть поиском типа «черпай и фильтруй». Этот подход был успешно применен к различным задачам, таким как запросы с ортогональным регионом, поиск близости (см. гл. 6), к задачам пересечения и т. д.

Возвращаясь к локализации точки, покажем теперь, как методология фильтрующего поиска и структура указателей Уилларда — Люкера могут быть объединены с методом цепей [Edelsbrunner, Guibas, Stolfi (1985)]. В структуре первичного дерева (где каждый узел $v$ связан с одной из монотонных цепей $C$) есть указатель от $v$ к вторичной структуре, главной компонентой которой является последовательность $T(v)$, содержащая не только ординаты ребер, отнесенных к цепи $C$ (как в версии Ли — Препараты), но и дополненная, кроме того, *каждым вторым членом* последовательности, полученной слиянием (дополненных) последовательностей для двух потомков $v$. Как и в схеме Уилларда — Люкера для дерева регионов, в этой последовательности $T(v)$ поиск выполняется с помощью двоичного дерева только в корне первичной структуры. Для всех остальных узлов доступ к $T(v)$ осуществляется через структуру указателей, что позволяет получить оценку для времени поиска $O(\log N)$. За счет распространения «каждого второго члена» с одного уровня на следующий обеспечивается линейных расход памяти, однако при этом, возможно, будет просмотрено вдвое больше областей, чем это необходимо, и их следует профильтровать в духе метода фильтрующего поиска.

[1]) Исходные предварительные идеи этого подхода можно проследить в работе [Bentley, Maurer (1979)].



Все методы, представленные в этой главе для локализации точки на плоскости, настроены на работу для худшего случая. В классе методов с хорошими оценками в среднем недавно опубликованный метод «черпаков» [Edahiro, Kokubo, Asano (1983)] экспериментально превосходит все ранее изложенные методы, а среди последних лучшую экспериментальную оценку показал метод трапеций.

Что касается регионального поиска, то в разд. 2.3 было показано, что единственный метод, использующий линейную память (метод $k$-$D$-дерева), обладает большим временем поиска. Поэтому кажется, что избыток памяти является ключевым условием быстрого поиска. Как метод прямого доступа, так и метод дерева регионов служат примерами этой идеи. В действительности это примеры приложений, названных Бентли и Сэйксом «задачами поиска, допускающими декомпозицию» [Saxe, Bentley (1979)], в которых ответ на запрос относительно всего пространства получается как комбинация (в данном случае как сумма) ответов на запросы относительно подходящего набора частей этого пространства. Остается открытым важный вопрос о существовании $(N, \log N)$-алгоритма для 2-мерного регионального поиска. Недавно Чазелле [Chazelle (1983c)], используя метод фильтрующего поиска, смог понизить потребность в памяти с $N \log N$ до $N \log N/\log\log N$, показав тем самым, что $N \log N$ не является нижней оценкой.

Другие варианты задачи регионального поиска можно классифицировать по типу области поиска. В одном из таких классов в качестве области поиска выбирают $k$-угольник (*поиск в многоугольном регионе*); предполагая, как обычно, что в файле $N$ точек, Уиллард создал $(N, kN^{0,77})$-алгоритм [Willard (1982)], который был позже улучшен до $(N, kN^{0,695})$-алгоритма [Edels-

brunner, Welzl (1983)]. Известен интересный частный случай этой задачи, когда в качестве многоугольного региона берется полуплоскость; для *поиска в полуплоскости* недавно был предложен оптимальный результат [Chazelle, Guibas, Lee (1983)] (этот метод неприменим, однако, к соответствующей задаче поиска типа подсчета). Очень интересен также случай, когда поисковая область имеет форму круга (*круговой региональный поиск*, или *поиск в диске*). Если радиус такого диска фиксирован, то задачу можно решить методом локализации, где плоскость разбита на области так, что все точки в заданной области порождают идентичные ответы. Этот подход можно реализовать, прибегнув, например, к методу трапеций и получив тем самым $(N^3 \log N, \log N)$-алгоритм. Однако этот алгоритм намного превзойден $(N, \log N)$-методом, предложенным недавно [Chazelle, Edelsbrunner (1984)]. Еще более интересен *поиск в переменном диске*, где круговая область может обладать произвольным радиусом и центром; лучший из известных методов решения этой задачи имеет сложность $(N(\log N \log\log N)^2 \log N)$ и будет описан в гл. 6 [Chazelle, Cole, Preparata, Yap (1984)].

Другие задачи поиска работают с различными геометрическими объектами, где пара (запрос, элемент файла) более не соответствует типам: (точка, область) — как при локализации точки или (область, точка) — как при региональном поиске. Например, могут встретиться такие пары, как (многоугольник, многоугольник), (отрезок, отрезок) и т. д.; в подобных случаях искомое отношение скорее является «пересечением», вот почему эти задачи лучше изучить в общем контексте задач о пересечениях (гл. 7). В частности, когда пара (запрос, элемент файла) имеет вид (прямоугольный регион, прямоугольный регион), то получается семейство интересных задач типа пересечения или принадлежности, которые уместнее обсудить в гл. 8.

И наконец, значительный прогресс достигнут при изучении динамических поисковых структур, т. е. таких структур, которые в дополнение к запросам поддерживают операции вставки и удаления. В то время как одномерные динамические структуры (АВЛ-деревья и т. п.) были известны уже более двадцати лет, подступы к решению многомерной задачи начали искать относительно недавно. Следом за пионерской работой Бентли [Bentley (1979)] были разработаны общие методы преобразования статических структур, удовлетворяющих некоторым слабым ограничениям, в динамические.

Необходимо упомянуть о методе Ван Леювена и Вуда [van Leeuwen, Wood (1980)], общей идеей которого является организация файла как набора отдельных структур данных, для того чтобы любая корректировка строго ограничивалась одной (или, возможно, небольшим фиксированным числом) из них; однако, чтобы избежать переноса центра тяжести с корректировок на поисковую работу, необходимо не допускать чрезмерной фрагментации, поскольку запросы обычно относятся ко всей совокупности данных. Самое современное и всестороннее исследование состояния дел в области «динамизации» — это диссертация Овермарса [Overmars (1983)], и мы настоятельно рекомендуем обратиться к ней.



## 2.5. Упражнения

1. Провести более глубокий анализ метода детализации триангуляции Киркпатрика и выбрать $K$ таким, чтобы достигнуть значения $\alpha \cong 59/63$. (Указание: оценить отдельно узлы со степенью больше $K$ и узлы со степенью не больше $K$.)

2. *Зейдель.* Пусть $S$ — множество из $N$ точек на плоскости и $A$ — планарное подразбиение с $\theta(N)$ областями.

(а) Показать, что необходимо $\Omega(N \log N)$ времени для локализации всех точек из $S$ в $A$.

(б) Предположим теперь, что известна триангуляция $S$. Триангуляция содержит информацию о том, как точки из $S$ расположены относительно друг друга. Эта информация могла бы помочь при локализации всех точек из $S$ в $A$. Показать, что, несмотря на знание триангуляции $S$, все же понадобится $\Omega(N \log N)$ времени для локализации всех точек из $S$ в $A$.

3. Доказать теорему 2.10. Предположить, что поиск состоит из $l$ этапов с возрастанием мелкости сеток, и получить $(N^{c_2}, c_1 \log N)$-алгоритм с минимальным значением $c_2$. Выразить $c_1$ и $c_2$ как функции от $l$.

4. Можно ли модифицировать дерево регионов в варианте Уилларда — Люкера так, чтобы получить $(N \log^{d-1} N, \log^{d-1} N)$-алгоритм для задачи регионального поиска типа подсчета при $d \geqslant 2$? Строго обосновать свой ответ.

5. *Эдельсбруннер — Гибас — Столфи.* Рассмотреть метод цепей для задачи плоской локализации точки на $N$-вершинной карте и модифицировать его следующим образом. Начать с назначения этих цепей узлам дерева $T$ первичной структуры, как это показано в работе [Lee, Preparata (1978)] (см. разд. 2.2.2.2), так что каждому узлу $v$ из $T$ назначена последовательность $Y(v)$, состоящая из концов ребер его цепи; затем для каждого нелистового узла $v$ сформировать $Y^*(v)$, дополняя $Y(v)$ всеми членами последовательности ОБЪЕДИНЕНИЕ($Y^*$(ЛСЫН[$v$]),$Y^*$(ПСЫН[$v$])). Каждый список $Y^*(v)$ представляет разбиение вертикальной прямой: создать указатель от каждого отрезка из $Y^*(v)$ к каждому из отрезков в $Y^*$(ЛСЫН[$v$]) или в $Y^*$(ПСЫН[$v$]), с которым он пересекается. Это определяет структуру данных для поиска $T^*$.

(а) Показать, что для запоминания $T^*$ можно затратить $O(N)$ памяти.

(б) Показать, что локализация точки на плоскости требует $O(\log N)$ времени.

6. Применить метод локализации при решении следующей задачи (круговой региональный поиск с фиксированным радиусом): даны $N$ точек на плоскости и число $d > 0$, требуется перечислить точки, находящиеся на расстоянии, не превышающем $d$ от заданной пробной точки $q$ (возможно, затратив при этом дополнительно логарифмическое время).

ГЛАВА 3

# Выпуклые оболочки: основные алгоритмы

Задача вычисления (построения) выпуклой оболочки не только является центральной в целом ряде приложений, но и позволяет разрешить ряд вопросов вычислительной геометрии, на первый взгляд не связанных с ней. Построение выпуклой оболочки конечного множества точек, и особенно в случае точек на плоскости, уже довольно широко и глубоко исследовано и имеет приложения, например, в распознавании образов [Akl, Toussaint (1978); Duda, Hart (1973)], обработке изображений [Rosenfeld (1969)], а также в задаче раскроя и компоновки материала [Freeman (1974); Sklansky (1972); Freeman-Shapira (1975)].

Понятие выпуклой оболочки множества точек $S$ является естественным и простым. В соответствии с определением — это наименьшее выпуклое множество, содержащее $S$. Чтобы наглядно представить это понятие в случае, когда $S$ — конечное множество точек на плоскости, предположим, что это множество охвачено большой растянутой резиновой лентой. Когда лента освобождается, то она принимает форму выпуклой оболочки.

Несмотря на интуитивную ясность понятия выпуклой оболочки, история создания алгоритмов вычисления выпуклых оболочек наглядно показывает общую картину в области разработки и исследования алгоритмов. К сожалению, приведенное выше простое определение выпуклой оболочки неконструктивно. Поэтому необходимо определить некоторые понятия, способствующие разработке алгоритмов.

Предметом этой главы является построение выпуклой оболочки на плоскости и в пространствах более высокой размерности. В следующей главе будут рассмотрены приложения, различные варианты задачи о выпуклой оболочке и некоторые связанные с ней, но отличные по существу задачи. Чтобы избежать повторений, удобно ввести подходящую совокупность понятий,



относящихся к выпуклой оболочке в пространстве произвольной размерности (Grünbaum (1967); Rockafellar (1970); McMullen, Shephard (1971); Klee (1966)]. В случае плоскости и трехмерного пространства эти понятия приобретают очень простой смысл. Этому посвящен следующий раздел. В дальнейшем мы часто будем ссылаться на понятия, введенные в разд. 1.3.1.

## 3.1. Предварительные сведения

Считая, что точки располагаются в $d$-мерном пространстве, будем использовать общепринятые обозначения для точек и $d$-мерных векторов, выходящих из начала системы координат в $E^d$.

Начнем с рассмотрения понятия аффинного множества.

**Определение 3.1.** Пусть в $E^d$ заданы $k$ различных точек $p_1$, $p_2$, ..., $p_k$. Множество точек $p$ таких, что

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \ldots + \alpha_k p_k \quad (\alpha_j \in \mathbb{R},\ \alpha_1 + \alpha_2 + \ldots + \alpha_k = 1), \tag{3.1}$$

называются *аффинным множеством*, порожденным точками $p_1$, $p_2$, ..., $p_k$, а $p$ является *аффинной комбинацией* точек $p_1, p_2, \ldots, p_k$.

Заметим, что аффинная комбинация представляет частный случай понятия линейной комбинации, возникающий при введении дополнительного условия $\alpha_1 + \ldots + \alpha_k = 1$. В случае $k=2$ аффинное множество — это прямая, проходящая через две заданные точки. Очевидно, что аффинными множествами являются точки, прямые линии, плоскости, гиперплоскости и т. п., т. е. все объекты, с которыми связано интуитивное представление о «прямолинейности». В действительности термин *прямолинейный объект* является синонимом аффинного множества (наряду с термином «аффинное многообразие»). Между векторными подпространствами и аффинными множествами имеется взаимосвязь, состоящая в том, что каждое аффинное множество получается из некоторого векторного подпространства (*линейного множества*) в результате преобразования *переноса* (на некоторый фиксированный вектор). Например, в пространстве $E^3$ линейными множествами являются прямые линии и плоскости, проходящие через начало координат, и сама точка начала координат, в то время как аффинными множествами являются произвольные точки, прямые линии и плоскости.

Пусть в $E^d$ заданы $k$ точек $p_1$, $p_2$, ..., $p_k$. Если $(k-1)$ векторов $p_2 - p_1$, ..., $p_k - p_1$ являются линейно независимыми, то эти точки называются *аффинно независимыми*. Интуитивно по-

нятный смысл этого определения состоит в том, что мы выполняем параллельный перенос множества точек так, что одна из точек (в нашем случае $p_1$) перемещается в начало координат, и проверяем линейную независимость получившегося множества векторов. Очевидно, что выбор точки, перемещаемой в начало координат, является несущественным. Если заданы $k$ аффинно независимых точек $p_1, p_2, \ldots, p_k$, то они образуют аффинный базис $(k-1)$-мерного аффинного множества (т. е. размерность аффинного множества равна размерности линейного множества, из которого оно получается в результате переноса).

**Определение 3.2.** Пусть в пространстве $E^d$ задано подмножество $L$. *Аффинной оболочкой* $\mathrm{aff}(L)$ множества $L$ называется наименьшее аффинное множество, содержащее $L$.

Другими словами, для любой пары точек из $L$ прямая, определяемая этими точками, целиком принадлежит $\mathrm{aff}(L)$. Так, например, аффинной оболочкой отрезка является прямая, аффинной оболочкой плоского многоугольника — плоскость и т. д.

Теперь перейдем к рассмотрению понятия выпуклого множества, порожденного конечным числом точек.

**Определение 3.3.** Пусть в пространстве $E^d$ заданы $k$ различных точек $p_1, p_2, \ldots, p_k$. Множество точек

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \ldots + \alpha_k p_k \quad (\alpha_j \in \mathbb{R},\ \alpha_j \geqslant 0,\ \alpha_1 + \alpha_2 + \ldots + \alpha_k = 1) \tag{3.2}$$

называется *выпуклым множеством*, порожденным точками $p_1, p_2, \ldots, p_k$, а $p$ называется *выпуклой комбинацией* $p_1, p_2, \ldots, p_k$.

Вновь заметим, что понятие выпуклой комбинации является сужением понятия аффинной комбинации, получающимся в результате введения дополнительного условия $\alpha_j \geqslant 0$, $j = 1, 2, \ldots, k$. В случае $k = 2$ результирующим выпуклым множеством является отрезок, соединяющий заданные точки. Размерность выпуклого множества равна размерности его аффинной оболочки. Например, размерность плоского выпуклого многоугольника равна двум, так как его аффинной оболочкой является плоскость. Теперь можно ввести следующее важное понятие.

**Определение 3.4.** Пусть в пространстве $E^d$ задано произвольное подмножество $L$ точек этого пространства. *Выпуклой оболочкой* $\mathrm{conv}(L)$ подмножества $L$ называется наименьшее выпуклое множество, содержащее $L$.

Далее мы будем иметь дело лишь со случаем, когда $L$ содержит конечное число точек (что соответствует выпуклому множеству, порожденному конечным числом точек). Чтобы охарактеризовать структуру $\mathrm{conv}(L)$ конечного множества точек $L$, необходимо обобщить понятия выпуклого многоугольника и выпуклого многогранника.



**Определение 3.5.** *Полиэдральным множеством* в $E^d$ называется пересечение конечного множества замкнутых полупространств (полупространство — это часть $E^d$, расположенная по одну сторону некоторой гиперплоскости).

Заметим, что полиэдральное множество является выпуклым, так как полупространство является выпуклым и пересечение выпуклых множеств также является выпуклым множеством. В частности, выпуклые плоские многоугольники и трехмерные многогранники — как они определены в разд. 1.3.1 — являются примерами (конечных) полиэдральных множеств. В общем случае мы будем называть конечное $d$-мерное полиэдральное множество *выпуклым $d$-политопом* (или, коротко, $d$-политопом, или просто политопом).

Следующая теорема характеризует выпуклые оболочки в нужном нам плане.

**Теорема 3.1** [McMullen, Shephard (1971)]. *Выпуклая оболочка конечного множества точек в $E^d$ является выпуклым политопом. Наоборот, каждый выпуклый политоп является выпуклой оболочкой некоторого конечного множества точек.*

Выпуклый политоп задается описанием его границы, состоящей из *граней*. Каждая *грань* выпуклого политопа является выпуклым множеством (т. е. выпуклым политопом более низкой размерности); *$k$-грань* обозначает $k$-мерную грань (т. е. грань, аффинная оболочка которой имеет размерность $k$). Если политоп $P$ имеет размерность $d$, то его $(d-1)$-грани называются *гипергранями*, $(d-2)$-грани — *подгранями*, 1-грани — *ребрами*, а 0-грани — *вершинами*. Ясно, что ребра и вершины сохраняют свое привычное значение в пространствах любой размерности. Для 3-политопа гиперграни являются плоскими многоугольниками, а подграни и ребра совпадают. Как мы уже видели ранее, эти четыре класса граней играют важную роль в алгоритмах построения выпуклой оболочки. Для единообразия может оказаться полезным называть $d$-политоп $d$-гранью, а пустое множество в такой терминологии превращается в $(-1)$-грань. Если $P$ является выпуклой оболочкой конечного множества точек $S$ в $E^d$, то грань политопа $P$ является выпуклой оболочкой подмножества $T$ множества $S$ (т. е. она определяется подмножеством множества $S$). Однако не все подмножества $S$ определяют грани.

Политопы некоторых типов заслуживают особого внимания. $d$-политоп $P$ называется *$d$-симплексом* (или кратко *симплексом*), если он является выпуклой оболочкой $(d+1)$ аффинно независимых точек. В этом случае каждое подмножество из этих $d$ вершин само является симплексом и является гранью $P$. Таким образом, каждая $k$-грань содержит $2^{k+1}$ граней [1]) (размерностей $k$, $k-1$, ..., 0, $-1$). Например, для $d=0$, 1, 2 и 3 соответствующий симплекс является точкой, ребром, треугольником и треугольной пирамидой. Заметим, что треугольная пирамида (3-грань в соответствии с принятым соглашением) содержит одну 3-грань (сама пирамида), четыре 2-грани (треугольники), шесть 1-граней (ребра), четыре 0-граней (вершины) и одну $(-1)$-грань (пустое множество), что вместе составляет $16 = 2^4$ граней.

*Симплициальным* называется $d$-политоп, каждая гипергрань которого является симплексом. Это эквивалентно тому, что каждая грань симплициального $d$-политопа содержит в точности $d$ подграней. С другой стороны (можно было бы сказать «двойственной»), $d$-политоп называется *простым*, если каждая из его вершин инцидентна в точности $d$ ребрам. В самом деле, легко понять, что симплициальный политоп является двойственным по отношению к простому политопу при понимании двойственности, как это принято в общей топологии, как отображения, отображающего множество размерности $s \leqslant d$ в множество размерности $d-s$ (обратите также внимание на более специальное отношение двойственности, воплощаемое полярными преобразованиями, рассмотренными в разд. 1.3.3). Симплициальные и простые политопы являются чрезвычайно важными не только из-за их структурной привлекательности и — простите за игру слов — простоты, но также ввиду того, что они естественным образом возникают в двух типичных (и двойственных!) ситуациях. В самом деле, обратимся для конкретности к хорошо знакомому трехмерному пространству. Выпуклая оболочка конечного множества точек, находящихся в общем положении, является симплициальным 3-политопом (вероятность того, что более трех точек лежат в одной плоскости, равна нулю), вместе с тем пересечение конечного множества полупространств, находящихся в общем положении, является простым 3-политопом.

Комбинаторной природе политопов, и в частности взаимосвязи между числом граней и размерностью, было уделено значительное внимание. Достаточно напомнить здесь, что число

[1]) Действительно, в этом случае множество всех подмножеств множества вершин является множеством граней. Как хорошо известно, диаграмма Хассе для этого множества представляет $(k+1)$-мерный куб. Такая диаграмма обычно называется *графом граней* политона.



$F(d, N)$ гиперграней $d$-политопа с $N$ вершинами может доходить до (см. [Klee (1966)])

$$F(d, N) = \begin{cases} \dfrac{2N}{d} \cdot \dbinom{N - \frac{d}{2} - 1}{\frac{d}{2} - 1} & \text{для четных } d, \\ 2 \cdot \dbinom{N - \left\lfloor \frac{d}{2} \right\rfloor - 1}{\left\lfloor \frac{d}{2} \right\rfloor} & \text{для нечетных } d. \end{cases} \tag{3.3}$$

Короче можно сказать, что $F(d, N) = O\left(N^{\lfloor d/2 \rfloor}\right)$. То, что число гиперграней в худшем случае растет экспоненциально в зависимости от числа вершин и наоборот (в соответствии с принципом двойственности), приводит к серьезным затруднениям при представлении $d$-политопов для больших значений $d$. К счастью, ситуация оказывается значительно более простой для таких важных случаев, как $d=2, 3$. В частности, для $d=2$ политоп является выпуклым многоугольником. Важно понимать, что многоугольник — независимо от того, является ли он выпуклым или нет, — представляет *упорядоченную последовательность* вершин. Такую последовательность можно адекватно представить как массивом, так и двусвязным списком.

В трехмерном случае проблемы, связанные с представлением, также не очень серьезные. Многогранник может быть полностью определен перечислением его вершин, ребер и граней. В соответствии с формулой Эйлера (разд. 1.3.1) число вершин ребер и граней трехмерного многогранника связано линейным соотношением, откуда следует, что для полного описания (представления) многогранника с $N$ вершинами достаточно объема памяти $O(N)$. Более того, скелет такого многогранника (множество его ребер) является планарным графом [1]), так что мы можем представить многогранник, используя любую структуру данных, подходящую для представления планарного графа (такие как список смежности или реберный список с двойными связями, описанные в разд. 1.2.3.2).

Прежде чем переходить к описанию алгоритмов, уместно дать формальную постановку задачи и указать на важный вопрос относительно нижних оценок сложности этой задачи. Так как алгоритмы построения выпуклой оболочки имеют дело с границей этой выпуклой оболочки, то мы будем обозначать символом $\mathrm{CH}(L)$ границу выпуклой оболочки $\mathrm{conv}(L)$. Заметим, однако, что в соответствии с общепринятой практикой как $\mathrm{conv}(L)$, так и $\mathrm{CH}(L)$ будут называться «выпуклой оболочкой».

[1]) Это следует из теоремы Штейница. См. [Grünbaum, (1967), p. 235].

## 3.2. Постановка задачи и нижние оценки сложности

Начнем с формулировки двух основных вариантов задачи построения выпуклой оболочки.

**Задача ВО1** (ВЫПУКЛАЯ ОБОЛОЧКА). В $E^d$ задано множество $S$, содержащее $N$ точек. Требуется построить их выпуклую оболочку (т. е. полное описание границы $\mathrm{CH}(S)$).

**Задача ВО2** (КРАЙНИЕ ТОЧКИ). В $E^d$ задано множество $S$, содержащее $N$ точек. Требуется определить те из них, которые являются вершинами выпуклой оболочки $\mathrm{conv}(S)$.

Должно быть понятно, что задача ВО1 асимптотически является по крайней мере такой же сложной, как и задача ВО2, так как решение задачи ВО1 превращается в решение задачи ВО2, если мы просто скопируем множество вершин, порождаемое в результате решения первой задачи, представив его в виде неупорядоченного списка точек. То есть одна задача преобразуется в другую, или

КРАЙНИЕ ТОЧКИ $\propto_N$ ВЫПУКЛАЯ ОБОЛОЧКА.

Естественно задать вопрос, является ли первая задача асимптотически более простой, чем вторая, или они имеют в действительности одинаковую сложность. Так как любая совокупность точек в двумерном пространстве тривиальным образом вкладывается в пространство $E^d$ для $d > 2$, то любой результат для нижней оценки, полученный для $d = 2$, заведомо остается справедливым для $d > 2$. Поэтому в дальнейшем обсуждении мы будем рассматривать двумерные варианты задач ВО1 и ВО2.

Начнем с рассмотрения задачи ВО1 о выпуклой оболочке на плоскости. То, что вершины многоугольника, являющегося выпуклой оболочкой, следуют в определенном порядке (в действительности мы можем говорить об упорядоченной выпуклой оболочке), указывает на естественную связь с задачей сортировки. В самом деле, следующая теорема устанавливает тот факт, что любой алгоритм решения задачи ВО1 должен быть способен выполнять сортировку.

**Теорема 3.2.** *Задача сортировки сводима за линейное время к задаче построения выпуклой оболочки, и, следовательно, для нахождения упорядоченной выпуклой оболочки $N$ точек на плоскости требуется время $\Omega(N \log N)$.*

*Доказательство.* Мы продемонстрируем процедуру сведения, а сформулированное в теореме заключение является следствием утверждения 1 из разд. 1.4. Пусть заданы $N$ положительных действительных чисел $x_1, \ldots, x_N$. Необходимо показать, как можно использовать алгоритм построения выпуклой оболочки для сортировки этих чисел, чтобы при этом дополнительные затраты линейно зависели от количества чисел. Поставим в соответствие числу $x_i$ точку $(x_i, x_i^2)$ и присвоим ей номер $i$ (рис. 3.1). Все эти точки лежат на параболе $y = x^2$. Выпуклая оболочка этого множества точек, представленная в стандартном виде, будет состоять из списка точек множества, упорядоченного по значению абсциссы. Один просмотр этого списка позволяет прочитать в нужном порядке значения $x_i$ [1]).

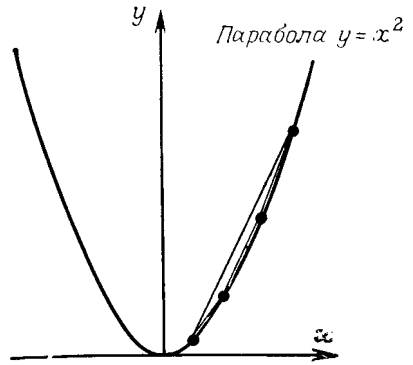



Рис. 3.1. Иллюстрация к доказательству теоремы 3.2.

Так как для процедуры сведения используются только арифметические операции, то теорема 3.2 остается справедливой во многих вычислительных моделях. А именно в моделях, допускающих умножение, и в которых время, необходимое для сортировки, равно $\Omega(N \log N)$. Как указывалось ранее, теорема 3.2 применима в пространствах любой размерности, большей 1 [2]).

Если взглянуть на задачу ВО2 о крайних точках, то сразу же становится ясно, что здесь не возникает никаких элементарных соображений, подобных приведенным выше. В действительности эта задача на протяжении некоторого времени оставалась нерешенной. Первым прорывом явилась работа А. Яо [Yao (1981)], а окончательный ответ был получен в результате соединения мощного метода алгебраических деревьев решений Бен-Ора (см. разд. 1.4) с недавним результатом Стили и Яо [Steele, Yao (1982)].

Следуя привычному ходу мысли, рассмотрим задачу распознавания, связанную с ВО2, которая формулируется следующим образом.

**Задача ВО3** (ПРОВЕРКА КРАЙНОСТИ ТОЧЕК ПЛОСКОСТИ). На плоскости заданы $N$ точек. Определить, являются ли они вершинами своей собственной выпуклой оболочки?

[1]) Шеймос первоначально доказал эту теорему, отображая числа $x_i$ на единичную окружность. Отображение на параболу, предложенное С. Эйзенстатом, является более предпочтительным, так как оно требует лишь операций рациональной арифметики.

[2]) Выпуклой оболочкой множества точек в пространстве размерности 1 является наименьший содержащий их интервал. Этот интервал может быть найден за линейное время.

Прежде чем обсуждать чрезвычайно важный результат Стили — Яо и Бен-Ора, давайте посмотрим, почему модель линейного дерева решений не подходит для нашей задачи. Коротко можно сказать, что не существует алгоритма построения выпуклой оболочки, использующего исключительно линейные проверки, поэтому модель линейного дерева решений не применима. Типичная элементарная операция имеет вид: даны три точки $p$, $p'$ и $p''$ и требуется определить, находится ли точка $p$ слева или справа или на ориентированном отрезке $\overrightarrow{p'p''}$? Полином, реализующий соответствующую проверку, задается определителем

$$\Delta = \begin{vmatrix} x & y & 1 \\ x' & y' & 1 \\ x'' & y'' & 1 \end{vmatrix}, \tag{3.4}$$

где $p = (x, y)$, $p' = (x', y')$ и $p'' = (x'', y'')$. Этот определитель, как отмечалось в разд. 2.2.1, дает удвоенную площадь треугольника $(pp'p'')$ со знаком. Очевидно, что полином (3.4) является *квадратным*.

Это — огорчающее обстоятельство, так как ранее Эйвис [Avis (1979)] предложил простое и изящное доказательство, основанное на модели линейного дерева решений. В этом доказательстве ключевым образом использовался тот факт, что линейная проверка $f(x_1, \ldots, x_m)$: 0 определяет гиперплоскость в $m$-мерном евклидовом пространстве $E^m$ и каждый путь от корня к листу дерева определяет пересечение выпуклых множеств (полупространства соответствуют случаям «$>$» или «$<$», гиперплоскости соответствуют случаю («$=$»), которое само является выпуклым множеством в $E^m$ (область решений, связанная с этим листом). Однако это свойство не сохраняется, когда проверочный полином имеет более высокую степень.

Искусное, но чрезвычайно сложное доказательство, представленное Яо, ограничивалось моделью *квадратичных* деревьев решений. Хотя существующие алгоритмы можно охватить, ограничиваясь квадратичными деревьями вычислений, непроизвольно возникает следующий вопрос: если допускаются полиномы степени выше второй, то можно ли по-прежнему доказывать аналогичные результаты? Или, если этого нельзя сделать, можно ли, используя такие, возможно, более мощные проверки, разработать более быстрые алгоритмы? Яо высказался против положительного ответа. Однако совсем недавно этот вопрос был окончательно решен с помощью следующего доказательства.

Предположим, дан алгоритм, представленный алгебраическим деревом решений фиксированного порядка $d \geqslant 2$, претендующий на решение задачи ВО3 для множества из $2N$ точек на плоскости, т. е. он воспринимает на входе вектор из $4N$ компонент (по две координаты для каждой точки) и определяет, содержит ли их выпуклая оболочка $2N$ крайних точек. Отметим, что входной вектор, имеющий $4N$ компонент, можно вполне корректно представлять как точку в пространстве $E^{4N}$. Рассматриваемая нами задача на распознавание характеризуется «множеством истинности» $W \subset E^{4N}$ (см. разд. 1.4). Для удобства мы повторяем здесь результат, полученный Бен-Ором.



**Теорема 1.2.** *Пусть $W$ — множество в $E^m$, а $T$ — алгебраическое дерево решений фиксированного порядка $d$, решающее задачу о принадлежности $W$. Если $h^*$ — глубина дерева $T$, а $\#(W)$ — число разделимых связных компонент $W$, то*

$$h^* = \Omega(\log \#(W) - m).$$

Для того чтобы воспользоваться этим результатом, необходимо получить нижнюю оценку для $\#(W)$. Это может быть сделано следующим образом.

Пусть $(p_0, p_1, \ldots, p_{2N-1})$ — последовательность вершин выпуклого многоугольника, перечисленных в порядке обхода по часовой стрелке (рис. 3.2). Исходные данные для алгоритма представлены в виде цепочки $\mathbf{z} = (z_0, z_1, \ldots, z_{4N-1})$ действительных

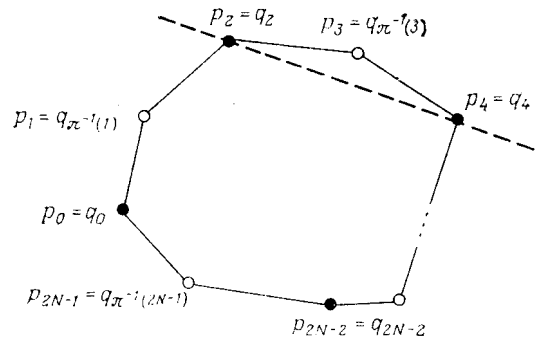



Рис. 3.2. Иллюстрация к доказательству теоремы 3.3.

чисел, получаемой следующим образом: на плоскости заданы $2N$ точек $q_0, q_1, \ldots, q_{2N-1}$ с координатами $q_i \equiv (x_i, y_i)$, полагаем $z_{2i} = x_i$ и $z_{2i+1} = y_i$ для $i = 0, 1, \ldots, 2N - 1$. Для фиксированной последовательности $(p_0, p_1, \ldots, p_{2N-1})$ можно построить $N!$ различных экземпляров входных цепочек для данного варианта задачи, выбирая произвольную перестановку $\pi$ целых чисел $\{0, 1, \ldots, N - 1\}$ и полагая

$$q_{2s} = p_{2s}, \quad q_{2s+1} = p_{\pi(2s+1)} \quad (s = 0, \ldots, N - 1). \tag{3.5}$$

Отметим, что все возможные для данной задачи входные последовательности имеют одинаковые подпоследовательности точек

с четными индексами (показанные на рис. 3.2 черными кружками). Для заданной перестановки $\pi$ обозначим через $\mathbf{z}(\pi)$ (последовательность координат точек $(q_0, \ldots, q_{2N-1})$).

Теперь для каждой перестановки $\pi$ построим массив $A(\pi)$ с $N^2$ элементами. Сделаем это следующим образом. Обозначив через $\Delta(u, v, w)$ площадь треугольника $(u, v, w)$ со знаком и через $A(\pi)[j]$ $j$-й элемент $A(\pi)$, имеем

$$A(\pi)[Ns+r] = \Delta(q_{2s},\ q_{(2s+2) \bmod 2N},\ q_{2r+1}),$$
$$0 \leqslant s < N, \quad 0 \leqslant r < N. \quad (3.6)$$

Из (3.5) и (3.6) и рис. 3.2 легко понять, что для данного $s$ имеется всего одно значение $r$, для которого $A(\pi)[Ns+r] < 0$ (это имеет место при $2s+1 = \pi^{-1}(2r+1)$). Предположим теперь, что $\pi_1$ и $\pi_2$ — две *различные* перестановки чисел $\{0, 1, \ldots, N-1\}$, и рассмотрим массивы $A(\pi_1)$ и $A(\pi_2)$. Мы утверждаем, что существует по крайней мере одно целое число $j$ такое, что $A(\pi_1)[j]$ и $A(\pi_2)[j]$ имеют противоположные знаки. Действительно, рассмотрим $N$ элементов массива $A(\pi_1)$, являющихся отрицательными, т. е. элементы $A(\pi_1)[Ni+r_i]$, $0 \leqslant i < N$. Если $A(\pi_1)[Ni+r_i] \cdot A(\pi_2)[Ni+r_i] > 0$, то обе точки $q_{\pi_1^{-1}(Ni+r_i)}$ и $q_{\pi_2^{-1}(Ni+r_i)}$ лежат слева от направленного отрезка $\overrightarrow{p_{2i}p_{2i+2}}$; но мы знаем, что имеется только одна точка в множестве $\{p_0, \ldots, p_{2N-1}\}$, обладающая этим свойством, и этой точкой является $p_{2i+1}$. Таким образом, $q_{\pi_1^{-1}(Ni+r_i)} = q_{\pi_2^{-1}(Ni+r_i)} = p_{2i+1}$. Если это имеет место для всех значений $i$, то эти две перестановки совпадают, что противоречит предположению.

Поэтому существует такое $j$, что $A(\pi_1)[j]$ и $A(\pi_2)[j]$ имеют противоположные знаки. Заметим теперь, что $\mathbf{z}(\pi_1)$ и $\mathbf{z}(\pi_2)$ являются различными точками в $E^{4N}$, и рассмотрим кривую $\rho$ в $E^{4N}$, соединяющую эти точки. Точка $\mathbf{z}$, двигающаяся по $\rho$ из $\mathbf{z}(\pi_1)$ в $\mathbf{z}(\pi_2)$, описывает непрерывную деформацию на плоскости конфигурации, соответствующей $\pi_1$, в конфигурацию, соответствующую $\pi_2$, при этом $A(\pi_1)$ преобразуется в $A(\pi_2)$. Так как $A(\pi_1)[j] \cdot A(\pi_2)[j] < 0$, то, согласно теореме о промежуточном значении, на кривой $\rho$ существует точка $\mathbf{z}^*$ такая, что в соответствующей конфигурации три точки становятся коллинеарными (т. е. треугольник с вершинами в этих точках имеет нулевую площадь), и поэтому не более чем $(2N-1)$ точек являются крайними. Это доказывает, что при перемещении из $\mathbf{z}(\pi_1)$ в $\mathbf{z}(\pi_2)$ мы должны выходить из $W$, так что $\mathbf{z}(\pi_1)$ и $\mathbf{z}(\pi_2)$ принадлежат различным компонентам $W$. Так как это справедливо при произвольном выборе $\pi_1$ и $\pi_2$, то отсюда следует, что $\#(W) \geqslant N!$.

С учетом теоремы Бен-Ора имеем:



**Теорема 3.3.** *В рамках модели алгебраических деревьев вычислений фиксированного порядка для определения крайних точек множества из $N$ точек на плоскости требуется $\Omega(N \log N)$ операций.*

*Доказательство.* В нашем случае $m = 4N$ и $\#(W) \geqslant N!$. Согласно теореме Бен-Ора, необходимо $(\log \#(W) - m) \geqslant \geqslant (\log N! - 4N) = \Omega(N \log N)$ операций.

На этом заканчивается наше обсуждение сложности задачи. Прежде чем закончить эту тему, заметим, что двумерные варианты как задачи об упорядоченной выпуклой оболочке BO1, так и задачи о крайних точках BO2 имеют тесную связь с задачей сортировки. Действительно, задача BO1 явным образом связана с ней посредством прямого сведéния, в то время как связь BO2 осуществляется через мощность симметрической группы. Эта связь является довольно глубокой, и в дальнейшем в этой главе мы будем часто обращаться к ней.

Теперь мы переходим к описанию алгоритмов построения выпуклой оболочки.

## 3.3. Алгоритм построения выпуклой оболочки на плоскости

### 3.3.1. Предварительная разработка алгоритма построения выпуклой оболочки

Теперь мы должны, опираясь на неконструктивное по своей сути определение 3.4, получить математические результаты, ведущие к эффективным алгоритмам. В работах такого рода принято представлять последовательность хорошо обоснованных и «отполированных» результатов, но сделать так здесь означало бы лишь совсем немного раскрыть процесс разработки алгоритмов, являющийся стержнем вычислительной геометрии. Поэтому мы начинаем с обсуждения некоторых малопродуктивных подходов.

**Определение 3.6.** Точка $p$ выпуклого множества $S$ называется *крайней*, если не существует пары точек $a, b \in S$ таких, что $p$ лежит на открытом отрезке $\overline{ab}$.

Множество $E$ крайних точек $S$ представляет собой наименьшее подмножество $S$, обладающее тем свойством, что $\mathrm{conv}(E) = \mathrm{conv}(S)$, и $E$ в точности совпадает с множеством вершин $\mathrm{conv}(S)$.

Отсюда следует, что для того, чтобы найти выпуклую оболочку конечного множества точек, требуется выполнить следующие два шага:

1. Определить крайние точки. (Это задача ВО2 в двумерном случае.)
2. Упорядочить эти точки так, чтобы они образовывали выпуклый многоугольник.

Необходима теорема, которая позволит нам проверять, является ли некоторая точка крайней.

**Теорема 3.4.** *Точка $p$ не является крайней плоского выпуклого множества $S$ только тогда, когда она лежит в некотором треугольнике, вершинами которого являются точки из $S$, но сама она не является вершиной этого треугольника* [1]) (рис. 3.3).

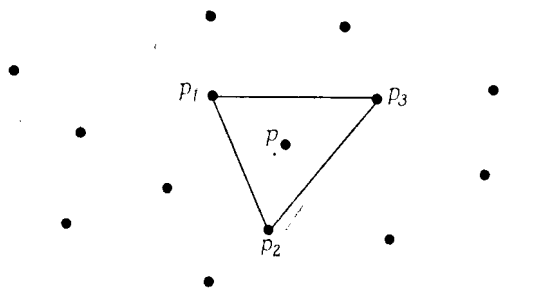



Рис. 3.3. Точка $p$ не является крайней, так как она находится внутри треугольника ($p_1p_2p_3$).

Эта теорема дает идею для алгоритма удаления точек, не являющихся крайними. Имеется $O(N^3)$ треугольников, определяемых $N$ точками множества $S$. Проверка принадлежности точки заданному треугольнику может быть выполнена за некоторое постоянное число операций, так что за время $O(N^3)$ можно определить, является ли конкретная точка крайней. Повторение этой процедуры для всех $N$ точек множества $S$ потребует времени $O(N^4)$. Хотя наш алгоритм является чрезвычайно неэффективным, он очень прост в идейном плане и показывает, что крайние точки могут быть определены за конечное число шагов.

Мы затратили время $O(N^4)$ только на определение крайних точек, которые должны быть как-то упорядочены, чтобы образовать выпуклую оболочку. Смысл этого порядка раскрывается следующими теоремами.

[1]) Это непосредственно следует из доказательств теорем 10 и 11 в книге [Hadwiger, Debrunner (1964)]. Теорему 3.4 можно обобщить на случай $d$-мерного пространства, заменив «треугольник» на «симплекс с $d+1$ вершинами». Заметим, что треугольник может вырождаться в отрезок, когда его вершины становятся коллинеарными.



**Теорема 3.5.** *Луч, выходящий из внутренней точки ограниченной выпуклой фигуры $F$, пересекает границу $F$ в точности в одной точке* [1]).

**Теорема 3.6.** *Последовательные вершины выпуклого многоугольника располагаются в порядке, соответствующем изменению угла относительно любой внутренней точки.*

Представьте луч, выходящий из некоторой внутренней точки $q$ многоугольника $P$ и заметающий вершины многоугольника $P$ в порядке движения против часовой стрелки, начиная из положения, совпадающего по направлению с положительным направлением оси $x$ системы координат. По мере движения от вершины к вершине полярный угол [2]) луча монотонно увеличивается. Именно это имелось в виду, когда говорилось о том, что вершины многоугольника $P$ «упорядочены» («располагаются в порядке») (рис. 3.4).

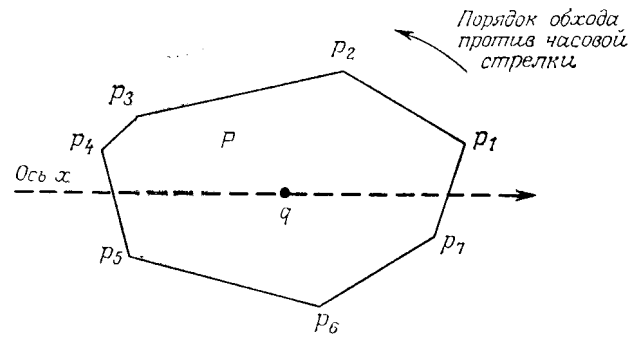



Рис. 3.4. Вершины многоугольника $P$ упорядочены относительно точки $q$.

Если даны крайние точки некоторого множества, то его выпуклую оболочку можно найти, выбрав точку $q$, про которую известно, что она является внутренней точкой оболочки, и упорядочив затем крайние точки в соответствии с полярным углом относительно $q$. В качестве точки $q$ можно взять центроид множества крайних точек: хорошо известно, что центроид множества точек является внутренней точкой выпуклой оболочки [3]).

[1]) Эта теорема является следствием теоремы 1.10 из [Valentine (1964)] и теоремы Жордана.

[2]) Полярный угол измеряется обычным способом в направлении против часовой стрелки от положительного направления оси $x$.

[3]) При условии что внутренность не пуста. В книге [Benson (1966), упр. 25.15] «внутренность» понимается в смысле относительной (индуцированной) топологии. В $E^3$ выпуклая оболочка двух различных точек представляет отрезок, внутренность которого является пустой в метрической топологии $E^3$, но не пустой в относительной топологии.

Центроид множества из $N$ точек в $k$-мерном пространстве может быть тривиально вычислен за $O(Nk)$ арифметических операций.

Грэхем предложил иной метод нахождения внутренней точки, заметив, что вполне достаточно взять центроид любых трех неколлинеарных точек [Graham (1972)]. Он начинает с двух произвольных точек и по очереди исследует оставшиеся $N-2$ точек, ища среди них одну, которая не лежит на прямой, определяемой первыми двумя точками. В худшем случае для этого требуется время $O(N)$, но почти всегда процесс заканчивается через некоторое фиксированное время: если точки выбираются из абсолютно непрерывного распределения, то с вероятностью единица первые три точки являются неколлинеарными [Efron (1965)].

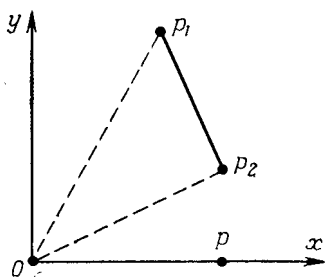



Рис. 3.5. Сравнение полярных углов путем вычисления ориентированной площади треугольника $(0, p_2, p_1)$.

Найдя крайние точки множества $S$, можно за время $O(N)$ найти некоторую точку $q$, являющуюся внутренней точкой оболочки. Остается только отсортировать (упорядочить) крайние точки в соответствии со значением полярного угла, используя точку $q$ в качестве начала системы координат. Это можно сделать, преобразуя точки в полярную систему координат за время $O(N)$, а затем отсортировав их за время $O(N \log N)$, но на самом деле не требуется явного вычисления полярных координат. Так как сортировка может быть выполнена путем попарных сравнений, то необходимо лишь определять, какой из двух заданных углов больше; нам не требуются их численные значения. Давайте рассмотрим этот вопрос более подробно, так как он иллюстрирует простой, но важный геометрический «трюк», оказывающийся полезным во многих приложениях. Пусть на плоскости заданы две точки $p_1$ и $p_2$. Какая из них имеет больший полярный угол? На этот вопрос можно без труда ответить, если рассмотреть ориентированную (с учетом знака) площадь некоторого треугольника. Действительно, если даны две точки $p_1$ и $p_2$, то $p_2$ образует с осью $x$ строго меньший полярный угол, чем $p_1$, тогда и только тогда, когда треугольник $(O, p_2, p_1)$ имеет строго положительную площадь (рис. 3.5).

На этом заканчивается подробное обсуждение нашего первого алгоритма построения выпуклой оболочки. Было показано, что эта задача может быть решена за время $O(N^4)$ с использованием только арифметических операций и сравнений.



### 3.3.2. Метод обхода Грэхема

Алгоритм со временем выполнения $O(N^4)$ не позволит обрабатывать очень большие наборы данных. Если временные характеристики должны быть улучшены, то это можно сделать, либо устранив избыточные вычисления в имеющемся алгоритме, либо выбрав иной теоретический подход. В этом разделе мы исследуем наш алгоритм с точки зрения наличия в нем ненужных вычислений.

Так ли необходимо проверять все треугольники, определяемые множеством из $N$ точек, чтобы узнать, лежит ли некоторая точка в каком-либо из них? Если нет, то имеется некоторая надежда, что крайние точки можно найти за время, меньшее чем $O(N^4)$. Грэхем в одной из первых работ, специально посвященных вопросу разработки эффективных геометрических алгоритмов [Graham (1972)], показал, что, выполнив предварительно сортировку точек, крайние точки можно найти за линейное время. Использованный им метод стал очень мощным средством в области вычислительной геометрии.

Предположим, что внутренняя точка уже найдена, а координаты других точек тривиальным образом преобразованы так, что найденная внутренняя точка оказалась в начале координат. Упорядочим лексикографически $N$ точек в соответствии со значениями полярного угла и расстояния от начала координат. При выполнении сортировки, естественно, не нужно вычислять действительное расстояние между двумя точками, так как требуется лишь сравнить две величины. Можно работать с *квадратом* расстояния, избегая тем самым операции извлечения квадратного корня, но рассматриваемый случай еще проще. Сравнение расстояний необходимо выполнять лишь в случае, если две точки имеют один и тот же полярный угол. Но тогда они лежат на одной прямой с началом координат, и сравнение в этом случае тривиально.

Представив упорядоченные точки в виде дважды связанного кольцевого списка, получаем ситуацию, представленную на рис. 3.6. Обратите внимание: если точка не является вершиной выпуклой оболочки, то она является внутренней точкой для некоторого треугольника $(Opq)$, где $p$ и $q$ — последовательные вершины выпуклой оболочки. Суть алгоритма Грэхема состоит в однократном просмотре упорядоченной последовательности точек, в процессе которого удаляются внутренние точки. Оставшиеся точки являются вершинами выпуклой оболочки, представленными в требуемом порядке.

Просмотр начинается с точки, помеченной как НАЧАЛО, в качестве которой можно взять самую правую с наименьшей ординатой точку из данного множества, заведомо являющуюся

вершиной выпуклой оболочки. Тройки последовательных точек многократно проверяются в порядке обхода против часовой стрелки с целью определить, образуют или нет они угол, больший или равный $\pi$. Если внутренний угол $p_1p_2p_3$ больше или равен $\pi$, то говорят, что $p_1p_2p_3$ образуют «правый поворот», иначе они образуют «левый поворот». Это можно легко определить, воспользовавшись формулой (3.4). Из выпуклости многоугольника непосредственно следует, что при его обходе будут делаться только левые повороты. Если $p_1p_2p_3$ образуют правый

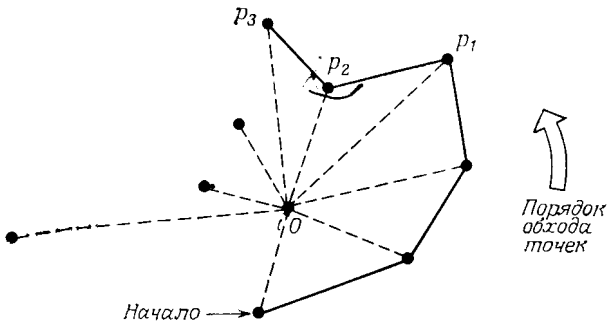



Рис. 3.6. Начало обхода точек в методе Грэхема. Вершина $p_2$ удаляется, если угол $p_1p_2p_3$ оказывается вогнутым.

поворот, то $p_2$ не может быть крайней точкой, так как она является внутренней для треугольника $(Op_1p_3)$. В зависимости от результата проверки угла, образуемого текущей тройкой точек, возможны два варианта продолжения просмотра:

1. $p_1p_2p_3$ образуют правый поворот. Удалить вершину $p_2$ и проверить тройку $p_0p_1p_3$.

2. $p_1p_2p_3$ образуют левый поворот. Продолжить просмотр, перейдя к проверке тройки $p_2p_3p_4$.

Просмотр завершается, когда, обойдя все вершины, вновь приходим в вершину НАЧАЛО. Заметим, что вершина НАЧАЛО никогда не удаляется, так как она является крайней точкой и поэтому при отходе назад после удаления точек, мы не сможем уйти дальше точки, предшествующей точке НАЧАЛО. Простой анализ показывает, что такой просмотр выполняется лишь за линейное время. Проверка угла может быть выполнена за фиксированное (постоянное) число операций. После каждой проверки происходит либо продвижение на одну точку (случай 2), либо удаляется точка (случай 1). Так как множество содержит лишь $N$ точек, то продвижение вперед не может происходить более $N$ раз, как не может быть удалено и более $N$ точек. Рассмотренный метод обхода границы многоугольника является настолько полезным, что в дальнейшем мы будем называть его



*методом обхода Грэхема.* Ниже дано более точное описание этого алгоритма. В алгоритме $S$ — это исходное множество из $N$ точек на плоскости.

**procedure** ОБОЛОЧКА-ГРЭХЕМА($S$)

1. Найти внутреннюю точку $q$.
2. Используя $q$ как начало координат, упорядочить точки множества $S$ лексикографически в соответствии с полярным углом и расстоянием от $q$. Организовать точки множества в виде кольцевого дважды связанного списка со ссылками СЛЕД и ПРЕД и указателем НАЧАЛО на первую вершину. Значение true логической переменной $f$ указывает на то, что вершина НАЧАЛО оказалась достигнутой при прямом продвижении по оболочке, а не в результате возврата.
3. (Обход)

```
begin v := НАЧАЛО; w := ПРЕД[v]; f := false;
  while(СЛЕД[v] ≠ НАЧАЛО or f = false) do
    begin if СЛЕД[v] = w then f := true;
          if (три точки v, СЛЕД[v] СЛЕД[СЛЕД[v]]
              образуют левый поворот) then v := СЛЕД[v]
          else begin УДАЛИТЬ СЛЕД[v];
                     v := ПРЕД[v]
               end
    end
end.
```

По окончании выполнения алгоритма список содержит упорядоченные нужным образом вершины оболочки.

**Теорема 3.7.** *Выпуклая оболочка $N$ точек на плоскости может быть найдена за время $O(N \log N)$ при памяти $O(N)$ с использованием только арифметических операций и сравнений.*

*Доказательство.* Из предыдущего обсуждения алгоритма Грэхема видно, что в нем используются лишь арифметические операции и сравнения. Шаги 1 и 3 требуют линейного времени, тогда как шаг 2, являющийся определяющим по времени работы, выполняется за время $O(N \log N)$. Для представления связного списка точек достаточно $O(N)$ памяти.

Если вспомнить нижнюю оценку, обсуждавшуюся в разд. 3.2, то видно, что этот простой и изящный алгоритм имеет оптимальное время выполнения. Однако имеется одно обстоятельство, которое может вызвать некоторое беспокойство у читателя, — это использование полярных координат. В самом деле, это предполагает выполнение преобразования координат, что может быть затруднительным в системах с ограниченным набором прими-

тивных операций. Как это обычно случается, ряд исследователей обратили внимание на это обстоятельство и предложили решения, позволяющие избежать его. Мы коротко опишем метод, предложенный Эндрью [Andrew (1979)]. Другой подход, также заслуживающий внимания, был предложен Эклом и Туссэном [Akl, Toussaint (1978)].

Пусть на плоскости задано множество из $N$ точек. Определим сначала его левую и правую крайние точки $l$ и $r$ (рис. 3.7) и построим прямую, проходящую через эти точки. Оставшиеся точки разбиваются на два подмножества (нижнее и верхнее)

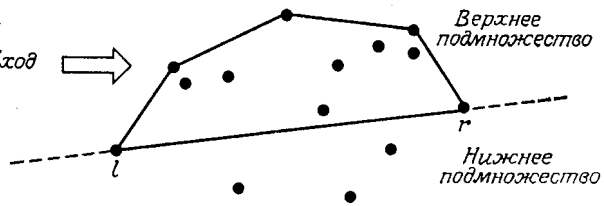



Рис. 3.7. Левая и правая крайние точки определяют разбиение множества на два подмножества.

в зависимости от того, по какую сторону от прямой они располагаются — ниже или выше прямой. Нижнее подмножество порождает ломаную (*нижнюю оболочку*, или H-*оболочку*), монотонную относительно оси $x$. Точно так же верхнее подмножество порождает аналогичную ломаную (*верхнюю оболочку*, или B-*оболочку*), а соединение этих двух ломаных дает выпуклую оболочку исходного множества.

Рассмотрим построение верхней оболочки. Точки упорядочиваются в соответствии с возрастанием абсциссы, и к полученной последовательности применяется метод обхода Грэхема. При таком подходе отпадает необходимость в тригонометрических операциях. Заметим, однако, что предлагаемый подход есть не что иное, как частный случай применения исходного метода Грэхема, когда точка $q$, совпадающая с началом координат, выбирается бесконечно удаленной в отрицательном направлении ($-\infty$) по оси $y$, так что в этом случае упорядоченность по абсциссе совпадает с упорядоченностью по полярному углу.

Несмотря на то что, как было показано, алгоритм Грэхема является оптимальным, по-прежнему имеется много причин для продолжения исследования задачи о выпуклой оболочке.

1. Рассмотренный алгоритм является оптимальным в *худшем случае*, но мы, однако, не изучили его поведение в среднем.

2. Так как алгоритм основывается на теореме 3.6, применимой только в случае плоскости, алгоритм не имеет обобщения на случай пространств более высокой размерности.



3. Алгоритм не является открытым алгоритмом [1]), так как все точки множества должны быть известны до начала работы алгоритма.

4. При возможности параллельной обработки более предпочтительным является рекурсивный алгоритм, допускающий разбиение исходной задачи и данных на меньшие подзадачи.

Прежде чем закончить этот раздел, отметим, что метод Грэхема явным образом использует сортировку. Вряд ли можно найти более прямую связь этой задачи с задачей сортировки. Аналогично тому, как исследование сортировки показало, что не существует одного алгоритма, лучшего во всех случаях, мы увидим, что то же самое справедливо и для задачи построения выпуклой оболочки. Давайте продолжим исследование комбинаторной геометрии в поисках идей, способных привести к иным алгоритмам построения выпуклой оболочки.

### 3.3.3. Обход методом Джарвиса

Многоугольник с одинаковым успехом можно задать упорядоченным множеством как его ребер, так и его вершин. В задаче о выпуклой оболочке мы до сих пор обращали внимание главным образом на изолированные крайние точки. А что если вместо этого попытаться определить ребра выпуклой оболочки, приведет ли такой подход к созданию практически пригодного алгоритма? Если задано множество точек, то довольно трудно быстро определить, является или нет некоторая точка крайней. Однако если даны две точки, то непосредственно можно проверить, является или нет соединяющий их отрезок ребром выпуклой оболочки.

**Теорема 3.8.** *Отрезок $l$, определяемый двумя точками, является ребром выпуклой оболочки тогда и только тогда, когда все другие точки заданного множества лежат на $l$ или с одной стороны от него* [Stoer, Witzgall (1970), теорема 2.4.7]. (См. рис. 3.8.)

$N$ точек определяют $\binom{N}{2} = O(N^2)$ прямых. Для каждой из этих прямых можно, используя формулу (3.4), определить за линейное время положение остальных $N-2$ точек относительно этой прямой и тем самым проверить, удовлетворяет или нет прямая условиям теоремы 3.8. Таким образом, за время $O(N^3)$ можно найти все пары точек, определяющих ребра выпуклой оболочки. Не надо затем большого труда, чтобы расположить эти точки в виде списка последовательных вершин оболочки.

[1]) См. определение открытого алгоритма в разд. 3.3.6. В оригинале «on-line». — *Прим. перев.*

Джарвис заметил, что этот алгоритм можно улучшить, если учесть следующий факт. Если установлено, что отрезок $\overline{pq}$ является ребром оболочки, то должно существовать другое ребро с концом в точке $q$ [Jarvis (1973)]. В его работе показано, как использовать этот факт, чтобы уменьшить требуемое время до $O(N^2)$. Кроме того, в ней содержится ряд других идей, заслуживающих подробного обсуждения. Здесь будет рассмотрен вариант, включающий изменения, предложенные Эклом [Akl (1979)] и устраняющие небольшие неточности.

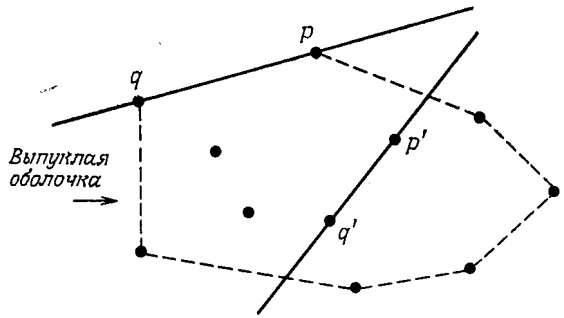



Рис. 3.8. Ребро выпуклой оболочки не может разделять множество точек на части. $\overline{pq}$ является ребром выпуклой оболочки, так как все точки множества располагаются по одну сторону от него; $\overline{p'q'}$ не является ребром выпуклой оболочки, так как по обе стороны от него имеются точки.

Предположим, что, как и в разд. 3.3.2, найдена наименьшая в лексикографическом порядке точка $p_1$ заданного множества точек. Эта точка заведомо является вершиной оболочки, и теперь хотелось бы найти следующую за ней вершину $p_2$ выпуклой оболочки. Точка $p_2$ — это точка, имеющая наименьший положительный полярный угол относительно точки $p_1$ как начала координат. Аналогично следующая точка $p_3$ имеет наименьший полярный угол относительно точки $p_2$ как начала координат, и каждая последующая точка выпуклой оболочки может быть найдена за линейное время. Алгоритм Джарвиса обходит кругом выпуклую оболочку (отсюда и соответствующее название — *обход Джарвиса*), порождая в нужном порядке последовательность крайних точек, по одной точке на каждом шаге (рис. 3.9). Таким образом строится часть выпуклой оболочки (ломаная линия) от наименьшей в лексикографическом порядке точки ($p_1$ на рис. 3.9) до наибольшей в лексикографическом порядке точки ($p_4$ на том же рисунке). Построение выпуклой оболочки завершается нахождением другой ломаной, идущей из наибольшей в лексикографическом порядке точки в наименьшую в лексикографическом порядке точку. Ввиду симметричности этих



двух этапов необходимо изменить на противоположные направления осей координат и иметь дело теперь с полярными углами, наименьшими относительно *отрицательного* направления оси $x$.

Как уже было показано в разд. 3.3.1, наименьший угол может быть найден с использованием лишь арифметических операций и сравнений, не прибегая к явному вычислению значений полярных углов.

Так как все $N$ точек множества могут лежать на его выпуклой оболочке (быть ее вершинами), а алгоритм Джарвиса затрачивает на нахождение каждой точки оболочки линейное

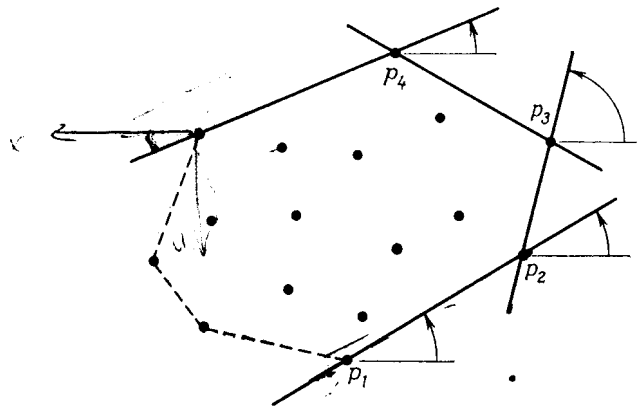



Рис. 3.9. Построение выпуклой оболочки методом Джарвиса. Алгоритм Джарвиса находит последовательные вершины оболочки путем многократного вычисления угла поворота. Каждая новая вершина определяется за время $O(N)$.

время, то время выполнения алгоритма в худшем случае равно $O(N^2)$, что хуже, чем у алгоритма Грэхема. Если в действительности число вершин выпуклой оболочки равно $h$, то время выполнения алгоритма Джарвиса будет $O(hN)$, и он очень эффективен, когда заранее известно, что значение $h$ мало. Например, если оболочка заданного множества является многоугольником с произвольным постоянным числом сторон, то ее можно найти за линейное относительно числа точек время. Этот факт чрезвычайно важен в свете анализа сложности алгоритмов построения выпуклой оболочки в среднем, который будет представлен в следующей главе.

Другое уместное здесь замечание состоит в том, что идея поиска последовательных вершин оболочки с помощью многократного применения процедуры определения минимального угла интуитивно ассоциируется с завертыванием двумерного предмета. В действительности метод Джарвиса можно рассматривать как

двумерный вариант подхода, основанного на идее «заворачивания подарка» и предложенного Чандом и Капуром [Chand, Kapur (1970)] еще до появления работы Джарвиса. Метод «заворачивания подарка» применим также в случае пространств размерности большей двух, и в общих чертах он будет рассмотрен в разд. 3.4.1.

### 3.3.4. Быстрые методы построения выпуклой оболочки

Задача сортировки является источником вдохновения при выработке идей решения задачи построения выпуклой оболочки. Так, например, в ряде методов, независимо предложенных почти в одно и то же время [Eddy (1977); Bykat (1978); Green, Silverman (1979); Floyd (частное сообщение, (1976)], легко угадывается основная идея (с небольшими вариациями) алгоритма быстрой сортировки БЫСТРСОРТ [1]). Ввиду их близкого сходства с алгоритмом БЫСТРСОРТ мы выбрали для них общее название — быстрые методы построения выпуклой оболочки, или БЫСТРОБОЛ-методы.

Чтобы лучше понять эту аналогию, коротко напомним механизм алгоритма БЫСТРСОРТ [Hoare (1962)]. Имеется массив из $N$ чисел, и необходимо разбить его на левый и правый подмассивы так, чтобы каждый элемент первого подмассива не превосходил каждый из элементов второго подмассива. Это делается с помощью двух указателей на элементы массива, установленных в начале соответственно на два крайних элемента и перемещаемых навстречу друг другу на один элемент за один шаг обработки. Каждый раз, когда элементы, на которые указывают указатели, не удовлетворяют требуемому порядку, производится перестановка этих элементов. Указатели двигаются попеременно — один двигается, а другой остается на месте. После каждой выполненной перестановки указатели меняются ролями. При достижении указателями одного и того же элемента массив оказывается разбитым на два подмассива, и затем к каждому из них по отдельности применяется та же самая процедура. Как хорошо известно, такой подход является очень эффективным (обработка завершается за время $O(N \log N)$), если каждый массив разбивается на приблизительно равные части.

Соответствующий БЫСТРОБОЛ-метод разбивает множество $S$ из $N$ точек на два подмножества, каждое из которых будет содержать одну из двух ломаных, соединение которых дает многоугольник выпуклой оболочки. Начальное разбиение множества определяется прямой, проходящей через две точки $l$ и $r$, имеющие соответственно наименьшую и наибольшую абсциссы (рис. 3.10), точно так же, как в варианте алгоритма Грэхема, предложенном Эндрью (см. разд. 3.3.2). Обозначим через $S^{(1)}$ подмножество точек, расположенных выше или на прямой, проходящей через $l$ и $r$, а через $S^{(2)}$ симметричным образом определяемое подмножество точек, расположенных ниже или на той же самой прямой. (Более точно, $\{S^{(1)}, S^{(2)}\}$ не является разбиением множества $S$, так как $S^{(1)} \cap S^{(2)} \supseteq \{l, r\}$; эта несущественная деталь, обеспечивающая удобную симметрию, никоим образом не должна вводить в заблуждение читателя.)

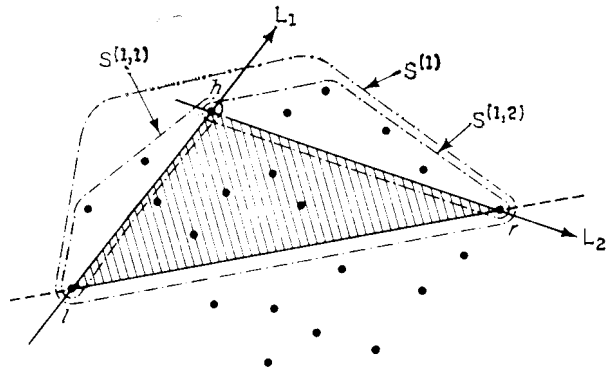



Рис. 3.10. Точки $l$, $r$ и $h$ определяют разбиение множества $S^{(1)}$. Все точки, попавшие в заштрихованный треугольник, не рассматриваются.

На каждом последующем шаге обработка множеств, подобных $S^{(1)}$ и $S^{(2)}$, выполняется следующим образом (для конкретности мы рассмотрим множество $S^{(1)}$ на рис. 3.10). Определим точку $h$, для которой треугольник $(hlr)$ имеет максимальную площадь среди всех треугольников $\{(plr): p \in S^{(1)}\}$, а если таких точек имеется более одной, то выбираем ту из них, у которой угол $\angle(hlr)$ больше. Заметим, что точка $h$ гарантированно принадлежит выпуклой оболочке. Действительно, если провести через точку $h$ прямую, параллельную отрезку $\overline{lr}$, то выше этой прямой не окажется ни одной точки множества $S$. Возможно, кроме точки $h$ на этой прямой окажутся другие точки из множества $S$, но, согласно сделанному нами выбору, $h$ является самой левой из них. Так что точка $h$ не может быть представлена в виде выпуклой комбинации двух других точек множества $S$.

Затем строятся две прямые: одна $L_1$, направленная из $l$ в $h$, другая $L_2$ — из $h$ в $r$. Для каждой точки множества $S^{(1)}$ определяется ее положение относительно этих прямых. Ясно, что ни одна из точек не находится одновременно слева как от $L_1$, так и от $L_2$, кроме того, все точки, расположенные справа от обеих

[1]) [Aho, Hopcrott, Ullman (1974)] — с. 111 русск. изд. и [Knuth (1973)] — с. 140 русск. изд. — *Прим. перев.*

прямых, являются внутренними точками треугольника $(lrh)$ и поэтому могут быть удалены из дальнейшей обработки. Точки, расположенные слева от $L_1$ или на ней (и расположенные справа от $L_2$), образуют множество $S^{(1,1)}$; аналогично образуется множество $S^{(1,2)}$. Вновь образованные множества $S^{(1,1)}$ и $S^{(1,2)}$ передаются на следующий уровень рекурсивной обработки.

Данный метод использует в качестве примитивных процедуры вычисления площади треугольника и определения положения точки относительно прямой. Каждая из этих процедур требует нескольких операций сложения и умножения.

Объяснив в общих чертах предлагаемый подход, укажем способ, позволяющий подогнать под общую схему обработки явную аномалию, связанную с первоначальным разбиением множества точек. Соответствующий выбор начальных значений $\{l_0, r_0\}$ точек $l$ и $r$ может быть выполнен следующим образом. Выберем в качестве $l$ точку $(x_0, y_0)$, имеющую наименьшую абсциссу, а в качестве $r_0$ возьмем точку $(x_0, y_0 - \varepsilon)$, где $\varepsilon$ — произвольное малое положительное число. Это приводит к тому, что в качестве исходной прямой, разбивающей множество на части, выбирается вертикальная прямая, проходящая через $l_0$. После завершения алгоритма точка $r_0$ удаляется (положив $\varepsilon = 0$, делаем точки $l_0$ и $r_0$ совпадающими).

Теперь можно привести более формальное описание предлагаемого метода. Предполагается, что множество $S$ содержит по крайней мере две точки, а функция САМАЯДАЛЬНЯЯТОЧКА$(S; l, r)$ вычисляет точку $h \in S$ описаным выше способом; кроме того, БЫСТРОБОЛ возвращает в качестве результата упорядоченный список точек, а «*» обозначает операцию сцепления (конкатенации) списков.

```
   function БЫСТРОБОЛ(S; l, r)
1. begin if(S = {l, r}) then return(l, r) (* выпуклая оболочка
   состоит из единственного ориентированного ребра *)
2.     else begin h := САМАЯДАЛЬНЯЯТОЧКА(S; l, r);
                   S^(1) := точки множеств S, расположенные
                   слева от lh или на ней;
4.                 S^(2) := точки множества S, расположенные слева от hr или на ней;
5.                 return БЫСТРОБОЛ(S^(1); l, r) *
                   (БЫСТРОБОЛ(S^(2); h, r) — h)
           end
   end
```

Таким образом, если уже имеется функция БЫСТРОБОЛ, то поставленную задачу можно решить с помощью следующей простой программы:



```
begin l_0 = (x_0, y_0) := точка множества S с наименьшей
абсциссой;
      r_0 := (x_0, y_0 — ε);
      БЫСТРОБОЛ(S; l_0, r_0);
      удалить точку r_0 (* это эквивалентно тому, чтобы
      положить ε = 0 *)
end.
```

Для того чтобы выделить из множества $S$ подмножества $S^{(1)}$ и $S^{(2)}$ (с неявным удалением точек, попадающих внутрь треугольника $(lrh)$), как это делается в строках 2, 3 и 4 приведенной выше программы, требуется $O(N)$ операций. Затем следует рекурсивное обращение к функции для обработки $S^{(1)}$ и $S^{(2)}$. Теперь, если мощность каждого из этих множеств не превосходит мощности множества $S$, умноженной на некоторую константу, меньшую 1, и это условие имеет место на каждом уровне рекурсии, то время выполнения алгоритма равно $O(N \log N)$. Однако в худшем случае БЫСТРОБОЛ, несмотря на его простоту, имеет тот же недостаток, что и БЫСТРСОРТ, дающий время выполнения $O(N^2)$.

### 3.3.5. Алгоритмы типа «разделяй и властвуй»

Описанный в предыдущем разделе метод БЫСТРОБОЛ не только прост в реализации, но и является попыткой достигнуть одну из целей, сформулированных в конце разд. 3.3.2 (цель 4 — возможность распараллеливания алгоритма). Действительно, алгоритм разбивает исходную задачу на две подзадачи, каждая из которых может решаться независимо от другой, и, следовательно, они могут решаться одновременно при возможности параллельной обработки. Кроме того, объединение результатов решения двух подзадач — так называемый шаг «слияния» — представляет очень простую конкатенацию двух результатов. К сожалению, эта простота является платой за невозможность правильно управлять размерами двух подзадач, на которые разбивается исходная задача, и тем самым гарантировать приемлемое поведение алгоритма в худшем случае.

Следовательно, несмотря на то что он представляет метод «разделяй и властвуй», являющийся фундаментальным в области разработки алгоритмов, БЫСТРОБОЛ имеет очевидные недостатки. В самом деле, подход «разделяй и властвуй» основан на принципе сбалансированности [Aho, Hopcraft, Ullman (1974), с. 75 [1])], предполагающем, что вычислительная задача должна разбиваться на подзадачи примерно одинакового размера.

[1]) Ссылка дается на русское издание. — *Прим. перев.*

Предположим, при решении задачи построения выпуклой оболочки, исходное множество точек было разбито на две части $S_1$ и $S_2$, каждая из которых содержит половину точек исходного множества. Если теперь рекурсивным образом найдены отдельно $CH(S_1)$ и $CH(S_2)$, то каковы дополнительные затраты, необходимые для построения $CH(S_1 \cup S_2)$, т. е. выпуклой оболочки исходного множества? Для ответа на этот вопрос можно воспользоваться следующим соотношением:

$$CH(S_1 \cup S_2) = CH(CH(S_1) \cup CH(S_2)). \tag{3.7}$$

Хотя при первом взгляде на соотношение (3.7) кажется, что такой способ требует больших затрат, чем прямой способ построения выпуклой оболочки, решающее соображение состоит в том,

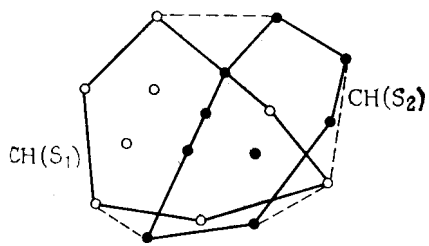



Рис. 3.11. Построение выпуклой оболочки методом «разделяй и властвуй». Разбив множество $S$ на два подмножества и построив рекурсивно выпуклые оболочки этих подмножеств, можно свести исходную задачу к построению выпуклой оболочки объединения двух выпуклых многоугольников.

что $CH(S_1)$ и $CH(S_2)$ — это не просто неупорядоченные множества точек, а выпуклые многоугольники (рис. 3.11).

**Задача BO4** (ВЫПУКЛАЯ ОБОЛОЧКА ОБЪЕДИНЕНИЯ ВЫПУКЛЫХ МНОГОУГОЛЬНИКОВ). Заданы два выпуклых многоугольника $P_1$ и $P_2$. Найти выпуклую оболочку их объединения.

Помимо того что эта задача интересна сама по себе, она важна и в силу того, что представляет собой шаг слияния процедуры «разделяй и властвуй» и, таким образом, дает фундаментальное средство для решения геометрических задач. Нельзя надеяться, что окончательный алгоритм будет эффективным, до тех пор, пока решения подзадач не могут быть быстро объединены.

**procedure** СЛИОБОЛ($S$)

1. Если $|S| \leqslant k_0$ ($k_0$ — некоторое небольшое целое число), то построить выпуклую оболочку одним из прямых методов и остановиться; иначе перейти к шагу 2.



2. Разбить исходное множество $S$ произвольным образом на два примерно равных по мощности подмножества $S_1$ и $S_2$.
3. Рекурсивно найти выпуклые оболочки $S_1$ и $S_2$.
4. Слить (соединить) две получившиеся оболочки вместе, образуя $CH(S)$.

Обозначим символом $U(N)$ время, необходимое для нахождения выпуклой оболочки объединения двух выпуклых многоугольников, каждый из которых имеет $N/2$ вершин. Если $T(N)$ — время, необходимое для нахождения выпуклой оболочки множества из $N$ точек, то, применяя соотношение (3.7), получаем

$$T(N) \leqslant 2T(N/2) + U(N). \tag{3.8}$$

Следующий алгоритм «слияния» предложен М. Шеймосом [Shamos (1978)]:

**procedure** ОБОЛОЧКА-ОБЪЕДИНЕНИЯ-ВЫПУКЛЫХ-МНОГОУГОЛЬНИКОВ($P_1$, $P_2$)

1. Найти некоторую внутреннюю точку $p$ многоугольника $P_1$. (Например, центроид трех любых вершин $P_1$. Такая точка $p$ будет внутренней точкой $CH(P_1 \cup P_2)$.)

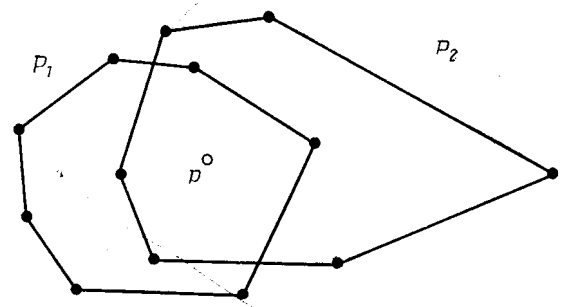



Рис. 3.12. Точка $p$ находится внутри многоугольника $P_2$. Так как точка $p$ одновременно находится внутри обоих многоугольников $P_1$ и $P_2$, то их вершины упорядочены по значению полярного угла точки $p$. Слияние двух упорядоченных множеств вершин можно выполнить за линейное время.

2. Определить, является ли $p$ внутренней точкой $P_2$. Это может быть сделано за время $O(N)$ методом, описанным в разд. 2.2.1. Если $p$ не является внутренней точкой $P_2$, перейти к шагу 4.
3. $p$ является внутренней точкой $P_2$ (рис. 3.12). По теореме 3.6 вершины как $P_1$, так и $P_2$ оказываются упорядоченными в соответствии со значением полярного угла относительно точки $p$. За время $O(N)$ можно получить упорядоченный список вершин как $P_1$, так и $P_2$ путем слияния списков вершин этих многоугольников. Перейти к шагу 5.

4. $p$ не является внутренней точкой $P_2$ (рис 3.13). Если смотреть из точки $p$, то многоугольник $P_2$ лежит в клине с углом разворота меньшим или равным $\pi$. Этот клин определяется двумя вершинами $u$ и $v$ многоугольника $P_2$, которые могут быть найдены за линейное время за один обход вершин многоугольника $P_2$. Эти вершины разбивают границу $P_2$ на две цепи вершин, являющиеся монотонными относительно изменения полярного угла с началом в $p$. При движении по одной цепи угол увеличивается, при движении по другой — уменьшается. Одна из этих двух цепей, являющаяся выпуклой по направлению к точке $p$, может быть немедленно удалена, так как ее вершины будут внутренними точками $\mathrm{CH}(S_1 \cup S_2)$. Другая цепь $P_2$ и граница $P_1$ представляют два упорядоченных списка, содержащих в сумме не более $N$ вершин. За время $O(N)$ их можно слить в один список вершин $P_1 \cup P_2$, упорядоченных по углу относительно точки $p$.

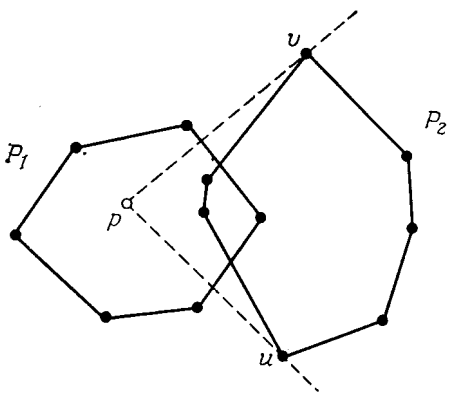



Рис. 3.13. Точка $p$ находится вне многоугольника $P_2$. Многоугольник $P_2$ находится внутри угла, определяемого точками $v$, $p$, $u$. Точки $v$ и $u$ разбивают последовательность вершин многоугольника $P_2$ на две цепи. Одну из них можно удалить, а слияние вершин второй цепи с упорядоченным множеством вершин многоугольника $P_1$ можно выполнить за линейное время.

5. Теперь к полученному списку можно применить метод обхода Грэхема (шаг 3 алгоритма ОБОЛОЧКА-ГРЭХЕМА), требующий лишь линейное время. Теперь получена выпуклая оболочка $P_1 \cup P_2$.

Если многоугольник $P_1$ имеет $m$ вершин, а $P_2$ имеет $n$ вершин, то время выполнения этого алгоритма равно $O(m+n)$, что со всей очевидностью является оптимальным. Так что теперь известно, что $U(N) = O(N)$, и решением рекуррентного соотношения (3.8) в этом случае является $T(N) = O(N \log N)$. Таким образом, имеет место

**Теорема 3.9.** *Выпуклая оболочка объединения двух выпуклых многоугольников может быть найдена за время, пропорциональное суммарному числу их вершин.*

Побочным результатом описанного метода слияния является вычисление (нахождение) «опорных прямых», если они имеются, для двух выпуклых многоугольников. *Опорной прямой* к выпуклому многоугольнику $P$ называется прямая $l$, проходящая через некоторую вершину многоугольника $P$ таким образом, что внутренность $P$ целиком находится по одну сторону от $l$ (в некотором смысле понятие опорной прямой аналогично понятию касательной). Очевидно, что два выпуклых многоугольника $P_1$ и $P_2$ с $n$ и $m$ вершинами соответственно, таких, что один не лежит целиком внутри другого, имеют общие опорные прямые (по крайней мере не более $2\min(n, m)$). Если уже получена выпуклая оболочка объединения $P_1$ и $P_2$, то опорные прямые вычисляются в результате просмотра списка вершин $\mathrm{CH}(P_1 \cup P_2)$. Каждая пара последовательных вершин $\mathrm{CH}(P_1 \cup P_2)$, одна из которых принадлежит $P_1$, а другая $P_2$, определяет опорную прямую.



Препарата и Хонг (Preparata, Hong (1977)) предложили иной метод нахождения выпуклой оболочки объединения двух непересекающихся выпуклых многоугольников, основанный на нахождении за линейное время двух опорных прямых к этим многоугольникам. Однако этот метод здесь демонстрироваться не будет.

### 3.3.6. Динамические алгоритмы построения выпуклой оболочки

В каждом из рассматривавшихся ранее алгоритмов требовалось, чтобы все обрабатываемые данные (а именно точки) были целиком представлены до начала работы алгоритма. Во многих прикладных областях, где возникают геометрические задачи, в частности связанные с обработкой данных в реальном времени, такое ограничение отсутствует, а ряд вычислений должен выполняться по мере поступления точек (данных). В общем случае алгоритм, обрабатывающий данные по мере поступления, называется *открытым*, а алгоритм, обрабатывающий всю совокупность целиком, называется *закрытым*.

Основной чертой открытых алгоритмов является отсутствие ограничений на время коррекции, что равносильно тому, что новый элемент данных (точка) вводится по запросу сразу же после того, как завершается коррекция, связанная с предыдущим элементом данных. Будем называть временной интервал между вводом двух последовательных элементов данных *задержкой поступления данных*. Более сложный в смысле предъявляемых требований случай применения открытых алгоритмов возникает, когда задержка поступления данных находится вне контроля алгоритма. Иначе говоря, данные поступают не по запросу алгоритма, а подаются на вход алгоритма в некоторые моменты времени, никак не связанные с работой алгоритма.

Однако мы будем считать, что поступление данных происходит (распределено) *равномерно* по времени. В такой ситуации коррекция должна выполняться за время, не превышающее постоянную задержку поступления данных. Алгоритмы, работающие в таком режиме, и называются соответственно алгоритмами *реального времени*. Следует отметить, что обычно открытые алгоритмы глобально являются менее эффективными по сравнению с соответствующими закрытыми алгоритмами (какая-то плата должна быть внесена за открытость алгоритма).

Целью этого раздела является разработка открытых алгоритмов построения выпуклой оболочки, в полной мере удовлетворяющих конкретным требованиям.

**Задача ВО5** (ОТКРЫТЫЙ АЛГОРИТМ ДЛЯ ВЫПУКЛОЙ ОБОЛОЧКИ). На плоскости задана последовательность из $N$ точек $p_1, \ldots, p_N$. Требуется найти их выпуклую оболочку, организовав обработку таким образом, чтобы после обработки точки $p_i$ имелась $\mathrm{CH}(\{p_1, \ldots, p_i\})$.

**Задача ВО6** (ВЫПУКЛАЯ ОБОЛОЧКА В РЕАЛЬНОМ ВРЕМЕНИ). На плоскости задана последовательность из $N$ точек $p_1, \ldots, p_N$. Требуется найти их выпуклую оболочку при условии, что время задержки поступления точек постоянно.

Соответствующий алгоритм должен поддерживать некоторое представление текущей выпуклой оболочки и корректировать его по мере поступления точек. Вопрос состоит в том, можно ли сделать это, не принося в жертву оценку $O(N \log N)$ времени выполнения алгоритма, необходимого на обработку всего множества точек.

Нет проблемы разработать открытый алгоритм построения выпуклой оболочки, если время обработки не принимается во внимание. Например, после получения каждой точки можно было бы выполнять алгоритм Грэхема, получив при этом оценку сложности $O(N^2 \log N)$. Можно проявить большую сообразительность, вспомнив, что алгоритм Грэхема состоит из двух отдельных шагов — сортировки и обхода.

1. Вводить точки до тех пор, пока не будут обнаружены три неколлинеарные точки. Их центроид будет внутренней точкой окончательной выпуклой оболочки и, таким образом, подходит для начала координат, относительно которого определяются полярные углы точек при сортировке в алгоритме ОБОЛОЧКАГРЭХЕМА.

2. Поддерживать связанный список упорядоченных крайних точек. При поступлении точки $p_i$ временно вставить ее в этот список в соответствии с ее полярным углом, затратив на это время $O(i)$.



3. Выполнить просмотр связанного списка крайних точек методом Грэхема. Так как просмотр методом Грэхема является линейным, то на это потребуется лишь $O(i)$ времени. Возможны три варианта завершения этого шага:

a) точка $p_i$ является крайней обработанного на текущий момент множества, и ее включение в число крайних точек вызывает удаление нескольких других точек;

b) точка $p_i$ является крайней, но никакие другие точки не удаляются;

c) точка $p_i$ является внутренней для создаваемой выпуклой оболочки, и поэтому она удаляется.

В каждом случае размер списка увеличивается не более чем на единицу. Полное время, затрачиваемое этим алгоритмом, в худшем случае равно $O(N^2)$, что имеет место, если каждая новая точка оказывается крайней точкой.

Может показаться, что предыдущая процедура могла бы быть улучшена, если на шаге 2 можно было бы вместо линейной вставки делать *бинарную* вставку и в соответствии с этим на шаге 3 более эффективно (т. е. логарифмически) выполнять поиск вместо использования линейного просмотра методом Грэхема.

Следуя по этому пути, Шеймос [Shamos (1978)] разработал алгоритм, который для множества из $N$ точек выполняется глобально за время $O(N \log N)$ и, следовательно, оптимален. Но при этом на корректировку тратится время $O(\log^2 N)$. Чтобы определить, насколько этот алгоритм соответствует требованиям, предъявляемым к алгоритмам для обработки в реальном времени, заметим, что нижние оценки, полученные для закрытых алгоритмов, в равной мере применимы и к открытым алгоритмам. Этот факт можно использовать для получения нижних оценок времени, которое открытый алгоритм должен тратить на обработку очередной порции данных. Пусть $T(N)$ — время, используемое открытым алгоритмом в худшем случае, при решении задачи с $N$ элементами вводимых данных; $U(i)$ — время, используемое на обработку между поступлением $i$-го и $(i+1)$-го элементов данных. Если $L(N)$ — нижняя оценка времени, достаточного для решения задачи, то из соотношения

$$T(N) = \sum_{i=1}^{N-1} U(i) \geqslant L(N) \tag{3.9}$$

находим нижнюю оценку для $U(i)$.

В нашем случае совместное использование теоремы 3.2 и соотношения (3.9) приводит к следующей теореме, устанавливающей для заданного $N$ нижнюю оценку для константы задержки поступления данных, упомянутой в формулировке задачи ВО6.

**Теорема 3.10.** *Любой открытый алгоритм построения выпуклой оболочки в худшем случае должен затрачивать на обработку между поступлением последовательных элементов данных время* $\Omega(\log N)$.

Таким образом, можно заключить, что упомянутый выше алгоритм не удовлетворяет требованию, предъявляемому к алгоритмам реального времени. Однако впоследствии Препарата [Preparata (1979)] разработал открытый алгоритм с той же самой глобальной оценкой временной сложности, но со временем коррекции $O(\log N)$, что сопоставимо с оценкой времени задержки поступления, установленной в теореме 3.10. Оба этих

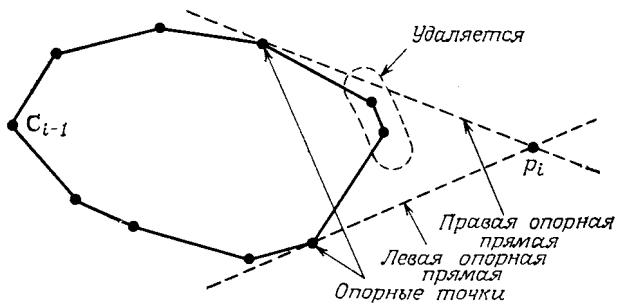



Рис. 3.14. Опорные прямые из точки $p_i$ к выпуклому многоугольнику $C_{i-1}$.

алгоритма тесно связаны друг с другом, и мы ограничимся лишь обсуждением последнего из них.

Ключ к созданию эффективного открытого алгоритма дает следующее соображение: единственное, что необходимо, — это уметь быстро строить две опорные прямые к выпуклому многоугольнику, проходящие через некоторую точку (см. разд. 3.3.5). В частности, если точки обрабатываются в порядке $p_1, p_2, \ldots$ и $p_i$ — текущая точка, то обозначим через $C_{i-1}$ выпуклую оболочку множества $\{p_1, p_2, \ldots, p_{i-1}\}$. Необходимо построить из точки $p_i$ опорные прямые к $C_{i-1}$, если они существуют (т. е. если точка $p_i$ является внешней для $C_{i-1}$). Таких прямых не существует тогда и только тогда, когда точка $p_i$ является внутренней для $C_{i-1}$. В последнем случае точка $p_i$ просто удаляется. В первом случае должна быть удалена соответствующая цепочка вершин, заключенная между двумя опорными точками, и вместо них должна быть вставлена точка $p_i$. Эта ситуация показана на рис. 3.14. Для удобства мы будем называть опорную прямую *левой* или *правой* в соответствии с тем, по какую сторону она находится, если смотреть из точки $p_i$ на $C_{i-1}$.

Рассмотрим построение опорных прямых из некоторой точки $p$ к выпуклому многоугольнику $C$. (Структура данных для



представления вершин $C$ пока не определена. Мы выберем ее после того, как определим, какие операции с ней будут проводиться.) В последующем обсуждении важную роль будет играть классификация вершин многоугольника $C$ по отношению к отрезку $\overline{pv}$, соединяющему вершину $v$ с точкой $p$. Вершина $v$ называется *вогнутой* (следовало бы добавить «по отношению к отрезку $\overline{pv}$», но обычно это уточнение будет опускаться), если

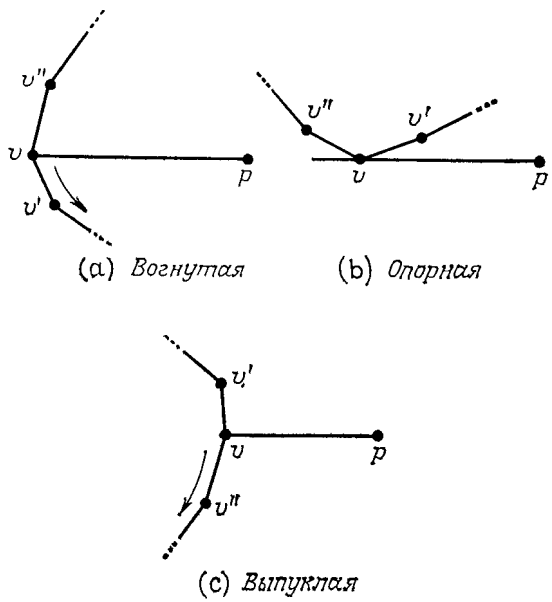



Рис. 3.15. Классификация вершины $v$ многоугольника $C$ относительно отрезка $\overline{pv}$. (Стрелки указывают направление обхода при поиске левой опорной прямой.)

отрезок $\overline{pv}$ пересекает внутренность многоугольника $C$. Иначе, если две смежные с $v$ вершины лежат по одну сторону от прямой, проходящей через точки $p$ и $v$, вершина $v$ называется *опорной*. В оставшемся случае $v$ называется *выпуклой* вершиной. Мы оставляем читателю в качестве упражнения показать, что вершина $v$ может быть классифицирована за постоянное время.

Если $v$ — опорная вершина, то на этом решение задачи завершается. Предположим, что ищется левая опорная прямая. Если точка $v$ не является опорной, то необходимо шагать (или еще лучше прыгать!) по вершинам многоугольника $C$ против или по часовой стрелке в зависимости от того, является ли $v$ вогнутой или выпуклой вершиной (рис. 3.15). Таким способом

можно определить две опорные точки (если они существуют). Если это сделано, необходимо иметь возможность удалить из циклической последовательности вершин многоугольника $C$ цепочку вершин (возможно, пустую) и вставить в образовавшийся разрыв точку $p$.

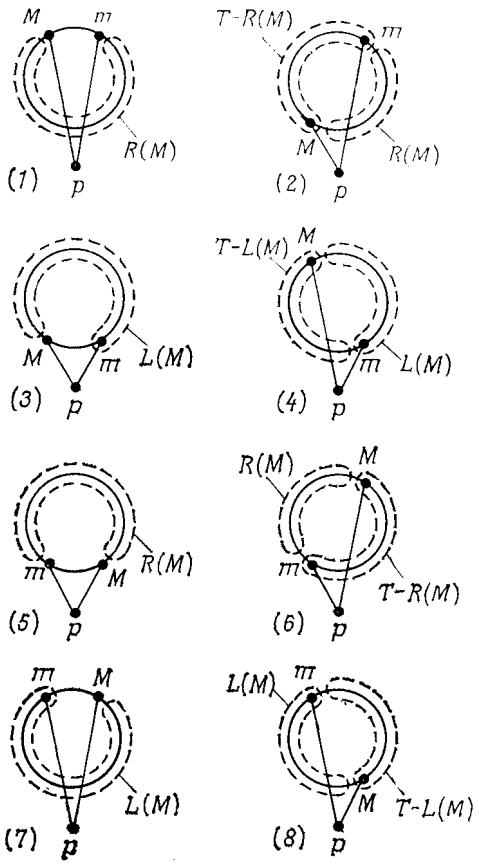



Рис. 3.16. Восемь возможных случаев в зависимости от классификации вершин $m$, $M$ и угла $\alpha$.

Теперь стало ясно, какой должна быть структура данных для открытого алгоритма. Необходимо иметь возможность эффективно выполнять следующие операции:

I. ПОИСК в упорядоченной последовательности элементов (кольцевого списка вершин оболочки) для определения опорных прямых из точки $p_i$;

II. РАСЦЕПЛЕНИЕ последовательности на две последовательности и СЦЕПЛЕНИЕ двух последовательностей;

III. ВСТАВКА одного элемента (текущей точки $p_i$).

Структура данных, в полной мере удовлетворяющая перечисленным требованиям, хорошо известна и называется *сцепляемой очередью* [1]). Она реализуется с помощью сбалансированного по высоте дерева поиска, и при этом каждая из указанных выше операций может быть выполнена за время $O(\log i)$ в худшем случае, где $i$ — число узлов дерева. Естественно, кольцевая последовательность вершин представляется в этой древовидной структуре данных (называемой далее $T$) цепочкой, и при этом первый и последний элементы считаются смежными.

[1]) См. разд. 1.2.3, а также [Aho, Hopcroft, Ullman (1974)] и [Reingold, Nievergelt, Deo (1977)], где этот вопрос рассмотрен более подробно.



В структуре $T$ будут выделены две вершины: $m$ — самый левый член цепочки и $M$ — корневой член цепочки. Кроме того, мы будем использовать угол $\angle(mp_iM)$, который обозначим $\alpha$. Этот угол называется *выпуклым*, если он меньше или равен $\pi$, и *вогнутым* в противном случае.

В зависимости от классификации вершин $m$ и $M$ (вогнутая, опорная или выпуклая) и угла $\alpha$ возможны всего восемнадцать случаев. Однако все эти случаи можно свести к восьми (покрывающим все возможности), сведенным в табл. I и показанным

*Таблица I*

| Случай | $\alpha$ | $m$ | $M$ |
|---|---|---|---|
| 1 | выпуклый | вогнутая | вогнутая |
| 2 | выпуклый | вогнутая | невогнутая |
| 3 | выпуклый | невогнутая | выпуклая |
| 4 | выпуклый | невогнутая | невыпуклая |
| 5 | вогнутый | выпуклая | выпуклая |
| 6 | вогнутый | выпуклая | невыпуклая |
| 7 | вогнутый | невыпуклая | вогнутая |
| 8 | вогнутый | невыпуклая | невогнутая |

на рис. 3.16. Диаграммы на рис. 3.16 расшифровываются следующим образом: окружность, на которой лежат точки $M$ и $m$, обозначает многоугольник $P$; упорядоченная последовательность вершин начинается в точке $m$ и располагается по окружности в порядке против часовой стрелки; $L(M)$ и $R(M)$ — это последовательности вершин, хранимые в левом и правом поддеревьях корня дерева $T$.

Каждый из представленных на рис. 3.16 случаев требует различной обработки для определения левой и правой опорных точек, обозначаемых соответственно $l$ и $r$.

Рассмотрим сначала случаи 2, 4, 6 и 8. Для этих случаев известно, что $l$ и $r$ существуют (так как $p$ не может быть внутренней точкой многоугольника) и их следует искать в различных поддеревьях корня дерева $T$ (одно из этих поддеревьев расширяется, чтобы включить сам корень). Таким образом, поиск $l$ и $r$ должен проходить одинаково. Например, следующая процедура осуществляет поиск вершины $l$:

**function** ЛЕВЫЙ-ПОИСК($T$)
**Input**: дерево $T$, описывающее последовательность вершин
**Output**: вершина $l$
1. **begin** $c :=$ КОРЕНЬ($T$);
2. **if** ($\overline{pc}$ — опорная прямая) **then** $l := c$

```
3.      else begin if (c — выпуклая вершина) then
                   T := ЛДЕРЕВО(c) else T := ПДЕРЕВО(c)
4.                 l := ЛЕВЫЙ-ПОИСК(T)
            end;
5.      return l
    end.
```

Очевидно, что функция ЛЕВЫЙ-ПОИСК проходит по дереву $T$ некоторый путь, затрачивая в каждом узле ограниченное время на классификацию соответствующей узлу вершины. То же самое имеет место и для функции ПРАВЫЙ-ПОИСК.

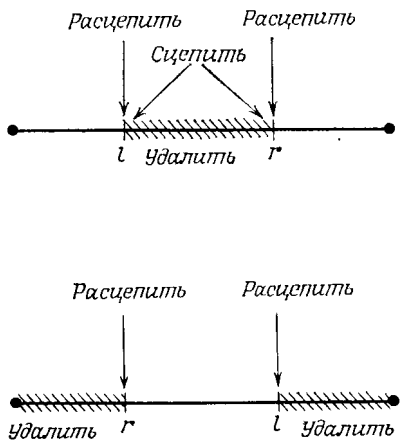



Рис. 3.17. Для удаления точек, оказавшихся внутри выпуклой оболочки, требуется выполнить различные действия в зависимости от относительного положения вершин $l$ и $r$.

Теперь рассмотрим случаи 1, 3, 5 и 7. В каждом из этих случаев обе вершины $l$ и $r$ следует искать в одном и том же поддереве корня, если они только существуют (следует отметить, что в случаях 1 и 7 точка $p$ может быть внутренней точкой $P$). Таким образом, в каждом случае процедура поиска рекурсивно вызывает себя для поиска в поддереве текущего дерева [1]) (выбирается поддерево, соответствующее последовательности, обведенной на рис. 3.16 штриховой линией) до тех пор, пока не встретится один из случаев 2, 4, 6 или 8. В этом месте инициируется раздельный поиск с помощью функций ЛЕВЫЙ-ПОИСК и ПРАВЫЙ-ПОИСК. Если в процессе поиска будет достигнут лист дерева $T$ и при этом не имел место ни один из случаев 2, 4, 6 или 8, то точка $p$ является внутренней для многоугольника. В заключение отметим, что в самом общем случае можно наглядно представить процесс поиска опорных прямых, прослеживая начальный участок пути из корня $T$ до узла $c$, в котором происходит разветвление пути на два. Учитывая, что

[1]) Для того чтобы каждое рекурсивное обращение выполнялось за время $O(1)$, необходимо, чтобы элементы $m$ и $M$ поддерева были легко доступны. Для $M$, являющегося корнем дерева, это не составляет проблемы. Однако элемент $m$ в правом поддереве будет достижим из текущего корня, если «прошить» дерево, т. е. ввести указатель СЛЕДУЮЩИЙ, связывающий вершину $v$ со следующей за ней вершиной $v_{i+1}$ на границе многоугольника.



дерево $T$ сбалансировано и содержит не более $i < N$ вершин, а на обработку в каждом узле требуется ограниченное время, получаем для времени поиска оценку $O(\log i)$.

Чтобы завершить описание, рассмотрим процедуру перестройки многоугольника $C_{i-1}$, выполняемую, если точка $p_i$ является внешней для него. Вершины между $l$ и $r$ должны быть удалены, а $p_i$ должна быть вставлена на их место. В зависимости от того, предшествует вершина $l$ вершине $r$ в дереве $T$ или нет, требуется выполнить немного различные действия (рис. 3.17). В первом случае необходимо дважды выполнить операцию расцепления и один раз сцепления; во втором случае только дважды выполняется операция расцепления.

Как уже упоминалось ранее, обе операции РАСЦЕПИТЬ и СЦЕПИТЬ выполняются за время $O(\log i)$. В результате, учитывая, что коррекция выпуклой оболочки может быть выполнена за время $O(\log i)$, имеем следующую теорему:

**Теорема 3.11.** *Выпуклая оболочка множества из $N$ точек на плоскости может быть найдена с помощью открытого алгоритма за время $\theta(N \log N)$ со временем коррекции $\theta(\log N)$, т. е. может быть построена в реальном времени.*

### 3.3.7. Обобщение: поддержка динамической выпуклой оболочки

Описанный в предыдущем разделе метод можно рассматривать как метод поддержки структуры данных, описывающей выпуклую оболочку множества точек в случае, когда допускается лишь операция *добавления* (вставки) точек. При этом вполне естественно возникает вопрос: можно ли разработать структуру данных, организующую множество точек на плоскости и описывающую их текущую выпуклую оболочку, в предположении, что допускаются не только *добавление* (вставка), но и *удаление* точек?

Как можно было ожидать, на этот вопрос нет простого ответа. В самом деле, в то время как в открытом алгоритме построения выпуклой оболочки из разд. 3.3.6 точки, про которые становится известно, что они являются внутренними точками текущей выпуклой оболочки, навсегда исключаются из рассмотрения, в этой новой ситуации необходимо с одинаковой тщательностью организовывать *все* точки, содержащиеся на текущий момент в множестве, так как удаление некоторой точки текущей выпуклой оболочки может привести к тому, что несколько внутренних точек станут граничными точками новой выпуклой оболочки (рис. 3.18). Эта задача рассматривалась Овермарсом и Ван Леювеном. Приведем формальную постановку этой задачи:

**Задача ВО7** (ПОДДЕРЖКА ОБОЛОЧКИ). Заданы изначально пустое множество $S$ и последовательность из $N$ точек

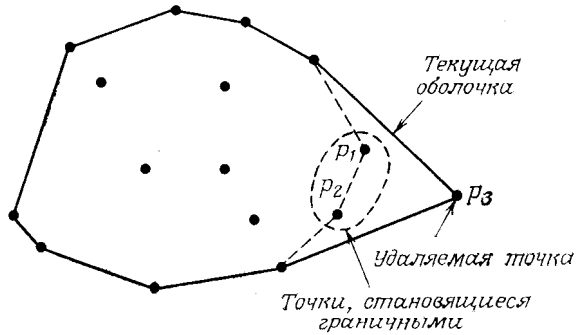



Рис. 3.18. При удалении точки $p_3$ точки $p_1$ и $p_2$ становятся граничными точками выпуклой оболочки.

$(p_1, p_2, \ldots, p_N)$, каждая из которых либо добавляется к множеству $S$, либо удаляется из него (разумеется, точка может быть удалена лишь при условии, что она содержится в $S$). Требуется поддерживать выпуклую оболочку множества $S$.

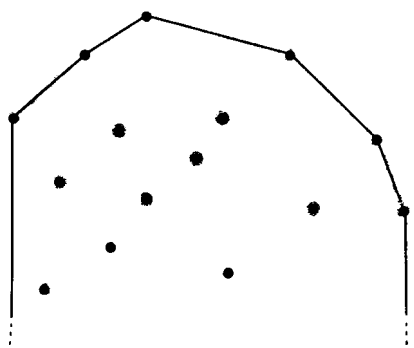


Рис. 3.19. В-оболочка множества точек.

Теперь мы проиллюстрируем очень интересное решение этой задачи, предложенное Овермарсом и Ван Леювеном [Overmars, van Leeuwen (1981)].

Прежде всего мы используем тот факт, что граница выпуклой оболочки представляет объединение двух (выпуклых) монотонных ломаных линий (цепей). Эти ломаные ограничивают оболочку сверху и снизу и в соответствии с этим называются В-оболочкой и Н-оболочкой множества точек («В» обозначает верхнюю, а «Н» — нижнюю оболочку). Например, В-оболочка множества точек $S$ получается как обычная выпуклая оболочка множества $S \cup \{\infty_H\}$ (рис. 3.19), где $\infty_H$ точка плоскости с координатами $(0, -\infty)$[1]). Поняв, что выпуклая оболочка множества $S$ тривиально получается в результате пересечения (сцепления) его В-оболочки и Н-оболочки, можно ограничить обсуждение вопросом построения одной из них, скажем В-оболочки.

[1]) Это понятие уже использовалось при обсуждении предложенного Эндрью варианта алгоритма построения оболочки методом Грэхема (см. разд. 3.3.2).



Нет ничего удивительного в том, что как открытый алгоритм построения выпуклой оболочки, рассмотренной в разд. 3.3.6, так и настоящий полностью динамический метод используют в качестве структур данных деревья. Однако в представлениях оболочки в этих двух подходах имеется существенное различие. При первом подходе каждый узел дерева представляет некоторую точку множества. При втором подходе только листья дерева используются для представления точек, а каждый внутренний узел представляет В-оболочку точек, соответствующих его листьям.

Соответствующая структура данных $T$ организована следующим образом. Ее основой является сбалансированное по высоте двоичное дерево поиска $T$ (с некоторой предосторожностью вместо него можно было бы использовать 2-3-дерево [Aho, Hopcroft, Ulman (1974), (с. 169 русского издания)]), листья которого используются для хранения точек текущего множества. Процедура поиска будет производиться в соответствии со значением абсциссы ($x$-координаты) точек, так что прохождение листьев дерева слева направо дает множество точек, упорядоченное по $x$-координате. Заметим, что последовательность точек В-оболочки (ее *вершин*) также упорядочена по возрастанию абсциссы, и, следовательно, она является *подпоследовательностью* глобальной последовательности точек, хранящейся в листьях дерева.

Пусть $v$ — узел дерева $T$. Обозначим через ЛСЫН[$v$] и ПСЫН[$v$] его левого и правого потомков соответственно. Мы хотим научиться строить В-оболочку точек, хранящихся в листьях поддерева с корнем в узле $v$. Так как для этого надо будет выполнять сцепление и расцепление цепей вершин, то мы предполагаем, что каждая такая цепь представлена в виде сцепляемой очереди (см. разд. 3.3.6). Обозначим через $U(v)$ В-оболочку множества точек, хранящихся в листьях поддерева с корнем в $v$. Следуя принципу индукции, предположим теперь, что уже имеются $U$(ЛСЫН[$v$]) и $U$(ПСЫН[$v$]), т. е. В-оболочки, связанные с потомками узла $v$. Каким образом можно построить $U(v)$? Взгляните на рис. 3.20. Все, что необходимо сделать, — это определить две опорные точки $p_1$ и $p_2$ единственного общего опорного отрезка для двух оболочек. Здесь нам необходима функция СОЕДИНИТЬ($U_1, U_2$), позволяющая найти опорный отрезок для двух В-оболочек $U_1$ и $U_2$. Функция СОЕДИНИТЬ позволяет эффективно расцеплять $U_1$ на две цепи, составляющие упорядоченную пару $(U_{11}, U_{12})$ (рис. 3.20), и аналогичным образом $U_2$ — на пару цепей $(U_{21}, U_{22})$. При этом выполняется следующее условие: опорная точка $p_1 \in U_1$ входит в состав $U_{11}$,

а точка $p_2 \in U_2$ входит в состав $U_{22}$ (т. е. в обоих случаях опорная точка принадлежит «внешней» подцепи). На этом этапе, сцепив $U_{11}$ и $U_{22}$, получаем искомую В-оболочку $U_1 \cup U_2$. Естественно, чтобы каждый узел $v$ дерева $T$ указывал на сцепляемую очередь, представляющую ту часть $U(v)$, которая не принадлежит $U$(ОТЕЦ[$v$]).

Предположим, что мы хотим выполнить обратную операцию, т. е., имея $U(v)$, получить $U$(ЛСЫН[$v$]) и $U$(ПСЫН[$v$]). Все, что требуется в этом случае, так это знать ребро $\overline{p_1p_2}$, соединяющее опорные точки, т. е. одно целое число $J[v]$, указывающее положение точки $p_1$ в цепи вершин $U(v)$. Имея эту информацию,

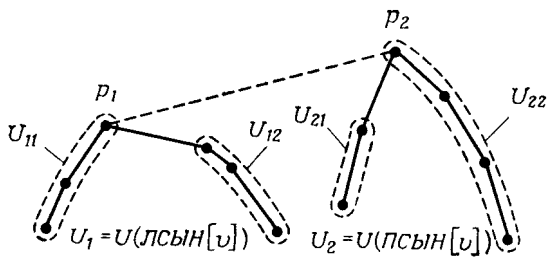



Рис. 3.20. Для построения В-оболочки объединения оболочек $U_1$ и $U_2$ необходимо построить общий опорный отрезок (мост) $\overline{p_1p_2}$.

можно расцепить $U(v)$ на цепи $U_{11}$ и $U_{22}$, которые в свою очередь могут быть сцеплены с цепями, хранящимися в ЛСЫН[$v$] и ПСЫН[$v$] соответственно. В заключение структура данных $T$ дополняется следующими атрибутами, связываемыми с каждым узлом $v$ дерева $T$:

1. указатель на сцепляемую очередь $Q[v]$, содержащую часть $U(v)$, не входящую в $U$(ОТЕЦ[$v$]) (если $v$ является корнем, то $Q[v] = U(v)$);

2. целым числом $J[v]$, указывающим положение (индекс) левой опорной точки в $U(v)$.

Эта интересная структура данных использует память объемом лишь $O(N)$, где $N$ — размер текущего множества точек. В самом деле, дерево $T$, являющееся основой структуры данных, имеет $N$ листьев и $N - 1$ узлов на более высоких уровнях, в то время как множества точек, хранимые в сцепляемых очередях, представляют разбиение множества всех точек.

Учитывая, что операции расцепления и сцепления сцепляемых очередей являются стандартными, наше внимание будет сосредоточено на операции, выполняемой функцией СОЕДИНИТЬ, для которой Овермарс и Ван Леювен [Overmars, van Leeuwen (1981)] предложили следующее решение:



**Лемма 3.1.** *Соединение двух разделимых выпуклых цепей, содержащих в сумме $N$ точек, может быть выполнено за $O(\log N)$ шагов.*

*Доказательство.* Пусть даны две В-оболочки $U_1$ и $U_2$ и две вершины $q_1 \in U_1$ и $q_2 \in U_2$. Каждая из этих двух вершин может быть легко классифицирована по отношению к отрезку $\overline{q_1q_2}$ как выпуклая, опорная или вогнутая (см. разд. 3.3.6, где дано объяснение этих терминов). В зависимости от этой классификации

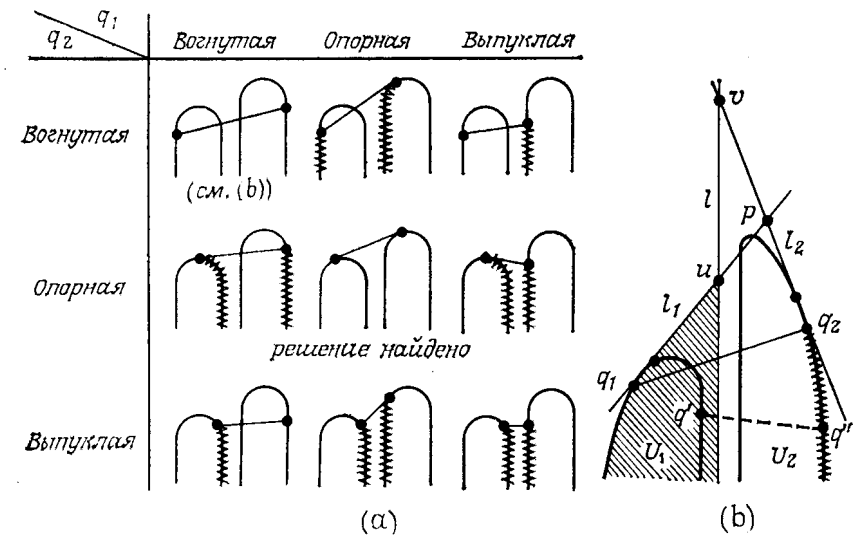



Рис. 3.21. (a) Все возможные случаи, возникающие при выборе некоторой вершины в $U_1$ и некоторой вершины в $U_2$; (b) иллюстрация случая (вогнутая, вогнутая).

возможны девять случаев, схематически показанных на рис. 3.21(a). Пилообразной линией на рисунке обозначены подцепи, которые можно не рассматривать в дальнейшем, так как они не содержат опорных точек. Все представленные случаи понятны без дополнительных объяснений, за исключением случая $(q_1, q_2)$ = (вогнутая, вогнутая), который более детально представлен на рис. 3.21(b). Пусть прямая $l_1$ проходит через $q_1$ и ее правого соседа на $U_1$. Аналогично прямая $l_2$ проходит через $q_2$ и ее левого соседа на $U_2$. Обозначим через $p$ точку пересечения прямых $l_1$ и $l_2$. Напомним, что по предположению $U_1$ и $U_2$ разделимы вертикальной прямой $l$. Предположим для начала, что точка $p$ находится справа от прямой $l$. Легко видеть, что опорная точка $p_1$ может принадлежать лишь заштрихованной на

рис. 3.21(b) области и что точка $u$ имеет меньшую ординату, чем точка $v$. Отсюда следует, что любая вершина $q''$, принадлежащая правой относительно $q_2$ подцепи, является вогнутой относительно отрезка $\overline{q'q''}$, где $q'$ — произвольная вершина $U_1$. Поэтому очевидно, что цепь справа от вершины $q_2$ можно не рассматривать, но что касается цепи слева от вершины $q_1$, то

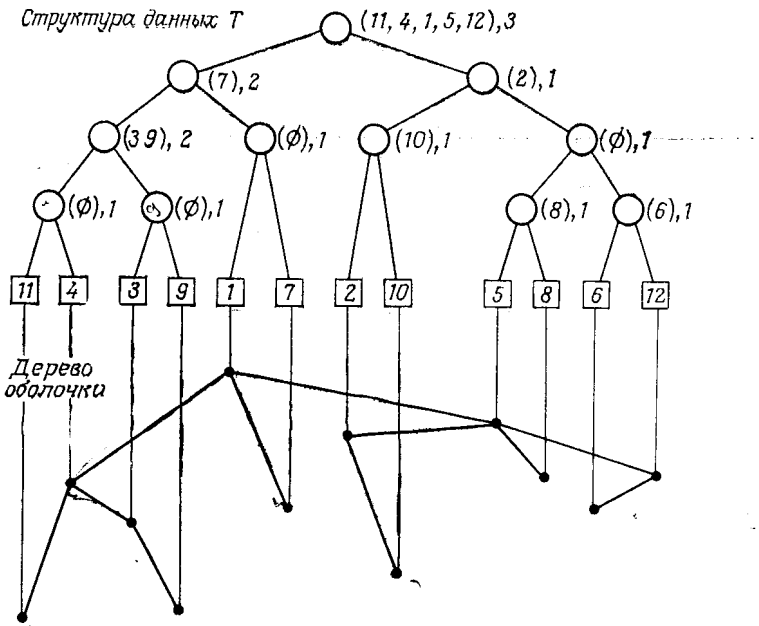



Рис. 3.22. Множество точек на плоскости и соответствующая ему структура данных $T$.

для нее нельзя сделать подобное утверждение. Если точка пересечения $p$ находится слева от разделяющей прямой $l$, то можно показать, что цепь слева от вершины $q_1$ можно не рассматривать.

Во всех случаях удаляется некоторая часть одной или двух оболочек. Если этот процесс начинается с корневых вершин обоих деревьев, представляющих $U_1$ и $U_2$ соответственно, то, так как эти деревья являются сбалансированными, время выполнения функции СОЕДИНИТЬ$(U_1, U_2)$ составит $O(\log N)$, где $N$ — это, как обычно, суммарное число вершин в двух оболочках.

Имея функцию СОЕДИНИТЬ, можно анализировать процесс динамической поддержки выпуклой оболочки на плоскости. Типичная ситуация показана на рис. 3.22. Номера точек соответствуют порядку, в котором они добавлялись к множеству.



В структуре данных $T$ каждый лист соответствует точке, а каждый узел, не являющийся листом, соответствует мосту (т. е. опорному отрезку, соединяющему две оболочки). Для каждого такого узла указана пара $Q[v]$, $J[v]$. Легко понять, что структура данных описывает некоторое свободное дерево на текущем множестве точек, которое называется *деревом оболочки*. Читателю

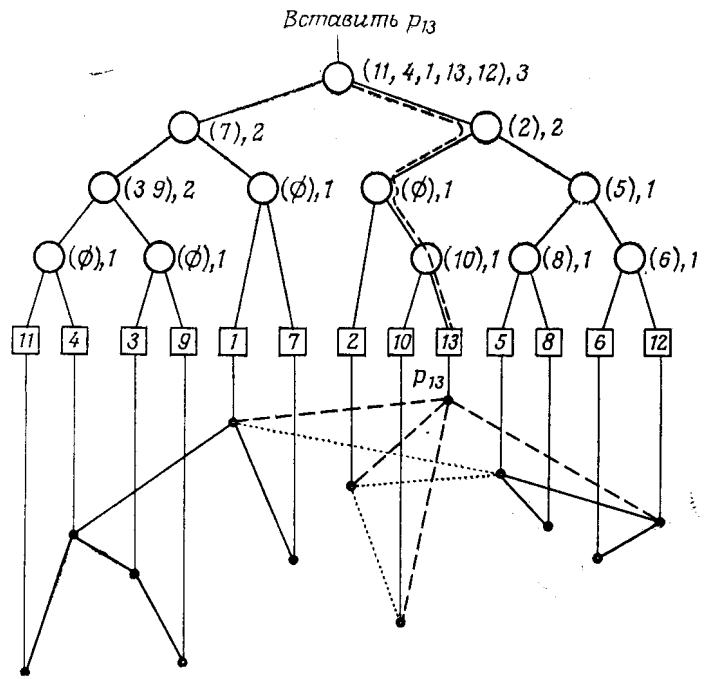



Рис. 3.23. Вставка точки $p_{13}$. При вставке затронуты узлы: {(11, 4, 1, 13, 12), 3}, {(2), 2}, {(∅), 1}, {(5), 1}, {(10), 1}, {13}.

следует затратить немного времени, чтобы полностью разобраться во всех деталях рис. 3.22.

Предположим теперь, что мы хотим вставить новую точку $p$. Процедура вставки должна не только давать сбалансированное по высоте дерево $T$, используя для этого обычный метод балансировки [Reingold, Nievergelt, Deo (1977)], но и делать все необходимое для того, чтобы была полная уверенность в том, что сцепляемая очередь, связанная с каждой вершиной, по-прежнему удовлетворяет требованиям, указанным в определении структуры данных $T$. Предположим для простоты, что балансировка не требуется (все действия по балансировке просто увеличивают не имеющим значения способом число узлов, подле-

жащих обработке, не приводя к каким-либо принципиальным отличиям). Поэтому будем считать, что точка $p_{13}$ добавляется к множеству точек, представленному на рис. 3.22, как это показано на рис. 3.23. На этом рисунке показано также состояние структуры данных $T$ после вставки точки $p_{13}$. Заметим, что $p_{13}$ однозначно определяет путь из корня дерева $T$ к листу, в который должна быть вставлена $p_{13}$. Двигаясь по этому пути из корня, в каждом узле, находящемся на этом пути, мы собираем B-оболочку, относящуюся к этому узлу, и затем, используя параметр $J[v]$, разбиваем ее на части, чтобы передать двум потомкам этого узла соответствующие им части. Действуя таким способом, будем иметь в каждом узле $u$, являющемся братом одного из узлов на этом пути, полное представление $U(u)$ в виде сцепляемой очереди $U[u]$. Более формально такой спуск по дереву с целью вставки точки $p$ осуществляется путем вызова процедуды СПУСК(корень($T$), $p$). СПУСК($v$, $p$) — это рекурсивная процедура, использующая функции РАСЦЕПИТЬ и СЦЕПИТЬ, обсуждавшиеся ранее. Приведем описание этой процедуры:

```
  procedure СПУСК(v, p)
begin if(v ≠ лист) then
        begin(Q_L, Q_R) := РАСЦЕПИТЬ(U[v]; J[v])
              U[ЛСЫН[v]] := СЦЕПИТЬ(Q_L, Q[ЛСЫН[v]]);
              U[ПСЫН[v]] := СЦЕПИТЬ(Q[ПСЫН[v]], Q_R);
              if (x[p] ⩽ x[v]) then v := ЛСЫН[v] else v :=
              ПСЫН[v];
              СПУСК(v, p)
        end
end.
```

Здесь мы вставляем новый лист и начинаем двигаться по тому же самому пути в $T$ к корню дерева. При достижении каждого узла $v$, лежащего на этом пути, у нас имеется полная B-оболочка, относящаяся к этому узлу. Как уже было показано, в узле $v$ доступна также B-оболочка брата узла $v$. Так что теперь мы соединяем эти две оболочки, используя эффективное расцепление B-оболочки $U[v]$ на две части $Q_1$ и $Q_2$ и соответственно $U$[БРАТ[$v$]] на $Q_3$ и $Q_4$. Части $Q_2$ и $Q_3$ должны сохраняться в узлах $v$ и БРАТ[$v$] соответственно, в то время как часть $Q_1$ должна быть передана в узел, являющийся отцом узла $v$, где она будет сцеплена с аналогичной частью $Q_4$, полученной от брата узла $v$. Таким образом мы получили B-оболочку узла, являющегося отцом узла $v$, и подъем к корню дерева может быть продолжен. И вновь, если говорить более формально, мы используем следующую процедуру:



```
  procedure ПОДЪЕМ(v)
begin if(v ≠ корень) then
        begin(Q_1, Q_2, Q_3, Q_4, J) := СОЕДИНИТЬ(U[v],
        U[БРАТ[v]]);
              Q[ЛСЫН[ОТЕЦ[v]]] := Q_2;
              Q[ПСЫН[ОТЕЦ[v]]] := Q_3;
              U[ОТЕЦ[v]] := СЦЕПИТЬ(Q_1, Q_4);
              J[ОТЕЦ[v]] := J;
              ПОДЪЕМ(ОТЕЦ[v])
        end;
      else Q[v] := U[v]
end.
```

Рассматривая эти процедуры с точки зрения времени их выполнения, напомним, что каждая из процедур РАСЦЕПИТЬ и СЦЕПИТЬ выполняется за время $O(\log k)$, где $k$ — размер очереди до расцепления или после сцепления. Так как $k \leqslant N$, то видно, что время обработки одного узла в процедуре СПУСК составляет $O(\log N)$. А так как глубина дерева $T$ также равна $O(\log N)$, то время выполнения процедуры СПУСК равно $O(\log^2 N)$ в худшем случае. Что касается процедуры ПОДЪЕМ, то, как мы видели ранее, время выполнения процедуры СОЕДИНИТЬ также равно $O(\log N)$, и поэтому та же оценка применима и в этом случае. Аналогичный анализ времени выполнения можно использовать и в случае удаления точки из текущего множества, и мы оставляем его в качестве упражнения для читателя. Таким образом, можно подвести итоги этого раздела, сформулировав следующую теорему:

**Теорема 3.12.** *Динамическая поддержка* B-*оболочки и* H-*оболочки множества из $N$ точек на плоскости может быть выполнена с временными затратами на операции вставки и удаления, равными $O(\log^2 N)$ в худшем случае.*

Заметим, что если пользоваться этим методом для построения выпуклой оболочки множества из $N$ точек в случае, когда допускается только добавление точек, то получим оценку для времени выполнения $O(N \log^2 N)$, более высокую по сравнению с $O(N \log N)$ для менее «мощного» метода. Ясно, что это является платой за использование метода, более мощного, чем требуется в конкретной задаче.

## 3.4. Выпуклые оболочки в пространствах размерности большей двух

Теперь мы готовы к тому, чтобы рассмотреть задачу построения выпуклой оболочки конечного множества точек в пространствах размерности большей двух. Мы уже видели, что

такая оболочка является выпуклым политопом. Мы также видели, что политопы в случае большого числа измерений уже не являются такими простыми геометрическими объектами, как их двумерные аналоги — выпуклые многоугольники. Однако напомним, что в трехмерном пространстве число вершин $v$, число ребер $e$ и число граней $f$ границы оболочки (многогранной поверхности) связаны формулой Эйлера $v - e + f = 2$.

В случае пространств высокой размерности удобно использовать следующие понятия. Будем говорить, что точка $p$ находится *под* гипергранью $F$ политопа $P$, если $p$ лежит в открытом

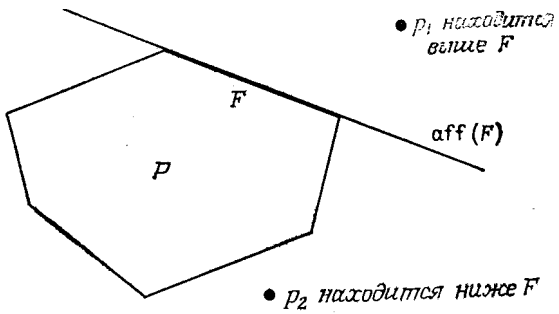



Рис. 3.24. Иллюстрация понятий «под» и «над» в двумерном случае.

полупространстве, определяемом гиперплоскостью aff($F$) и содержащем $P$. (Другими словами, aff($F$) является опорной гиперплоскостью к $P$, а $p$ и $P$ принадлежат одному и тому же полупространству, ограничиваемому этой гиперплоскостью.) Точка $p$ находится *над* $F$, если $p$ лежит в открытом полупространстве, определяемом aff($F$) и не содержащем $P$. Рис. 3.24 иллюстрирует эти понятия в двумерном случае.

### 3.4.1. Метод «заворачивания подарка»

Как упоминалось ранее (разд. 3.1.2), неизвестны разумно эффективные алгоритмы построения выпуклой оболочки конечного множества точек, которые не порождали бы полного описания границы (граф граней) выпуклой оболочки политопа. Поэтому в случае $d$-мерного пространства необходимо тщательно организовывать процедуру вычисления гиперграней выпуклой оболочки, чтобы ограничить возможные дополнительные затраты. Важным шагом в этом направлении является метод «заворачивания подарка», предложенный Чандом и Капуром [Chand, Kapur (1970)]. Этот метод был проанализирован более чем десятилетие спустя Бхаттачарья [Bhattacharya (1982)].



Основная идея метода заключается в последовательном переходе от одной гиперграни к смежной с ней гиперграни примерно так, как это происходит при заворачивании в лист бумаги объекта, ограниченного плоскими гранями. Простейшим примером этого подхода является метод обхода Джарвиса, обсуждавшийся ранее (разд. 3.3.3): в этом случае материал, используемый для заворачивания, больше похож на тесемку, чем на лист бумаги. Идея метода более естественно воспринимается в трехмерном пространстве, и мы часто будем обращаться именно к этому случаю, чтобы привлечь интуицию читателя. Однако необходимо подчеркнуть, что обсуждение на примере трехмерного пространства обусловлено причинами, связанными чисто с представлением материала, и при этом нисколько не ограничивается общность рассматриваемого подхода. Последующее обсуждение основано на предположении о том, что результирующий политоп является *симплициальным* (см. разд. 3.1). Позже мы прокомментируем ситуацию в общем случае.

Напомним, что для симплициального $d$-политопа каждая его гипергрань, являющаяся $(d-1)$-симплексом, определяется в точности $d$ вершинами. Кроме того, справедлива теорема, непосредственно следующая из сказанного.

**Теорема 3.13.** *В симплициальном политопе некоторая подграни является общей в точности для двух гиперграней и две гиперграни $F_1$ и $F_2$ имеют общую подграни $e$ тогда и только тогда, когда $e$ определяется общим подмножеством из $(d-1)$ вершины для множеств вершин, определяемых $F_1$ и $F_2$ (в этом случае $F_1$ и $F_2$ называются смежными по $e$).*

Эта теорема служит основой для рассматриваемого метода, который использует некоторую подграни $e$ уже построенной гиперграни $F_1$ для построения смежной с ней гиперграни $F_2$, имеющей с $F_1$ общую подграни $e$. В этом смысле подграни являются основными геометрическими объектами рассматриваемого метода [1]).

Пусть $S = \{p_1, p_2, \ldots, p_N\}$ — конечное множество точек в $E^d$ и предположим, что известна некоторая гипергрань $F$ выпуклой оболочки CH($S$) вместе со всеми своими подгранями. Механизм перехода от гиперграни $F$ к смежной с ней гиперграни $F'$, имеющей с $F$ общую подграни $e$, заключается в определении среди всех точек множества $S$, не являющихся вершинами $F$, такой точки $p'$, что все другие точки находятся под гиперплоскостью aff($e \cup p'$). Другими словами, среди всех гиперплоскостей, опре-

[1]) В самом деле, характеристика метода «заворачивания подарка» как «ориентированного на ребре» означает в нашей терминологии «ориентированный на подграни».

деляемых $e$ и некоторой точкой из $S$, не принадлежащей $F$, ищется гиперплоскость, образующая «наибольший угол» (в некотором смысле) с $\mathrm{aff}(F)$. Для случая $d = 3$ эта ситуация показана на рис. 3.25(а). Здесь просматривается совокупность полуплоскостей, имеющих общую прямую, проходящую по ребру $e$, и среди них ищется полуплоскость, образующая наибольший угол $<\pi$ (*выпуклый угол*) с полуплоскостью, содержащей $F$. На рис. 3.25(а) искомая полуплоскость содержит точку $p_6$.

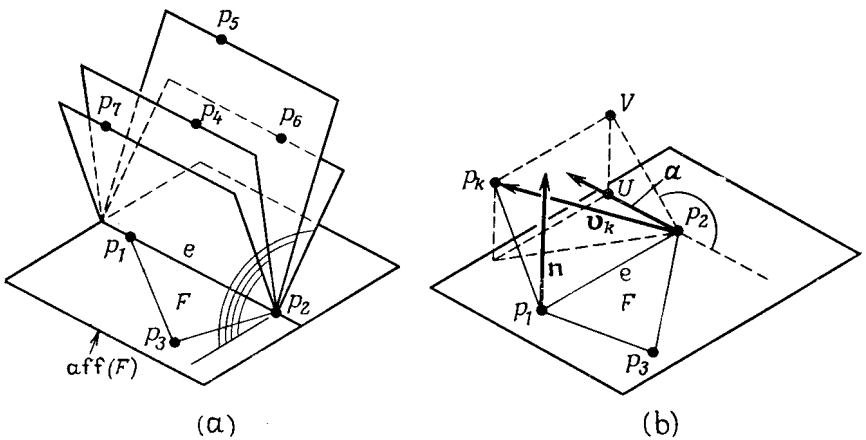



Рис. 3.25. (а) — Иллюстрация выбора полуплоскости, определяемой ребром $e$ и точкой $p_6$ и образующей наибольший выпуклый угол с полуплоскостью, содержащей $F$; (b) — иллюстрация вычисления котангенса угла.

Сравнение углов производится через сравнение котангенсов этих углов. Пусть $\mathbf{n}$ — единичная нормаль к $F$ (в полупространстве под $\mathrm{aff}(F)$), а $\mathbf{a}$ — единичный вектор, ортогональный как ребру $e$, так и вектору $\mathbf{n}$ (таким образом, ориентация вектора $\mathbf{a}$ аналогична ориентации вектора $\mathbf{n} \times \overrightarrow{p_2p_1}$). Обозначим через $\mathbf{v}_k$ вектор $\overrightarrow{p_2p_k}$. Котангенс угла, образованного гиперплоскостью, содержащей $F$, с гиперплоскостью, содержащей $e$ и $p_k$, задается отношением $|\overline{Up_2}|/|\overline{UV}|$, где $|\overline{Up_2}| = \mathbf{v}_k \cdot \mathbf{a}^T$ и $|\overline{UV}| = \mathbf{v}_k \cdot \mathbf{n}^T$. Таким образом, для каждой точки $p_k$, не принадлежащей $F$, вычисляется значение

$$\rho_k \triangleq -\mathbf{v}_k \cdot \mathbf{a}^T / \mathbf{v}_k \cdot \mathbf{n}^T \qquad (3.10)$$

и выбирается такое $\rho_i$, что

$$\rho_i = \max_k \rho_k. \qquad (3.11)$$

В случае, когда результирующий политоп является симплициальным, последнее уравнение имеет единственное решение.



Представив наглядным образом ситуацию в привычном для нас трехмерном пространстве, легко показать, что в пространстве произвольной размерности $d$ соотношения (3.10) и (3.11) дают решение нашей задачи. Вычислительную сложность этой примитивной операции можно легко оценить. Предположим, что вектор $\mathbf{n}$ известен (это единичная нормаль к $\mathrm{aff}(F)$), и будем считать, что подгрань $e$ определяется вершинами $\{p'_1, p'_2, \ldots, p'_{d-1}\}$. Следовательно, вектор $\mathbf{a}$ ортогонален вектору $\mathbf{n}$ и каждому из $(d-2)$ векторов $\overrightarrow{p'_1p'_{d-1}}$ и может быть определен в результате решения системы из $(d-2)$ уравнений с $(d-1)$ неизвестными с последующим нормированием результата, чтобы получить вектор единичной длины. Эта процедура включает $O(d^3)$ арифметических операций. Для вычисления каждого $\rho_k$ требуется $O(d)$ арифметических операций, а выбор $\rho_i$ осуществляется за $O(Nd)$ операций. Теперь можно построить единичную нормаль к новой гиперграни $F'$. Эта нормаль пропорциональна вектору $-\rho_i\mathbf{a} + \mathbf{n}$. Таким образом, полная сложность процедуры перехода от гиперграни $F$ к смежной с ней гиперграни $F'$ (*шаг «заворачивания подарка»*) равна $O(d^3) + O(Nd)$.

Поняв механизм основной операции «заворачивания подарка», можно описать организацию всего алгоритма в целом. Алгоритм начинает свою работу с некоторой исходной гиперграни (далее мы увидим, как найти ее) и для каждой ее подграни строит смежную гипергрань. Затем он переходит к новой гиперграни и продолжает подобную обработку до тех пор, пока не будут построены все гиперграни. Для каждой вновь полученной гиперграни строятся все ее подграни и при этом поддерживается список всех подграней, являющихся кандидатами на использование на шаге «заворачивания подарка». (Отметим, что некоторая подгрань $e$, общая для гиперграней $F$ и $F'$, может рассматриваться в качестве такого кандидата лишь в случае, когда либо $F$, либо $F'$, но не обе одновременно была построена в процессе «заворачивания подарка».) Упорядоченный обход всех гиперграней лучше всего организовать, использовав обычную очередь гиперграней $Q$ и файл $\mathcal{T}$, содержащий подграни.

```
      procedure ЗАВОРАЧИВАНИЕ-ПОДАРКА(p_1, ..., p_N)
1.    begin Q := ∅; T := ∅;
2.          F := найти некоторую исходную гипергрань
            выпуклой оболочки;
3.          T ⇐ подграни гиперграни F;
4.          Q ⇐ F;
5.          while(Q ≠ ∅) do
6.              begin F ⇐ Q (* выбрать первый элемент
                        из очереди *)
7.                    T := подграни гиперграни F;
```

```
8.          for each e ∈ T ∩ 𝒯 do (* e является
              кандидатом для шага "заворачивания
              подарка" *)
9.          begin F' := гиперграпь, смежная с F по
                  e (*"заворачивание подарка"*)
10.               вставить в 𝒯 все подграни F',
                  которых еще нет в файле, и
                  удалить те из них, которые там
                  уже имеются;
11.               Q ⇐ F'
            end;
12.         выдать F
        end
    end.
```

Этот алгоритм содержит несколько основных этапов, реализацию которых необходимо теперь проанализировать более подробно:

*Шаг* 2: Найти некоторую исходную гиперграпь выпуклой оболочки.

*Шаг* 7: Построить подграни гиперграни $F$.

*Шаг* 8: Проверить, является ли подграпь $e$ кандидатом на использование в последующей обработке.

*Шаг* 9 представляет собой шаг «заворачивания подарка», обсуждавшийся ранее.

Рассмотрим теперь каждую из указанных операций и оценим их сложность.

*Шаг* 2. «Найти некоторую исходную гиперграпь выпуклой оболочки». Идея состоит в том, чтобы путем последовательной аппроксимации получить гиперплоскость, содержащую гиперграпь выпуклой оболочки. То есть путем построения последовательности из $d$ опорных гиперплоскостей $\pi_1, \pi_2, \ldots, \pi_d$, каждая из которых имеет с выпуклой оболочкой на одну общую вершину больше, чем предыдущая гиперплоскость. По сути, этот метод является видоизменением механизма «заворачивания подарка», когда на $j$-й из $d$ итераций гиперплоскость $\pi_j$ содержит $(j-1)$-грань выпуклой оболочки. Таким образом, процедура начинается с определения точки с наименьшей первой координатой $(x_1)$. Назовем эту точку $p'_1$. Эта точка со всей очевидностью является вершиной (0-гранью) выпуклой оболочки. Поэтому, гиперграпь $\pi_1$ выбирается ортогональной вектору $(1, 0, \ldots, 0)$ и при этом должна проходить через точку $p'_1$. Сделав таким образом начальную итерацию, на $j$-й итерации $(j = 2, \ldots, d)$ получим гиперплоскость $\pi_{j-1}$ с нормалью $\mathbf{n}_{j-1}$, содержащую вершины $p'_1, p'_2, \ldots, p'_{j-1}$. Вектор $\mathbf{a}_j$ выбирается ортогональным к нормали $\mathbf{n}_{j-1}$, каждому из $(j-2)$ векторов $\overrightarrow{p'_1p'_2}, \overrightarrow{p'_1p'_3}, \ldots, \overrightarrow{p'_1p'_{j-1}}$ и каждой из координатных осей $x_{j+1}, x_{j+2}, \ldots, x_d$. Эти $(d-1)$ условий определяют вектор $\mathbf{a}_j$, и, следовательно, теперь можно выполнить основную операцию по «заворачиванию», использовав для этого векторы $\mathbf{a}_j$ и $\mathbf{n}_{j-1}$. На рис. 3.26 показана последовательность гиперплоскостей для случая $d = 3$. Вычислительные затраты на каждой итерации связаны главным образом с построением вектора $\mathbf{a}_j$ и выбором очередной точки $p'_j$. Последняя операция имеет сложность $O(Nd)$, в то время как для определения вектора $\mathbf{a}_j$ требуется лишь время $O(d^2)$, так как $\mathbf{a}_j$ получается в результате добавления в точности одного линейного ограничения к ограничениям, определяющим вектор $\mathbf{a}_{j-1}$. Таким образом, поскольку выполняется $d$ итераций, построение исходной грани завершается за время $O(Nd^2) + O(d^3) = O(Nd^2)$, учитывая при этом, что $N \geqslant d$.



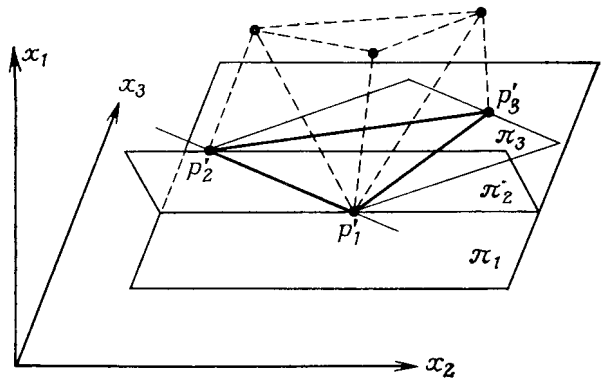



Рис. 3.26. Последовательность гиперплоскостей $(\pi_1, \pi_2, \pi_3)$ такова, что $\pi_3$ содержит гиперграпь, которая берется в качестве исходной для процедуры построения выпуклой оболочки.

*Шаг* 7. «Построить подграни гиперграни $F$». В силу сделанного предположения о симплициональности политопа каждая гиперграпь определяется в точности $d$ вершинами, а каждое подмножество, содержащее $(d-1)$ вершин гиперграни $F$, определяет подграпь. Таким образом, можно за время $O(d^2)$ очевидным способом построить подграни гиперграни $F$. Каждая гиперграпь будет описываться $d$-компонентным вектором индексов ее вершин, а подграпь — аналогичным $(d-1)$-компонентным вектором.

*Шаг* 8. «Проверить, является ли подграпь $e$ кандидатом на использование в последующей обработке». Подграпь является

кандидатом, если она содержится лишь в одной грани, порождаемой алгоритмом, и, следовательно, если поддерживается файл $\mathcal{T}$ таких подграней, то требуемым тестом является просмотр этого файла. Как отмечалось выше, подгрань описывается $(d-1)$-компонентным вектором из целых чисел (индексов определяющих ее вершин). Можно упорядочить файл $\mathcal{T}$ в соответствии с лексикографическим порядком и хранить его в виде сбалансированного по высоте бинарного дерева поиска. Если имеется некоторая подгрань $e$, то потребность в доступе к $\mathcal{T}$ может возникнуть в двух случаях: либо для проверки принадлежности $e$ к числу кандидатов (шаг 8 алгоритма), либо для изменения файла (шаг 10 алгоритма). В первом случае просматривается файл $\mathcal{T}$ и определяется, содержит ли он $e$. Во втором случае также происходит просмотр файла $\mathcal{T}$ с целью определения принадлежности $e$ файлу, но, кроме того, если $e$ уже содержится в файле, то она удаляется, в противном случае она добавляется в файл. В обоих случаях просмотр файла осуществляется за время $O(d \log M)$, где $M$ — размер файла, ограниченный сверху величиной $\mathrm{SF}(d, N)$, равной максимальному числу подграней [1]).

Теперь можно оценить полную сложность алгоритма. Сложность начального этапа работы алгоритма (шаги 1—4) составляет $O(Nd^2)$. Шаги 6, 11 и 12 (добавление в очередь и удаление из очереди или вывод гиперграни) можно рассматривать совместно. Если обозначить через $\varphi_{d-1}$ и $\varphi_{d-2}$ действительные числа соответственно гиперграней и подграней политопа, то их полная сложность равна $O(d) \times \varphi_{d-1}$. Полная сложность шага 7 — порождение всех подграней — равна $O(d) \times \varphi_{d-2}$, так как каждая подгрань строится за время $O(d)$ и порождается дважды. Проверка, выполняемая на шаге 8, так же как и изменение файла на шаге 10, выполняется за время $O(d \cdot \log \varphi_{d-2})$ на каждую подгрань, так что полная сложность составляет $O(d \log \varphi_{d-2}) \times \varphi_{d-2}$. Наконец, полная сложность шага 9 — шаг заворачивания — равна $\varphi_{d-1} \times (O(d^3) + O(Nd))$. Если вспомнить, что максимальное значение как для $\varphi_{d-1}$, так и для $\varphi_{d-2}$ равно $O\left(N^{\lfloor d/2 \rfloor}\right)$, то получим следующий результат:

**Теорема 3.14.** *Выпуклая оболочка множества из $N$ точек в $d$-мерном пространстве может быть построена методом «заворачивания подарка» за время $T(d, N) = O(N \cdot \varphi_{d-1}) + O(\varphi_{d-2} \times \log \varphi_{d-2})$. Таким образом, в худшем случае имеем $T(d, N) = O\left(N^{\lfloor d/2 \rfloor + 1}\right) + O\left(N^{\lfloor d/2 \rfloor} \log N\right)$.*

В заключение отметим, что нельзя игнорировать тот факт, что рассмотренный выше метод применим в предположении

[1] $\mathrm{SF}(d, N) = O^{1)}\left(N^{\lfloor d/2 \rfloor}\right)$ [Grünbaum (1967), 9.6.1].



симплициальности результирующего политопа. Что происходит в общем, хотя практически невероятном случае, когда гиперграни содержат более $d$ вершин? Если гипергрань $F$ не является симплексом, то уравнение (3.10) имеет более одного решения, т. е. в множестве $S$ имеются несколько точек, реализующих это условие. Все эти точки принадлежат гиперплоскости $\mathrm{aff}(F)$. Однако для того, чтобы определить состав граней, входящих в $F$ (т. е. $(d-2)$-грани, ..., 0-грани), необходимо в свою очередь (рекурсивно) решать $(d-1)$-мерную задачу построения выпуклой оболочки в $\mathrm{aff}(F)$. Таким образом, процедура порождения подграней, не вызывавшая затруднений в случае симплициального политопа, становится значительно более сложной. А рекурсивное определение алгоритма для общего случая, естественно, существенно затрудняет его анализ.

### 3.4.2. Метод «под-над»

В течение длительного времени метод «заворачивания подарка» оставался единственным из известных общих методов построения выпуклой оболочки конечного множества точек в $d$-мерном пространстве. Новый метод, названный «под-над», был предложен Кейли [Kallay (1981)]. Этот алгоритм обладает также свойством открытости.

Основная идея метода существенно отличается от используемой в методе «заворачивания подарка» и довольно близка идее, используемой в алгоритме для динамической двумерной выпуклой оболочки, описанном в разд. 3.3.6. На содержательном уровне этот метод последовательно обрабатывает по одной точке, назовем ее $p$, и если $p$ — внешняя точка по отношению к текущей оболочке $P$, то из точки $p$ строится опорный «конус» к $P$ и удаляется часть оболочки $P$, затеняемая этим конусом. (Использование терминов «конус» и «затенение» происходит по аналогии с трехмерным случаем.) Рассматриваемый метод основан на следующей теореме [McMullen, Shephard (1971), p. 113]:

**Теорема 3.15.** *Пусть $P$ — политоп, а $p$ — точка в пространстве $E^d$. Введем обозначение $P' = \mathrm{CH}(P \cup p)$. Тогда для каждой грани $P'$ справедливо одно из двух утверждений:*

(1) *грань $f$ политопа $P$ является также гранью $P'$ тогда и только тогда, когда существует гипергрань $F$ политопа $P$ такая, что $f \subset F$ и $p$ находится под $F$;*

(2) *если $f$ — грань политопа $P$, то $f' = \mathrm{CH}(f \cup p)$ является гранью $P'$ тогда и только тогда, когда либо*

(a) *среди гиперграней политопа $P$, содержащих $f$, имеется по крайней мере одна такая, что $p$ находится под ней, и по крайней мере одна такая, что $p$ находится над ней, либо*

(b) $p \in \mathrm{aff}(f)$.

Интуитивно понятно, что случай (1) относится к гиперграням политопа $P$, незатеняемым конусом и которые становятся гипергранями $P'$, представляющего измененную выпуклую оболочку. (См. рис. 3.27, на котором показаны все случаи на примере простого, но вместе с тем не теряющего общности двумерного пространства.) Случай (2) относится к гиперграням конуса. В частности, пункт (а) случая (2) определяет грани, на которые опирается конус, а пункт (b) определяет вырожденный

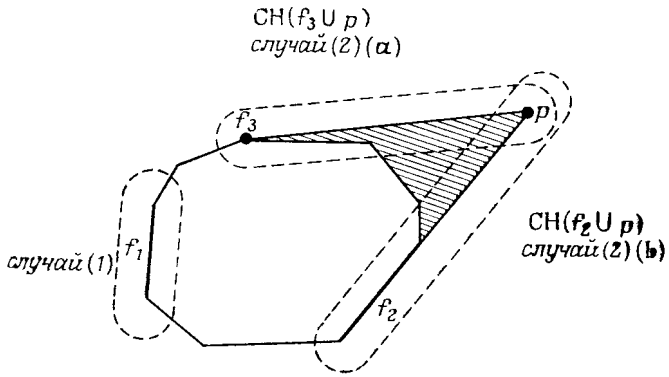



Рис. 3.27. Иллюстрация случаев, рассматриваемых в теореме 3.15.

вариант ситуации, определяемой в пункте (а). Заметим, что в п. (а) случая (2), размерность грани $f' = \mathrm{CH}(f \cup p)$ оболочки $P'$ на единицу больше размерности грани $f$ политопа $P$. И хотя в общем случае целью является построение лишь гиперграней выпуклой оболочки, эффект увеличения размерности, имеющий место в п. (а) случая (2), вынуждает иметь необходимое описание всех граней текущего политопа.

Так как алгоритм является открытым, т. е. выпуклая оболочка строится путем последовательного добавления по одной точке, то потребуется по крайней мере $(d+1)$ итераций, прежде чем текущий политоп будет иметь размерность $d$. Это значит, что сначала получается 0-грань, затем 1-грань и т. д. Однако механизм перестройки выпуклой оболочки остается тем же самым, и далее он будет описан в общем случае. Для простоты предполагается, что точки находятся в общем положении, так что вырождение не имеет места.

Адекватное описание текущего политопа $P$ дает его граф граней $H(P)$ (т. е. диаграмма Хассе для множества граней политопа $P$, основанная на отношении включения; см. разд. 3.1), из которого удален узел, соответствующий политопу $P$ в целом, так как он является избыточным при размерности $P$, равной $d$. В качестве структуры данных можно было бы использовать



$(d+1)$ списков $L_{-1}, L_0, L_1, \ldots, L_{d-1}$, где $L_j$ — список $j$-граней. Каждая запись в $L_j$ относится к некоторой $j$-грани $f$ и содержит:

(1) аффинный базис БАЗИС($f$) грани $f$;

(2) указатели СУПЕРГРАНЬ($f$) на каждую $(j+1)$-грань, содержащую $f$, и указатели ПОДГРАНЬ($f$) на каждую $(j-1)$-грань, содержащуюся в $f$;

(3) указатели ГИПЕРГРАНЬ($f$) на каждую гиперграни, содержащую $f$.

Указатели (3) играют основную роль при определении относительного положения точки (п. (а) случая (2) теоремы 3.15). Остальные данные нужны для получения необходимой информации о том, следует ли повышать размерность грани, чтобы в конечном итоге превратить ее в гиперграни.

Изменение политопа $P$, вызванное добавлением новой точки, можно осуществить не единственным способом. Хотя предлагаемый ниже подход допускает некоторые улучшения на уровне алгоритма, зато он довольно прост для описания.

Пусть политоп $P$ представлен своим графом граней $H(P)$. Создадим другой граф $H_p(P)$, изоморфный $H(P)$, узлы которого определяются следующим образом:

$$\{f' : f' = \mathrm{CH}(f \cup p) \text{ для каждой грани } f \text{ политопа } P\},$$

и, кроме того, для каждой грани $f$ политопа $P$ создадим ребро, отображающее включение $f \subset f'$ (рис. 3.28(b)). Все, что требуется сделать при создании $f'$, это добавить к БАЗИС($f$) точку $p$ так как по предположению точки находятся в общем положении (что гарантирует выполнение условия $p \notin \mathrm{aff}(f)$). Обозначим полученный таким образом граф $H(P, p)$.

Очевидно, что $H_p(P)$ содержит опорный «конус» к $P$ из точки $p$. Действительно, когда размерность политопа $P$ меньше $d$, то $H_p(P)$ представляет в точности опорный конус, и изменение текущего политопа на этом заканчивается с $H(P, p) = H(\mathrm{CH}(P \cup p))$.

Рассмотрим механизм корректировки в случае, когда размерность политопа $P$ равна $d$. В предположении, что точка $p$ лежит вне $P$, требуемый граф $H(\mathrm{CH}(P \cup p))$ получается из $H(P, p)$ в результате удаления некоторых узлов графа. А именно имеет место следующее:

(1) $H(P)$ строго содержит часть, затеняемую конусом. В самом деле, каждая грань $f$ политопа $P$ такая, что для каждой гиперграни $F$, содержащей $f$, точка $p$ находится над $F$, принадлежит тени и поэтому должна быть удалена (в примере на рис. 3.28 это имеет место для $p_3$, $e_{34}$ и $e_{23}$). Воспользуемся следующим очевидным свойством:

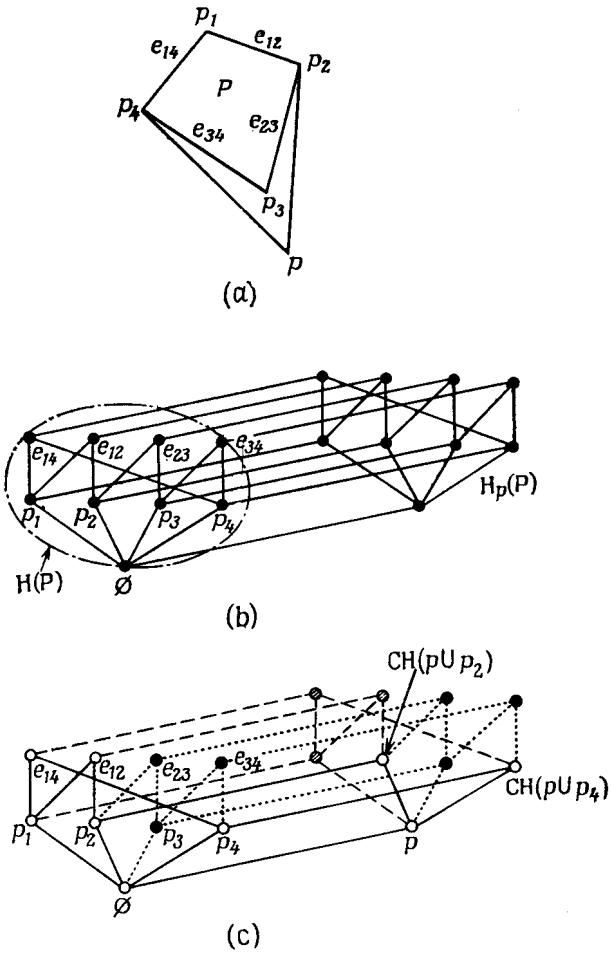



Рис. 3.28. Иллюстрация перестройки политопа $P$; (a) — геометрия ситуации, возникающей при перестройке; (b) — создание $H_p(P)$ путем «переноса» графа $H(P)$; (c) — $H(\mathrm{CH}(P \cup p))$ получается в результате удаления из графа ряда узлов.

**Свойство 1.** *Если* $f \notin \mathrm{CH}(R \cup p)$ *и* $f \subset f'$, *то* $f' \notin \mathrm{CH}(P \cup p)$.

Использование свойства 1 позволяет удалить в графе $H(P, p)$ все узлы, доминирующие над $f$. (Грани, удаляемые по этой причине, показаны на рис. 3.28(c) черными кружками.) Для каждой удаляемой грани $f$ удаляются также инцидентные ей ребра (т. е. как СУПЕРГРАНЬ($f$), так и ПОДГРАНЬ($f$)).

(2) $H_p(P)$ содержит опорный конус. В самом деле, каждая грань $f$ политопа $P$ такая, что для каждой гиперграни $F$, содержащей $f$, точка $p$ находится под $F$, также является гранью $\mathrm{CH}(P \cup p)$ (случай (1) теоремы 3.15), так что $\mathrm{CH}(f \cup p) \in H_p(P)$ должна быть удалена (в нашем примере это верно для $f = p_1, e_{14}, e_{12}$). Воспользуемся следующим очевидным свойством:



**Свойство 2.** *Если* $\mathrm{CH}(f \cup p) \notin \mathrm{CH}(P \cup p)$ *и* $f \subset f'$, *то* $\mathrm{CH}(f' \cup p) \notin \mathrm{CH}(P \cup p)$.

Использование свойства 2 позволяет удалять в графе $H_p(P)$ все узлы, доминирующие над $\mathrm{CH}(f \cup p)$. (Грани, удаляемые по этой причине, показаны на рис. 3.28(c) заштрихованными кружками.) Затем для каждого узла графа $H_p(P)$, удаленного таким образом, удаляются инцидентные ему ребра. Этот результирующий граф, полученный путем удаления из $H(P, p)$ некоторых узлов, является графом граней политопа $\mathrm{CH}(P \cup p)$. Заметим также, что, когда точка $p$ не является внешней по отношению к $P$, свойство 2 остается справедливым для каждой грани $f$ политопа $P$. Так что в этом случае удаляется целиком подграф $H_p(P)$ и при этом получается правильный результат $\mathrm{CH}(P \cup p) = \mathrm{CH}(P)$.

С точки зрения реализации основной частью этого метода является процедура классификации каждой из $N$ вершин политопа $P$ относительно точки $p$. При этом вершина может быть классифицирована как вогнутая, выпуклая или опорная (в соответствии с обобщением терминологии из разд. 3.3.6). В частности,

вершина $v$ относительно точки $p$ является
- *вогнутой*, если для каждой гиперграни $F$, содержащей $v$, точка $p$ находится под $F$;
- *выпуклой*, если для каждой гиперграни $F$, содержащей $v$, точка $p$ находится над $F$;
- *опорной* во всех других случаях.

В качестве интересного упражнения читателю предлагается самому придумать метод, выполняющий указанную выше классификацию вершин за время $O(\varphi_{d-1}) + O(Nd)$, где, как обычно, $\varphi_{d-1}$ — число гиперграней политопа $P$ (упражнение 3.9 в конце главы). Если такая классификация сделана, то свойство 1 применимо ко всем граням, доминирующим над любой из выпуклых вершин, а свойство 2 — ко всем граням, доминирующим над вогнутыми вершинами в $H_p(P)$, и эта операция может быть выполнена за время, пропорциональное полному числу вершин и ребер $H(P)$. Нетрудно показать, что последняя величина равна $O(\varphi_{d-1})$. Таким образом, полное время одной операции корректировки выпуклой оболочки равно $O(\varphi_{d-1})$, а так как выполняется $N$ таких операций, то получаем следующую теорему:

**Теорема 3.16.** *Выпуклая оболочка множества из $N$ точек в $d$-мерном пространстве может быть построена методом «под-над», являющимся открытым, за время $T(d, N) = O(N^{\lfloor d/2 \rfloor + 1})$.*

Сделаем два замечания. Первое, в рассмотренном методе необходимо предусмотреть специальную обработку вырожденных случаев, когда точка $p$ принадлежит аффинной оболочке некоторой грани политопа $P$. Второе, можно показать, что в случае, когда политоп $P$ не является симплициальным, можно слегка изменить $P$, не увеличив существенным образом сложность структуры его граней или сложность отношения включения. И как следствие, временная оценка, полученная для симплициального случая, остается применимой и в общем случае.

В заключение отметим, что метод «под-над» весьма привлекателен, так как его сложность сравнима со сложностью метода «заворачивания подарка» и, кроме того, он является открытым, что тоже очень важно.

### 3.4.3. Выпуклые оболочки в трехмерном пространстве

Для задачи построения выпуклой оболочки в многомерных пространствах случай трехмерного пространства имеет огромную важность, так как он естественным образом возникает во множестве прикладных областей, включающих машинную графику, автоматизацию проектирования, распознавание образов, исследование операций. К счастью, этот важный для приложений случай одновременно является «изюминкой» с точки зрения сложности вычислений. Далее мы попытаемся пояснить эту несколько туманную мысль.

Как уже отмечалось ранее, результатом работы алгоритмов построения выпуклой оболочки должно быть описание политопа, представляющего выпуклую оболочку. Из разд. 3.1 известно, что тривиальная нижняя оценка сложности в худшем случае для любого алгоритма построения выпуклой оболочки $N$ точек в пространстве $E^d$, основанная на размере результирующего описания, равна $\Omega(N^{\lfloor d/2 \rfloor})$. В предыдущем разделе было показано также, что оценка для метода «под-над» превышает эту нижнюю оценку на множитель $O(N)$, и для $d = 3$ ($d$ в этом случае нечетное число) это лучшее, что можно ожидать для открытого метода. Если отказаться от свойства открытости, то можно рассчитывать, что удастся уменьшить этот множитель, получив вместо $O(N)$ что-нибудь, подобное $O(\log N)$. В трехмерном случае это дало бы оценку $O(N^{\lfloor 3/2 \rfloor} \log N) = O(N \log N)$ С другой стороны, нижняя оценка в общем случае в точности равна $\Omega(N \log N)$ (см. разд. 3.2), поэтому лучшее, на что можно



надеяться, это получить алгоритм со сложностью $\theta(N \log N)$. И эта цель достижима [Preparata, Hong (1977)].

Как обычно, пусть имеется множество $S = \{p_1, p_2, \ldots, p_N\}$ из $N$ точек в пространстве $E^3$. Для простоты предположим, что для любой пары точек $p_i$ и $p_j$ из $S$ имеет место $x_k(p_i) \neq x_k(p_j)$, $k = 1, 2, 3$. Это упрощение поможет яснее показать основные идеи алгоритма, а изменения, которые необходимо сделать в алгоритме для общего случая, очевидны. Здесь мы попытаемся разработать алгоритм типа «разделяй и властвуй» со сложностью $O(N \log N)$.

Для начала упорядочим множество $S$ по координате $x_1$ и, если это необходимо, перенумеруем точки так, чтобы $x_1(p_i) < x_1(p_j) \Leftrightarrow i < j$. Существует следующий простой рекурсивный алгоритм построения выпуклой оболочки (представленный в виде функции):

```
            function CH(S)
1.  begin if (|S| ≤ k0) then
            begin построить CH(S) методом "грубой силы";
                  return CH(S)
            end
2.        else begin S1 := {p1, ..., p⌊N/2⌋};
                     S2 := {p⌊N/2⌋+1, ..., pN};
3.                   P1 := CH(S1); P2 := CH(S2);
4.                   P := СЛИТЬ(P1, P2);
5.                   return P
                end
    end.
```

Предварительная сортировка элементов множества $S$ по координате $x_1$ требует $O(N \log N)$ операций. Отметим, что благодаря такой сортировке и тому, как выполняется шаг 2 алгоритма, множества $\mathrm{CH}(S_1)$ и $\mathrm{CH}(S_2)$ представляют два непересекающихся трехмерных политопа. Если «слияние», т. е. построение выпуклой оболочки объединения двух выпуклых оболочек с суммарным числом вершин, равным $N$, может быть выполнено не более чем за $M(N)$ операций, то верхняя оценка для $T(N)$ — числа операций, выполняемых алгоритмом CH, определяется уравнением $T(N) = 2T(N/2) + M(N)$. (Отметим, что для простоты мы считаем $N$ четным, но это не влияет на общность результата.) Таким образом, если мы сумеем показать, что $M(N)$ равно $O(N)$, то получим для $T(N)$ оценку $O(N \log N)$ и, учитывая затраты на предварительную сортировку, получим оценку $O(N \log N)$ для сложности всего алгоритма построения выпуклой оболочки.

Ясно, что функция СЛИТЬ является критическим местом рассматриваемого метода. Начнем с обсуждения структуры дан-

ных, подходящей для представления выпуклого трехмерного политопа $P$. Необходимо описать лишь границу политопа $P$, которая в трехмерном случае топологически эквивалентна плоскому графу. Как отмечалось в разд. 1.2.3.2, естественной структурой для представления плоского графа является рёберный список с двойными связями (РСДС). РСДС обладает интересным свойством — явным образом он содержит только ребра графа, но эта структура позволяет за оптимальное время выделять вершины и множества вершин граней [1]).

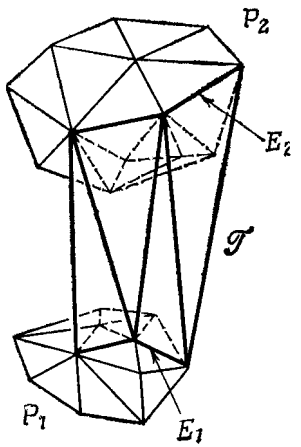



Рис. 3.29. Иллюстрация основной идеи метода. Триангуляция $\mathcal{T}$ образуется в результате «заворачивания» выпуклых оболочек $P_1$ и $P_2$.

Предположим теперь, что рекурсивным образом получены политопы $P_1 = \mathrm{CH}(S_1)$ и $P_2 = \mathrm{CH}(S_2)$. Важно подчеркнуть, что вследствие предварительной сортировки и выбранного способа разбиения результирующего множества (строка 2 в алгоритме $\mathrm{CH}(S)$) политопы $P_1$ и $P_2$ не пересекаются. Слияние $P_1$ и $P_2$ можно выполнить с помощью следующих операций (рис. 3.29 иллюстрирует идею этих операций):

1. Построить «цилиндрическую» триангуляцию $\mathcal{T}$, опирающуюся на $P_1$ и $P_2$ по некоторым контурам $E_1$ и $E_2$ соответственно [2]).

2. Удалить из $P_1$ и $P_2$ части, оказавшиеся «скрытыми» в результате построения триангуляции $\mathcal{T}$.

Здесь использованы не определенные формально термины «цилиндрическая» и «скрытые» в расчете на их интуитивное понимание в соответствии с рис. 3.29. Оставаясь на интуитивном уровне, построение триангуляции можно рассматривать как операцию «заворачивания подарка»: с помощью $\mathcal{T}$ мы как бы заворачиваем в один «сверток» одновременно и $P_1$, и $P_2$. Хотя $\mathcal{T}$ может иметь $O(N)$ гиперграней, а каждый шаг заворачивания в общем случае требует $O(N)$ операций, использование особенностей трехмерных политопов позволяет, как будет показано далее, сделать существенное упрощение.

[1]) Здесь термин «грань» используется в общепринятом для плоских графов смысле. Однако в оставшейся части этого раздела будет использоваться термин «гиперграннь», чтобы подчеркнуть, что они являются $(d-1)$-мерными множествами.

[2]) Мы неявно предполагаем, что на каждом этапе обработки мы имеем симплициальные (триангулированные) политопы. Как будет видно из дальнейшего, это несущественно влияет на общность результата.



Построение триангуляции $\mathcal{T}$ начинается с определения некоторой ее гиперграни или ее ребра. Хотя имеются несколько способов решения этой задачи, довольно удобный способ основывается на построении проекций (например, на плоскость $(x_1, x_2)$), являющихся многоугольниками, каждого из политопов (рис. 3.30). Обозначим через $P_i'$ проекцию $P_i$ на плоскость $(x_1, x_2)$ $(i = 1, 2)$. Теперь можно воспользоваться нашим любимым алгоритмом (см. разд. 3.3.5), чтобы построить общую опорную прямую к $P_1'$ и $P_2'$ и получить тем самым на плоскости $(x_1, x_2)$ ребро $e'$, являющееся проекцией ребра $e$ триангуляции $\mathcal{T}$. Имея ребро $e$, можно начать построение $\mathcal{T}$, выбрав в качестве опорной плоскость, проходящую через ребро $e$ параллельно оси $x_3$.

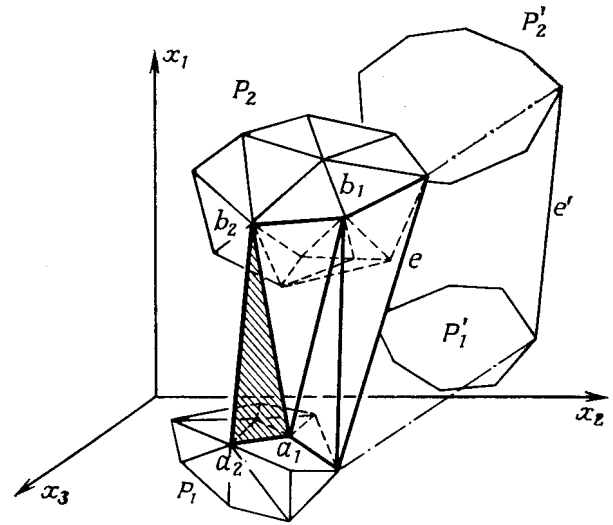



Рис. 3.30. Начальный и последующие шаги процедуры построения $\mathcal{T}$.

На очередном шаге построения $\mathcal{T}$ в качестве базы используется последняя построенная гиперграннь триангуляции $\mathcal{T}$. Будем использовать при обозначении вершин политопа $P_1$ букву $a$, а для $P_2$ — букву $b$. Пусть гиперграннь $(a_2, b_2, a_1)$ является базовой на текущем шаге (на рис. 3.30 она заштрихована). Теперь среди вершин, смежных с $a_2$, необходимо выбрать вершину $\hat{a}$, так чтобы гиперграннь $(a_2, b_2, \hat{a})$ образовала наибольший выпуклый угол с $(a_2, b_2, a_1)$ среди всех гиперграней $(a_2, b_2, v)$, где $v$ — вершина, смежная с $a_2$, и $v \neq a_1$. Аналогичным образом среди вершин, смежных с $b_2$, выберем вершину $\hat{b}$. По причинам,

которые станут понятными в дальнейшем, эти сравнения называются сравнениями типа 1.

Теперь, когда выявлены два «претендента» — $(a_2, b_2, \hat{a})$ и $(a_2, b_2, \hat{b})$, делается заключительное сравнение, называемое сравнением типа 2. Если $(a_2, b_2, \hat{a})$ образует с $(a_2, b_2, a_1)$ больший выпуклый угол, чем $(a_2, b_2, \hat{b})$, то $\hat{a}$ добавляется к $\mathscr{T}$ (в противном случае добавляется $\hat{b}$), и на этом очередной шаг заканчивается. Описанный порядок выполнения сравнений является, как будет видно далее, эффективным способом реализации шага «заворачивания подарка» вокруг ребра $(a_2, b_2)$.

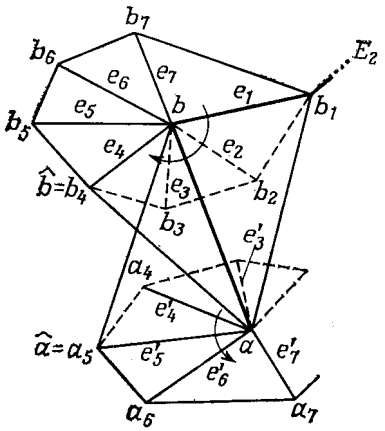



Рис. 3.31. Шаг, обеспечивающий продвижение при построении $\mathscr{T}$.

Эффективная реализация описанного только что механизма последовательного построения граней триангуляции основывается на следующих соображениях. В РСДС можно эффективно проходить ребра, инцидентные некоторой вершине, в порядке обхода по часовой стрелке или против нее. Для большей конкретности обратимся к рис. 3.31. Предположим, что вершина $b$ и ребро $(b, a)$ триангуляции $\mathscr{T}$ были добавлены последними. Пусть $(b_1, b)$ — ребро контура $E_2$, входящее в вершину $b$. Не теряя общности, можно предположить, что нумерация ребер, инцидентных вершине $b$, и их конечных вершин $b_1, b_2, \ldots, b_k$ аналогична показанной на рис. 3.31 (где $k = 7$). Подобное же предположение имеет место и относительно вершины $a$. Пусть $(b_s, b, a)$ — гиперграни, образующая наибольший выпуклый угол с $(b_1, b, a)$ из всех $(b_i, b, a)$ с $i = 2, \ldots, k$ (в нашем примере $s = 4$). Первое важное соображение состоит в том, что любое ребро $(b, b_i)$ при $1 < i < s$ *оказывается внутри выпуклой оболочки* $\mathrm{CH}(P_1 \cup P_2)$, *и поэтому его можно исключить из дальнейшего рассмотрения.* Что касается вершины $a$, то заметим, что множество инцидентных ей ребер было частично просмотрено на более раннем этапе работы алгоритма, так как вершина $a$ была получена прежде, чем вершина $b$. И как следствие сказанного выше просмотр ребер, инцидентных вершине $a$, следует начинать с последнего из просмотренных ребер, так как все другие были удалены (на рис. 3.31 это ребро $(a, a_4)$).

Следующее соображение связано с большим количеством сравнений углов, каждое из которых должно выполняться за постоянное время путем адаптации общего метода, описанного



в разд. 3.4.1 для случая $d$-мерного пространства. В результате сравнения типа 1 из дальнейшего рассмотрения обязательно исключается одно ребро, принадлежащее либо $P_1$, либо $P_2$. Пусть $v_i$ и $m_i$ обозначают соответственно число вершин и число ребер $P_i$ $(i = 1, 2)$. Таким образом, число сравнений типа 1 ограничено сверху величиной $(m_1 + m_2)$. Далее каждое сравнение типа 2 добавляет новую вершину либо к $E_1$, либо к $E_2$. Так как число вершин в контурах $E_1$ и $E_2$ не может превышать соответственно $v_1$ и $v_2$, то число сравнений типа 2 ограничено величиной $(v_1 + v_2 - 1)$. *Поскольку граф, представляющий структуру поверхности политопа, является планарным,* $m_i \leqslant 3v_i - 6$, и, следовательно, число сравнений углов растет не быстрее линейной функции от суммарного числа вершин политопов $P_1$ и $P_2$.

Корректировку РСДС можно производить динамически в процессе построения $\mathscr{T}$. Воспользуемся иллюстрацией на рис. 3.31. Предположим, мы продвинулись от ребра $(a, b_1)$ к ребру $(a, b)$, «поворачиваясь» вокруг вершины $a$. Произойдет просмотр ребер, инцидентных как вершине $b$, так и вершине $a$, хотя ребра, инцидентные $a$, уже были частично просмотрены ранее. Сначала в РСДС создается новый узел для ребра $(a, b)$ и ребро $(a, b)$ становится следующим за $(a, b_1)$ вокруг $a$, а $(b_1, b)$ становится следующим за ним вокруг $b_1$. Теперь начинается просмотр ребер, инцидентных $b$, в порядке движения по часовой стрелке вокруг $b$, начиная с ребра $e_1$ и кончая $e_4$: узлы, соответствующие ребрам $e_2$ и $e_3$, были удалены [1]) (так как $e_2$ и $e_3$ оказались внутри оболочки), а ребро $(a, b)$ становится следующим за ребром $e_1$. Аналогичный просмотр осуществляется вокруг вершины $a$, начиная с некоторого ребра $e'_4$ до некоторого другого ребра $e' = (a, a_5)$. Затем, если в результате сравнения углов выбирается гиперграни $(a, b, b_4)$, то ребра $(a, b_4)$ и $(b, b_4)$ становятся следующими за ребром $(a, b)$ вокруг вершин $a$ и $b$ соответственно. В противном случае вместо них берутся $(a, a_5)$ и $(b, a_5)$. Это приводит к корректировке РСДС как связанной структуры данных. Отсюда следует, что полное время, требуемое алгоритму СЛИТЬ, равно $O(N)$, и поэтому имеет место следующая теорема.

**Теорема 3.17.** *Выпуклая оболочка множества из $N$ точек в трехмерном пространстве может быть построена за оптимальное время* $\theta(N \log N)$.

[1]) Заметим, что при такой процедуре удаления ребер из РСДС не все ребра, попавшие внутрь выпуклой оболочки, могут быть удалены. Однако неудаленные (таким образом) ребра образуют планарный граф с множеством вершин $V'$, которые сами являются внутренними точками выпуклой оболочки. Поэтому, число неудаленных ребер пропорционально $|V'|$, и полный объем памяти, занимаемой РСДС, остается пропорциональным $|S|$. Отметим, что граф с множеством вершин $V'$ не связан с графом выпуклой оболочки.

В заключение необходимо обсудить, как быть с вырожденными ситуациями, когда политопы не является симплициальными. Самое разумное решение основывается на введении «искусственной» триангуляции гиперграней, не являющихся треугольниками, и тогда политоп все время остается симплициальным. Рассмотрим, в частности, процедуру просмотра множества ребер, инцидентных вершине $b$ (рис. 3.32). В предположении симплициальности политопа просмотр завершается на ребре $e$, принадлежащем двум гиперграням $F$ и $F'$, так что при этом вершина $a$ находится над $F$ и под $F'$ (относительно смысла терминов «над» и «под» см. начало разд. 3.4). Это единственно возможная ситуация, так как в соответствии с предположением никакие четыре точки не могут лежать в одной плоскости. При отказе от этого предположения критерий завершения просмотра ребер становится следующим: просмотр заканчивается на ребре $e$, принадлежащем гиперграням $F$ и $F'$ таким, что вершина $a$ находится не под $F$ и над $F'$. В таком случае, если вершина $a$ и гипергрань $F$ принадлежат одной гиперплоскости, то результирующая гипергрань $\mathrm{CH}(F \cup a)$ оказывается разбитой на треугольники пучком ребер, выходящих из вершины $a$ (рис. 3.32). Таким образом, грани $\mathscr{T}$ будут треугольниками, хотя смежные гиперграни могут лежать в одной гиперплоскости. Лишние ребра легко могут быть удалены за один проход по РСДС по окончании построения выпуклой оболочки. При прохождении РСДС отмечается каждое ребро, компланарное с двумя своими соседями, инцидентными одной и той же вершине, а затем все отмеченные ребра удаляются.

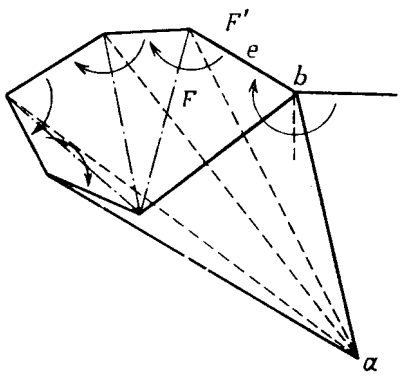



Рис. 3.32. Пример триангуляции гиперграни, не являющейся треугольником.

## 3.5. Замечания и комментарии

В этой главе мы неоднократно указывали на тесную связь задач сортировки и построения выпуклой оболочки на плоскости. В частности, любой алгоритм, строящий упорядоченную последовательность вершин выпуклой оболочки, т. е. решающий задачу ВО1 в случае плоскости независимо от реализуемого им метода, является замаскированным алгоритмом



сортировки. При каждом подходящем случае мы иллюстрировали эту аналогию, делая соответствующие примечания. В этом разделе мы воспользуемся удобным случаем, чтобы дать более систематический обзор алгоритмов, представленных в этой главе, раскрывая соответствующие им методы сортировки.

Уместно сделать следующее замечание. Если смотреть более строго, то аналогия будет довольно полной, когда все заданные точки являются вершинами выпуклой оболочки. Это вполне понятно, так как в задаче сортировки нет аналога для внутренних точек выпуклой оболочки.

В алгоритме Грэхема сортировка используется явным образом в самом начале работы. Первый алгоритм из разд. 3.3.1, проверяющий каждую пару точек для определения, образуют они или нет ребро выпуклой оболочки (и настолько не эффективный, что мы даже не удостоили его собственного имени), аналогичен следующему необычному алгоритму сортировки:

1. Для того чтобы отсортировать числа $x_1, \ldots, x_N$, нужно сначала найти наименьшее по величине $x_m$ и поменять местами $x_1$ и $x_m$. (Требуемое время — $O(N)$.)
2. Теперь нужно найти число, которое в окончательно отсортированном списке будет следовать за $x_1$, просматривая возможных кандидатов $x_2, \ldots, x_N$. Число $x_j$ следует за $x_1$ тогда и только тогда, когда между ними нет других элементов списка. (Требуемое время — $O(N^2)$.)
3. Аналогичным образом найти элементы, следующие за $x_2, \ldots, x_{N-1}$. (Время, требуемое для нахождения каждого последующего элемента, — $O(N^2)$. Полное время — $O(N^3)$.)

Эта процедура может быть очевидным образом улучшена за счет ускорения процесса поиска последующего элемента. Для того чтобы определить элемент, следующий за $x_i$, необходимо лишь найти наименьший элемент среди $x_{i+1}, \ldots, x_N$. После такого изменения приведенный выше алгоритм превращается в СОРТИРОВКУ ПОСРЕДСТВОМ ВЫБОРА [Knuth (1973)] и соответствует алгоритму Джарвиса (разд. 3.3.3).

Алгоритмы построения выпуклой оболочки, основанные на методе «разделяй и властвуй», разбивающие исходное множество точек на две части, строящие выпуклую оболочку для каждой из этих частей и затем объединяющие их за линейное время, являются геометрическими аналогами алгоритма сортировки слиянием. Алгоритм, строящий выпуклую оболочку в реальном времени, является аналогом алгоритма сортировки вставками. Аналоги алгоритма быстрой сортировки обсуждались в разд. 3.3.4.

Хотя $\Omega(N \log N)$ является нижней оценкой сложности в худшем случае для любого алгоритма построения выпуклой обо-

лочки множества $S$, содержащего $N$ точек, тем не менее в случае, когда число крайних точек $h$ существенно меньше $N$, по-видимому, можно достигнуть меньшей сложности. Это соображение с большим успехом использовали Киркпатрик и Зейдель, разработавшие алгоритм со сложностью $O(N \log h)$ и предложившие его в качестве «предельного». Они также доказали, что этот алгоритм является асимптотически оптимальным [Kirkpatrick, Seidel (1983)]. Их алгоритм отдельно строит верхнюю и нижнюю оболочки. Рассмотрим построение верхней оболочки. Исходное множество разбивается на две равные части, отделимые друг от друга вертикальной прямой. Затем вместо рекурсивного построения верхней оболочки для каждой из двух частей и последующего вычисления общей опорной прямой (называемой верхним соединением) алгоритм сначала строит это соединение, а затем раздельно строит верхние оболочки для подмножеств точек, расположенных соответственно слева и справа от левого и правого концов соединения. Ключевым моментом этого метода является эффективный способ построения соединения. Опираясь на то, что соединение можно определить в результате решения задачи линейного программирования, авторы алгоритма воспользовались методом решения задач линейного программирования за линейное время, который был недавно предложен Меджиддо и Дайером и подробно будет рассмотрен в разд. 7.2.5. Обозначим через $h_1$ и $h_2$ $(h_1 + h_2 = h)$ число вершин оболочки соответственно слева и справа от соединения, а через $T(N, h)$ время выполнения алгоритма. Легко проверить, что следующее рекуррентное соотношение:

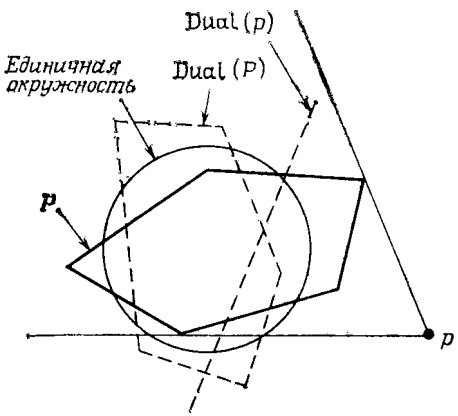



Рис. 3.33. Построение опорных прямых эквивалентно поиску пересечения в двойственном пространстве.

$$T(N, h) = cN + \max_{h_1+h_2=h} (T(N/2), h_1) + T(N/2, h_2)),$$

где $c$ — константа, имеет решение $T(N, h) = O(\log h)$.

Лишь совсем недавно задаче построения выпуклой оболочки в $E^d$ было уделено значительное внимание. Анализ алгоритма Чанда — Капура, выполненный Бхаттачарья, и метод «под-над», предложенный Кейли, представляют два примера возрождения интереса к этой проблеме. Зейдель [Seidel (1981)] независимо и почти одновременно разработал алгоритм, аналогичный алгоритму Кейли. Зейдель применил аналогичный подход, но только в двойственном пространстве (см. разд. 1.3.3). Основную идею алгоритма для случая $d = 2$ иллюстрирует рис. 3.33, где показано, что построение опорных прямых из точки $p$ к многоугольнику $P$ эквивалентно в двойственном пространстве определению пересечения dual$(P)$ с dual$(p)$. Сложность эффективного алгоритма вычисления указанного пересечения в $E^d$ равна $O(N^{\lfloor (d+1)/2 \rfloor})$. Эта оценка была улучшена Кейли до $O(N^{\lfloor d/2+1 \rfloor})$, которая в действительности является оптимальной для четных $d$ ($O(N^{\lfloor d/2 \rfloor})$ — размер получаемого результата).



## 3.6. Упражнения

1. *Простая замкнутая ломаная.* На плоскости заданы $N$ точек. Построить простой многоугольник, вершинами которого являются данные точки.
(a) Показать, что нижняя оценка сложности алгоритма, решающего эту задачу, равна $\Omega(N \log N)$.
(b) Разработать алгоритм решения этой задачи. (Указание: модифицировать метод обхода Грэхема.)

2. На плоскости заданы два непересекающихся выпуклых многоугольника $P_1$ и $P_2$.
(a) Сколько общих опорных прямых могут иметь $P_1$ и $P_2$?
(b) Построить общие опорные прямые с помощью алгоритма, основанного исключительно на использовании углов между ребрами многоугольников и прямыми, проходящими через вершины многоугольников.

3. Разработать алгоритм, выполняющий за постоянное время классификацию вершины $v$ (как вогнутой, опорной или выпуклой) выпуклого многоугольника относительно отрезка $\overline{pv}$, где $p$ — заданная точка на плоскости.

4. Для алгоритма реального времени построения выпуклой оболочки на плоскости из разд. 3.3.6:
(a) доказать, что в случаях 2, 4, 6 и 8, показанных на рис. 3.16, точка $p$ с необходимостью является внешней для многоугольника $P$;
(b) предположив, что $p$ является внутренней точкой многоугольника $P$, подумать, как процедура поиска обнаружит эту ситуацию.

5. Для метода поддержки динамической выпуклой оболочки из разд. 3.3.7:
(a) написать программу на псевдоалголе, реализующую функцию СОЕДИНИТЬ$(U_1, U_2)$, вычисляющую опорный отрезок для $B$-оболочек $U_1$ и $U_2$;
(b) разработать спаренные процедуры ПОДЪЕМ-СПУСК для выполнения операции удаления точки из множества на плоскости.

6. *Кейл — Киркпатрик.* Пусть $S$ — множество из $N$ точек на плоскости с целочисленными координатами в интервале от 1 до $N^d$, где $d$ — константа. Показать, что выпуклую оболочку множества $S$ можно найти за линейное время.

7. *Зейдель.* Пусть $S = \{p_1, \ldots, p_N\} \subset E^3$ и $x(p_1) < x(p_2) < \ldots < x(p_N)$. Показать, что, несмотря на априорное знание порядка следования точек множества $S$ на оси $x$, нижняя оценка сложности построения выпуклой оболочки множества $S$ по-прежнему остается равной $\Omega(N \log N)$. (Указание:

использовать преобразование, переводящее задачу о выпуклой оболочке на плоскости в данную.)

8. *Зейдель.* В алгоритме «заворачивания подарка» Чанда — Капура необходимо поддерживать структуру данных для множества $Q$ из $(d-2)$ гиперграней, которые могут быть «завернуты» на последующих шагах обработки. В этом алгоритме можно печатать гиперграни сразу же после того, как они найдены и нет необходимости запоминать их. Таким образом, размер структуры данных для $Q$ определяет емкостную сложность алгоритма «заворачивания подарка». Пусть $S$ — множество из $n$ точек в пространстве $E^d$, находящихся в общем положении.

(a) Показать, что если для построения выпуклой оболочки множества $S$ используется алгоритм «заворачивания подарка», то существует некоторая последовательность шагов «заворачивания подарка», такая, что размер $Q$ никогда не превышает максимального числа гиперграней политопа с $N$ вершинами в $E^d$.

(b) Разработать вариант алгоритма «заворачивания подарка» с емкостной сложностью

$$O(\varphi_{d-2}) = O(O^{\lfloor (d-1)/2 \rfloor}),$$

где $\varphi_{d-1}$ — число гиперграней политопа.

9. Показать, что в алгоритме построения $d$-мерной выпуклой оболочки методом «под-над» (разд. 3.4.2) классификация каждой из $N$ вершин политопа $P$ как вогнутой, выпуклой или опорной может быть выполнена за время $O(\varphi_{d-1}) + O(Nd)$, где $\varphi_{d-1}$ — число гиперграней политопа $P$.

ГЛАВА 4

# Выпуклые оболочки: расширения и приложения

В этой главе преследуются две цели. Первая — обсуждение вариантов и частных случаев задачи построения выпуклой оболочки, а также анализ поведения алгоритмов построения выпуклой оболочки в среднем. Вторая цель состоит в обсуждении приложений, в которых используется выпуклая оболочка. Новые задачи будут сформулированы и обсуждены в том виде, в каком они возникают в этих приложениях. Многообразие этих задач должно убедить читателя в важности задачи построения выпуклой оболочки как для практики, так и в качестве фундаментального средства для решения задач вычислительной геометрии.

## 4.1. Расширения и варианты

### 4.1.1. Анализ среднего случая

Возвращаясь к обсуждавшимся в разд. 3.3 алгоритмам построения выпуклой оболочки на плоскости, следует отметить, что независимо от исходных данных алгоритм Грэхема всегда требует $O(N \log N)$ времени, так как в начале он выполняет сортировку данных. Напротив, время работы алгоритма Джарвиса колеблется от линейного до квадратичного, и в связи с этим уместно поставить вопрос: каково *среднее* время работы алгоритма? Поиск ответа на этот вопрос приведет в сложную и вместе с тем «захватывающую» область стохастической геометриии, где мы познакомимся с некоторыми трудностями, касающимися анализа поведения геометрических алгоритмов в среднем случае.

Так как алгоритм Джарвиса имеет сложность $O(hN)$, где $h$ — число вершин выпуклой оболочки, то для оценки сложности этого алгоритма в среднем необходимо лишь вычислить $E(h)$ — математическое ожидание величины $h$. Для этого надо

сделать ряд предположений о законе распределения исходных точек. Ответ на этот вопрос лежит в области стохастической геометрии, изучающей свойства случайных геометрических объектов и являющейся основой для анализа работы алгоритмов в среднем [1]). Нам, конечно, хотелось бы сказать: «Даны $N$ точек, равномерно распределенные на плоскости...», — но технические проблемы делают это невозможным, так как точки могут быть равномерно распределены лишь в множестве с конечной мерой Лебега [Kendall, Moran (1963)][2]), так что мы вынуждены определить конкретную область, из которой выбираются точки. К счастью, задаче вычисления $E(h)$ было уделено значительное внимание в литературе по статистике, и ниже приведен ряд теорем, которые имеют прямое отношение к анализу некоторых геометрических алгоритмов.

**Теорема 4.1** [Rényi, Sulanke (1963)]. *Если $N$ точек выбираются равномерно и независимо из выпуклого $r$-угольника на плоскости, то при $N \to \infty$*

$$E(h) = \left(\frac{2r}{3}\right)(\gamma + \log_e N) + O(1), \tag{4.1}$$

*где $\gamma$ — константа Эйлера.*

**Теорема 4.2** [Raynaud (1970)]. *Если $N$ точек выбираются равномерно и независимо из внутренности $k$-мерной гиперсферы, то при $N \to \infty$ математическое ожидание $E(f)$ гиперграней выпуклой оболочки этих точек асимптотически стремится к*

$$E(f) = O(N^{(k-1)/(k+1)}). \tag{4.2}$$

*Отсюда следует, что*

$E(h) = O(N^{1/3})$ *для $N$ точек, равномерно выбранных из круга, и*
$E(h) = O(N^{1/2})$ *для $N$ точек, равномерно выбранных из сферы.*

**Теорема 4.3** [Raynaud (1970)]. *Если $N$ точек выбраны независимо в соответствии с $k$-мерным нормальным распределением, то при $N \to \infty$ $E(h)$ асимптотически стремится к*

$$E(h) = O((\log N)^{(k-1)/2}). \tag{4.3}$$

**Теорема 4.4** [Bentley, Kung, Schkolnick, Thompson (1978)]. *Если координаты $N$ точек в $k$-мерном пространстве выбраны независимо в соответствии с произвольным непрерывным распределением (возможно, для каждой координаты используется свое распределение), то*

$$E(h) = O((\log N)^{k-1}). \tag{4.4}$$

[1]) Довольно исчерпывающую информацию о результатах в этой области можно найти в книге [Santaló (1976)].

[2]) Речь идет об ограниченных множествах точек. — *Прим. перев.*



Условиям этой теоремы удовлетворяют многие распределения, включая равномерное распределение в гиперкубе.

Такое качественно неожиданное поведение выпуклых оболочек случайных множеств точек можно интуитивно объяснить следующим образом: при равномерной выборке из произвольной ограниченной области $F$ оболочка случайного множества точек стремится принять форму границы $F$. В случае многоугольника накопление точек в «углах» приводит к тому, что результирующая выпуклая оболочка имеет очень мало вершин. Так как у круга нет углов, то ожидаемое число вершин выпуклой оболочки сравнительно велико, хотя, как нам известно, нет элементарного объяснения зависимости $N^{1/3}$ для плоского случая [1]).

Из приведенных результатов непосредственно следует, что сложность в среднем алгоритма Джарвиса имеет значения, представленные в табл. I.

*Таблица I.* **Сложность в среднем алгоритма Джарвиса**

| Распределение | Сложность в среднем |
|---|---|
| Равномерное в выпуклом многоугольнике | $O(N \log N)$ |
| Равномерное в круге | $O(N^{4/3})$ |
| Нормальное на плоскости | $O(N(\log N)^{1/2})$ |

Заметим: можно ожидать, что при нормальном распределении точек алгоритм Джарвиса потребует несколько меньше времени, чем алгоритм Грэхема [2]).

Все распределения, рассмотренные в этом разделе, обладают следующим свойством: математическое ожидание числа крайних точек выборки объемом $N$ равно $O(N^p)$, где $p < 1$ — некоторая константа. Такие распределения называются *$N^p$-распределениями*.

Обратимся вновь к описанному в разд. 3.3.5 алгоритму «разделяй и властвуй» для плоского случая. Как уже было пока-

[1]) Математическое ожидание числа вершин выпуклой оболочки множества из $N$ точек можно записать как интеграл от $N$-й степени интеграла плотности вероятности [Efron (1965)]. Но интерес представляет не значение этой величины, а ее асимптотическое поведение в зависимости от $N$.

[2]) Двумерное нормальное распределение часто используется в качестве модели распределения попаданий пуль в цель.

зано, этот алгоритм имеет оптимальную сложность в худшем случае. Бентли и Шеймос [Bentley, Shamos (1978)] показали, что с небольшими изменениями в реализации этот алгоритм достигает линейной оценки сложности в среднем, и, по-видимому, имеется возможность его обобщения на случай пространств более высокой размерности. Для пользы дела коротко напомним идею алгоритма. Если число заданных точек $N$ меньше некоторой константы $N_0$, то оболочка вычисляется некоторым прямым методом. Если же $N$ больше $N_0$, то процедура сначала произвольным образом разбивает множество из $N$ точек на два под множества, содержащие примерно по $N/2$ точек каждое. Затем рекурсивно ищутся выпуклые оболочки для этих двух подмножеств, т. е. решаются две подзадачи, аналогичные исходной. Результатом каждого из этих рекурсивных обращений к процедуре является выпуклый многоугольник. Оболочка исходного множества есть не что иное, как оболочка объединения оболочек подмножеств. Последняя может быть найдена за время, пропорциональное полному числу вершин обеих подоболочек.

При разработке на основе этого метода алгоритма с линейной сложностью в среднем возникает проблема, связанная с шагом разбиения множества на части. Действительно, в том виде, как это сделано в разд. 3.3.5, это разбиение могло бы потребовать линейного времени, как того требует простое копирование подзадач. Такой подход, однако, приводил бы к оценке $O(N \log N)$ для полного времени выполнения алгоритма. Таким образом, следует использовать иной подход, чтобы одновременно достичь двух следующих целей:

(1) шаг разбиения должен выполняться за время $O(N^p)$, где $p < 1$;

(2) точки в каждом из двух подмножеств должны подчиняться тому же закону распределения, что и исходное множество точек.

Вторая цель может быть довольно легко достигнута, если вначале выполнить произвольную перестановку исходных точек и расположить их в массиве (это делается путем записи точек в этот массив в порядке их поступления). Тем самым гарантируется, что любая подпоследовательность полученной последовательности является случайной, т. е. она подчиняется закону исходного распределения. Каждая подзадача касается множества точек, представленных некоторой последовательностью элементов массива (подмассивом), и это множество полностью определяется двумя указателями на граничные элементы соответствующего подмассива. Так что разбиение осуществляется путем расщепления подмассива в средней его части (эта операция выполняется за постоянное время) с последующей передачей в каждом из рекурсивных обращений к процедуре лишь



двух указателей. Использование такого приема позволяет достичь первой цели. Эта цель неявно достигается, если воспользоваться итерационной (а не рекурсивной) схемой реализации алгоритма. При такой реализации последовательно обрабатывались бы снизу вверх сначала множества из четырех точек, затем из восьми и т. д.

Таким образом, время выполнения шага слияния пропорционально сумме размеров выпуклых оболочек каждого из двух подмножеств. Обозначим последнюю величину $C(N)$ и, как обычно, $T(N)$ — время работы алгоритма на множестве размера $N$. Без труда получаем следующее рекуррентное соотношение:

$$T(N) \leqslant 2T(N/2) + C(N).$$

Если теперь положить $T_{\text{ср}}(N) \triangleq E[T(N)]$ (оператор $E[\ ]$ обозначает математическое ожидание), то, учитывая линейность оператора $E$, получаем новое рекуррентное соотношение:

$$T_{\text{ср}}(N) \leqslant 2T_{\text{ср}}(N/2) + E[C(N)]. \tag{4.5}$$

Решение этого соотношения зависит от вида $E[C(N)]$. В частности, если

$$E[C(N)] = \begin{cases} O(N), & \text{то } T_{\text{ср}}(N) = O(N \log N), \\ O(N/\log N), & \text{то } T_{\text{ср}}(N) = O(N \log \log N), \\ O(N^p),\ p < 1, & \text{то } T_{\text{ср}}(N) = O(N). \end{cases} \tag{4.6}$$

Благодаря линейности оператора $E[\ ]$ значение $E[C(N)]$ не превосходит удвоенного значения математического ожидания размером $h(N/2)$ выпуклой оболочки множества из $N/2$ точек. Учитывая, что для всех упоминавшихся выше распределений имеет место свойство $E[h(N/2)] = O((N/2)^p)$ для некоторого $p < 1$, то $E[C(N)] = O(N^p)$. Полученный результат можно сформулировать в следующей теореме:

**Теорема 4.5.** *Для $N^p$-распределения выпуклая оболочка выборки из $N$ точек на плоскости может быть найдена в среднем за время $O(N)$.*

Довольно соблазнительно сказать, что проведенный анализ применим и к описанному в разд. 3.4.3 алгоритму Препараты и Хонга построения выпуклой оболочки в трехмерном пространстве, который также основывается на методе «разделяй и властвуй». Однако (по крайней мере в приведенном изложении) этот алгоритм существенным образом использует факт разделимости двух выпуклых оболочек при их слиянии. К сожалению, это предположение нарушает условие, согласно которому в ходе обработки каждое подмножество должно быть случайным и подчиняться одному и тому же распределению.

### 4.1.2. Алгоритмы аппроксимации выпуклой оболочки

Вместо выбора алгоритма построения выпуклой оболочки на основании его сложности в среднем (о чем говорилось в предыдущем разделе) можно пойти по альтернативному пути, разрабатывая алгоритмы, строящие аппроксимации для реальной выпуклой оболочки, разменивая тем самым точность на

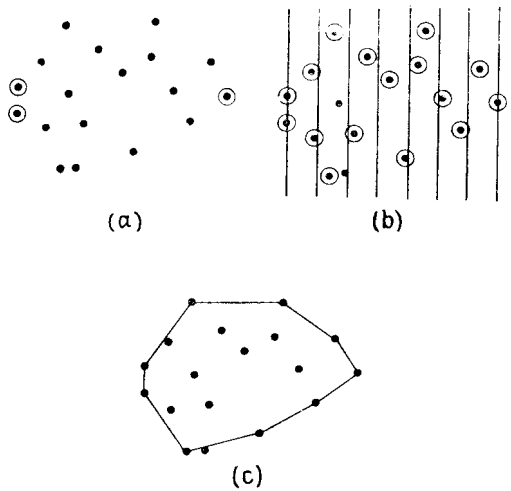



Рис. 4.1. Иллюстрация алгоритма аппроксимации выпуклой оболочки: (a) — задано множество точек на плоскости, в котором выделяются самые левые и самые правые точки; (b) — разбиение множества точек на части в зависимости от принадлежности их одной из $k$ полос и определение в каждой полосе точек с экстремальной ординатой; (c) — приближенная выпуклая оболочка.

простоту и эффективность алгоритма. Такой алгоритм мог бы, в частности, быть очень полезным для приложений, когда необходимо быстро получить решение, жертвуя ради этого даже точностью. Так, например, это вполне приемлемо в прикладной статистике, где результаты наблюдений не являются точными, а известны с некоторой вполне определенной точностью.

Здесь будет рассмотрен один из таких алгоритмов для плоского случая [Bentley, Faust, Preparata (1982)]. Основная идея этого алгоритма проста и состоит в том, чтобы выделить из множества точек некоторое подмножество, выпуклая оболочка которого и будет служить аппроксимацией выпуклой оболочки исходного множества точек. Однако конкретная схема выделения аппроксимирующего подмножества точек, которая будет выбрана далее, основывается на модели вычислений, отличной



от рассматривавшихся до сих пор. А именно в этом разделе будет предполагаться, что функция взятия *целой части* «$\lfloor\ \rfloor$» входит в состав множества примитивных операций [1]). Обратимся к рис. 4.1(a). На первом шаге алгоритма ищутся минимальное и максимальное значения $x$-координаты точек множества, а затем вертикальная полоса между ними разбивается на $k$ полос равной ширины. Эти $k$ полос образуют последовательность «ящиков», по которым будут распределены $N$ точек исходного множества $S$ (для этого как раз и понадобится функция взятия целой части). После этого в каждой из полос ищутся две точки, имеющие минимальное и максимальное значения $y$-координаты (рис. 4.1(b)). Кроме того, выбираются точки с экстремальными значениями $x$-координаты. Если при этом одно и то же экстремальное значение по $x$-координате имеют сразу несколько точек, то из них выбираются только точки с минимальной и максимальной $y$-координатами. Таким образом, результирующее множество, обозначаемое $S^*$, содержит самое большее $2k + 4$ точек. Наконец, строится выпуклая оболочка множества $S^*$, которая служит аппроксимацией оболочки исходного множества (рис. 4.1(c)). Отметим, что получившаяся оболочка в самом деле является лишь аппроксимацией: в примере на рис. 4.1(c) одна из точек исходного множества лежит вне построенной оболочки.

Описанный здесь коротко метод чрезвычайно прост в реализации. Указанные $k$ полос отображаются на массиве из $(k + 2)$ элементов (нулевой и $(k + 1)$-й элементы содержат две точки с экстремальными значениями $x$-координаты: соответственно $x_{\min}$ и $x_{\max}$). Чтобы определить полосу, в которую попадает некоторая точка $p$, необходимо вычесть из $x(p)$ минимальное значение $x$-координаты ($x_{\min}$) и разделить получившуюся разность на $(1/k)$-ю часть разности значений $x$-координат экстремальных точек. Взяв целую часть от получившегося результата, получим номер полосы, содержащей точку. Можно параллельно искать минимум и максимум в каждой полосе, так как для каждой точки проверяется, выходит ли она за пределы текущих значений максимума и минимума, и, если это имеет место, производится необходимое изменение в массиве. И наконец, можно выбрать удобный в данном случае алгоритм построения выпуклой оболочки: очевидно, что наиболее подходящим является вариант алгоритма Грэхема, предложенный Эндрью (см. разд. 3.3.2). Заметим, что точки множества $S^*$ почти упорядочены по значению $x$-координаты. Чтобы получить полностью упорядоченное

[1]) Заметим, что модель вычислений, основанная на деревьях решений, использовавшаяся во всех рассмотренных до сих пор алгоритмах построения выпуклой оболочки, исключает использование функции взятия целой части.

множество, необходимо лишь сравнить в каждой полосе значения $x$-координаты двух точек множества $S^*$, принадлежащих этой полосе.

Работа этого метода также легко поддается анализу. Объем необходимой памяти пропорционален $k$ (в предположении, что исходные точки уже представлены в памяти). Для нахождения минимального и максимального значений $x$-координаты требуется время $\theta(N)$. Поиск экстремумов во всех полосах требует времени $\theta(N)$, а выпуклая оболочка множества $S^*$ может быть построена за время $O(k)$ (так как множество $S^*$ содержит не более $2K + 4$ точек, причем эти точки уже упорядочены по значению $x$-координаты). Следовательно, полное время работы алгоритма равно $\theta(N + k)$.

Простота и эффективность описанного алгоритма имели бы небольшую ценность, если бы получаемый результат был слишком неточным. Попробуем оценить точность такого подхода. Прежде всего, так как для построения выпуклой оболочки алгоритм использует лишь точки исходного множества, то получающаяся аппроксимация является внутренней в том смысле, что любая точка, попавшая внутрь приближенной оболочки, будет также и внутренней точкой действительной выпуклой оболочки. Возникает вопрос: насколько далеко от приближенной выпуклой оболочки может оказаться точка из $S$, находящаяся вне её? Ответ на это вопрос дает следующая довольно очевидная теорема:

**Теорема 4.6.** *Любая точка $p \in S$, не попавшая внутрь приближенной выпуклой оболочки, находится на расстоянии, не превышающем $(x_{\max} - x_{\min})/k$ от этой оболочки.*

*Доказательство.* Рассмотрим полосу, содержащую точку $p$ (рис. 4.2). Так как точка $p$ находится вне приближенной оболочки, то она не может иметь ни наибольшую, ни наименьшую $y$-координату среди точек, попавших в ту же полосу. Поэтому $y_{\min} \leqslant y(p) \leqslant y_{\max}$. Если $u$ — это точка пересечения горизонтальной прямой, проходящей через точку $p$, с ребром приближенной оболочки, то длина отрезка $\overline{pu}$ ограничивает сверху расстояние от точки $p$ до оболочки. Длина отрезка $\overline{pu}$ сама ограничена сверху шириной полосы $(x_{\max} - x_{\min})/k$.

Рассмотренный метод аппроксимации дает приближенную оболочку, являющуюся подмножеством истинной выпуклой оболочки, но нетрудно изменить метод, чтобы получать «супермножество» истинной оболочки. Достаточно лишь заменить каждую экстремальную точку $p$ парой точек на границах полосы, но с той же ординатой, что и у точки $p$. Очевидно, что получающаяся в результате оболочка охватывает истинную выпуклую оболочку. Анализ точности аппроксимации, аналогичный проведенному выше, показывает, что любая точка приближенной оболочки (а не истинной оболочки) находится на расстоянии, не превышающем $(x_{\max} - x_{\min})/k$ от истинной оболочки.



Можно ли применить этот подход в случае пространств более высокой размерности? Ограничимся обсуждением алгоритма, представляющего обобщение на трехмерный случай только что рассмотренного алгоритма для плоскости. Сначала выполняется просмотр множества и определяются точки с минимальным и максимальным значениями координат по осям $x$ и $y$. Получившийся прямоугольник $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ разбивается на (не более чем) $(k+2)\times(k+2)$ клеток прямыми, параллельными осям $x$ и $y$ ($k$ клеток в каждом измерении вводятся для «действительных» точек в клетках и две клетки по краям вводятся для экстремальных значений координат). Отметим, что по одному из измерений может оказаться менее $(k+2)$ интервалов, в случае если область, «заполняемая» проекциями точек множества, не является квадратом. Для каждой клетки разбиения определяются точки с минимальным и максимальным значениями $z$-координаты. Эти точки образуют множество $S^*$, содержащее не более $2(k+2)^2$ точек. В заключение строится выпуклая оболочка множества $S^*$, которая и служит приближенной оболочкой исходного множества. Эта оболочка может быть построена с помощью общего алгоритма Препараты — Хонга для множеств точек в трехмерном пространстве (разд. 3.4.3). Обратите внимание на бросающееся в глаза отличие от метода, использованного в двумерном случае, когда можно было воспользоваться знанием порядка точек на оси $x$: в данном случае структурная организация точек, определяемая решеткой, не используется для получения более быстрого алгоритма, а используется общий алгоритм. Любопытный вопрос состоит в том, можно ли использовать регулярное расположение точек множества $S^*$ в двумерной решетке для построения более эффективного по сравнению с общим алгоритма.

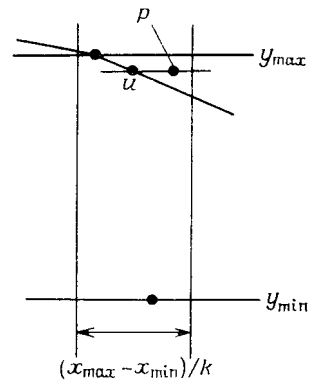


Рис. 4.2. Анализ алгоритма аппроксимации.

Анализ алгоритма для трехмерного случая несколько отличается от того, как это делалось в двумерном случае. Множество $S^*$ может быть найдено за время $\theta(N + k^2)$, но построение выпуклой оболочки множества $S^*$ при использовании общего алгоритма требует времени $O(k^2 \log k)$, а не $O(k^2)$, так как нет

возможности использовать какой-либо естественный порядок точек решетки, как это имело место в двумерном случае. Таким образом, полное время работы алгоритма равно $O(N + k^2 \log k)$.

Для оценки точности метода заметим, что длина диагонали горизонтального сечения трехмерной ячейки является верхней границей для расстояния от приближенной оболочки до любой точки множества $S$, лежащей вне этой оболочки. Как нетрудно найти, длина этой диагонали равна

$$\sqrt{(x_{\max} - x_{\min})^2 + (y_{\max} - y_{\min})^2}/k.$$

Интересная особенность этого подхода состоит в том, что точность результата можно быстро повышать, выполняя больший объем вычислений (увеличивая $k$). В частности, число точек, выбранных для аппроксимации, определяет ее точность, а способ выбора этих точек определяет тип аппроксимации (охватывающая или внутренняя).

### 4.1.3. Задача о максимумах множества точек

В этом разделе обсуждается задача, на первый взгляд имеющая интригующее сходство с задачей построения выпуклой оболочки, однако при более глубоком рассмотрении, что будет сделано несколько позже, обнаруживается ее принципиальное отличие от последней. Эта задача возникает в целом ряде приложений — в статистике, экономике, исследовании операций и т. п. В действительности первоначально эта задача была описана как «задача о плавающих курсах валют» [1]): каждый гражданин Эревона имеет некоторое количество («пакет») денег в иностранной валюте; курсы иностранных валют плавают в довольно широком диапазоне, поэтому каждый вечер человек, пакет валюты которого имеет наибольшую общую стоимость, провозглашается Королем Эревона. Каково наименьшее подмножество населения Эревона, с определенностью содержащее всех потенциальных королей?

Теперь можно точно сформулировать эту задачу. Как обычно, все точки принадлежат $d$-мерному пространству $E^d$ с осями координат $x_1, x_2, \ldots, x_d$ [2]).

**Определение 4.1.** Точка $p_1$ *доминирует над* точкой $p_2$ (записывается так: $p_2 \prec p_1$), если $x_i(p_2) \leqslant x_i(p_1)$ для $i = 1, 2, \ldots, d$. (Отношение «$\prec$» естественно называть *отношением доминирования.*)

[1]) Эта забавная формулировка задачи принадлежит Фримену (1973, не опубликовано).

[2]) В несколько более общей формулировке можно считать, что заданы $d$ упорядоченных множеств $U_1, \ldots, U_d$ и каждый элемент является вектором $v \in V$, где $V$ — декартово произведение $U_1 \times \ldots \times U_d$.



Если задано множество $S$ из $N$ точек в $E^d$, то очевидно, что отношение доминирования определяет на $S$ частичный порядок (при $d > 1$). Точка $p \in S$ является *максимальным элементом* (или коротко просто *максимумом*) множества $S$, если не существует $q \in S$ такого, что $p \neq q$ и $q \succ p$. «Задача о максимумах» заключается в определении *всех* максимумов множества $S$ для данного отношения доминирования.

Проиллюстрируем теперь взаимосвязь между задачей о максимумах и задачей о построении выпуклой оболочки (см. так-

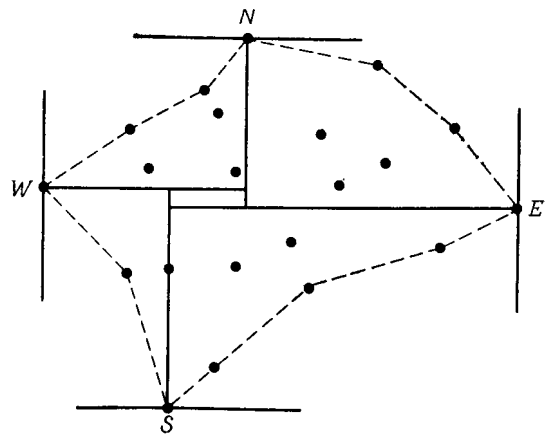



Рис. 4.3. Максимумы в квадрантах, определяемых четырьмя крайними точками $N$, $S$, $E$ и $W$, дают грубое представление границы множества точек.

же [Bentley, Kung, Schkolnick, Thompson (1978)]). Суть имеющегося сходства заключается в том, что как максимумы, так и выпуклая оболочка являются представлениями «границы» множества $S$, хотя первые дают неполное представление этой границы. В частности, если задано некоторое множество точек $S$, то можно поставить столько задач о максимумах, сколько имеется ортантов в пространстве $E^d$ (т. е. $2^d$ задач; см. рис. 4.3, где для случая $d = 2$ имеются четыре ортанта, называемых квадрантами). Каждая из этих задач может быть получена путем приписывания одного из знаков «+» или «—» каждой из координат точек множества $S$ (исходной формулировке задачи соответствует комбинация знаков $(+ + \ldots +)$). Нужную нам взаимосвязь между задачами выражает следующая теорема:

**Теорема 4.7.** *Вершина $p$ выпуклой оболочки множества $S$ является максимумом по крайней мере при одном присваивании знаков координатам точек множества $S$.*

*Доказательство.* Предположим противное, а именно что имеется вершина выпуклой оболочки $p$, не являющаяся максимумом ни при каком присваивании знаков координатам точек. Поместим начало системы координат в точку $p$ и рассмотрим все $2^d$ ортантов пространства $E^d$. Так как точка не является максимумом ни при одном присваивании знаков координатам точек множества $S$, то в каждом ортанте имеется по крайней мере одна точка множества $S$, отличная от $p$. Обозначим множество этих точек через $S^*$. Очевидно, что $\mathrm{conv}(S^*)$ содержит *внутри* точку $p$ и, следовательно, $\mathrm{conv}(S^*) \subseteq \mathrm{conv}(S)$, что противоречит предположению о принадлежности точки $p$ границе выпуклой оболочки.

Далее в этом разделе мы еще вернемся к этой интересной взаимосвязи. Прежде чем переходить к описанию алгоритма решения задачи о максимумах, уместно оценить ее сложность, получив нижнюю оценку для времени работы любого алгоритма отыскания максимумов в рамках модели деревьев вычислений. В частности, приводимое ниже рассуждение относится к несколько более слабому случаю модели линейных деревьев решений, известному как модель деревьев сравнений, хотя область применения полученных результатов можно расширить и на более сильный случай. (В модели деревьев сравнений число аргументов линейной функции ограничено двумя переменными.) Как это уже было в случае задачи построения выпуклой оболочки (разд. 3.1), будем искать нижнюю оценку для случая двумерного пространства. Очевидно, что поскольку любую задачу о максимумах в $E^{d-1}$ можно рассматривать как задачу в $E^d$, то такая нижняя оценка будет справедлива для пространств любой размерности (к сожалению, неизвестно, насколько эта оценка точна для $d > 3$). Нет ничего удивительного в том, что для того, чтобы получить нетривиальную нижнюю оценку сложности, мы прибегнем, как это часто делается для нижних оценок небольшой сложности, к уже знакомому сведению задач:

СОРТИРОВКА $\propto_N$ МАКСИМУМЫ.

Пусть $\mathscr{A}$ — алгоритм, описанный с помощью дерева сравнений $T$ (см. разд. 1.4), о котором утверждается, что он решает любую задачу о максимумах для множества из $N$ точек, заданных на плоскости. Выполнение алгоритма $\mathscr{A}$ при решении любой конкретной индивидуальной задачи соответствует прохождению в дереве $T$ пути из его корня до некоторого листа. Сведение от задачи сортировки выполняется следующим образом. Пусть заданы $N$ действительных чисел $x_1, x_2, \ldots, x_N$. Образуем множество точек на плоскости, положив $S = \{p_i\colon i = 1, \ldots, N;\ p_i = (x_i, y_i),\ x_i + y_i = \mathrm{const}\}$. Это можно сделать за линейное время. Условие $x_i + y_i = \mathrm{const}$ для $i = 1, \ldots, N$ эквивалентно следующему:

$$x_i < x_j \Leftrightarrow y_i > y_j. \tag{4.7}$$

Заметим, что при таком способе построения множества $S$ каждый элемент $(x_i, y_i)$ является максимальным. Для того чтобы определить, что $(x_i, y_i)$ — максимум, алгоритм $\mathscr{A}$ должен уметь устанавливать тот факт, что «не существует такого $j \neq i$, что $x_i < x_j$ и $y_i < y_j$». Но это эквивалентно следующему утверждению: «для всех $j \neq i$ либо $x_i > x_j$, либо $y_i > y_j$, т. е. либо $x_i > x_j$, либо $x_i < x_j$» (в соответствии с (4.7)). Другими словами, для каждой пары $\{x_i, x_j\}$ известен относительный порядок её элементов, т. е. каждому листу дерева $T$ соответствует единственное упорядоченное множество $\{x_1, x_2, \ldots, x_N\}$, что дает решение задачи сортировки. Таким образом, имеет место следующая теорема:

**Теорема 4.8** [Kung, Luccio, Preparata (1975)]. *В рамках модели деревьев сравнений любой алгоритм, решающий задачу о максимумах для двумерного случая, имеет сложность* $\Omega(N \log N)$.

Теперь, имея для сложности алгоритма оценку снизу, рассмотрим вопрос разработки алгоритмов решения задачи о максимумах. Эта задача предоставляет нам великолепную возможность сравнить относительные достоинства и эффективность двух основных парадигм вычислительной геометрии: «разделяй и властвуй» и заметания по одному из измерений (см. разд. 1.2.2).

Начнем с последней. Первым шагом в методе заметания является сортировка всех точек множества $S$ в соответствии со значением выбранной координаты, например $x_d$. Эта координата становится измерением, по которому производится заметание. В соответствии с терминологией, введенной в разд. 1.2.2, «список точек событий» — это очередь $Q$, содержащая точки множества в порядке *уменьшения* координаты $x_d$. Организация структуры, представляющей «статус заметающего сечения», будет рассмотрена позже. Запись $p \prec U$ обозначает, что существует точка $q \in U$ такая, что $p \prec q$. Теперь можно дать набросок следующего общего алгоритма:

```
      function МАКСИМУМЫ1(S, d)
1.  begin M := ∅;
2.        M* := ∅; (* M и M* — это множества максимумов
          соответственно по d и d — 1 измерениям *)
3.        while(Q ≠ ∅) do
4.          begin p ⇐ Q (* выбрать элемент из очереди *)
```

```
5.              p* := проекция p на (x_1, ..., x_{d-1});
6.              if(p* ⊀ M*) then
7.                begin M* := МАКСИМУМЫ1(M* ∪ {p*},
                  d — 1);
8.                      M := M ∪ {p}
                  end;
            end;
9.          return M
    end.
```

Говоря на описательном уровне, функция МАКСИМУМЫ1 выполняет заметание точек множества $S$ в порядке убывания координаты $x_d$ и для каждой точки $p$ определяет (пункты 5, 6), доминирует над ней или нет по координатам $x_1, x_2, \ldots, x_{d-1}$ какая-либо из уже просмотренных точек (эти точки уже доминируют над $p$ по координате $x_d$ в соответствии с организацией процедуры заметания). Тем самым обеспечивается корректность этого подхода, критическим местом которого являются, как это очевидно, пункты 6 и 7: «*если* $p^* \not\prec M^*$, *то* $M^* :=$ МАКСИМУМЫ1$(M^* \cup \{p^*\}, d-1)$». По существу, это не что иное, как последовательное (т. е. по одной точке за один раз) построение максимумов $(d-1)$-мерного множества. Это совсем не тривиальная задача. Эффективный способ ее решения известен лишь для $d = 2, 3$.

В самом деле, для $d = 2$ множество $M^*$ является одномерным (а значит, оно содержит лишь один элемент) и проверка условия «$p^* \prec M^*$?» сводится к проверке «$x_1(p) \leqslant \bar{x}_1 = \max\limits_{q \in M} x_1(q)$?».

При этом операция по изменению множества $M^*$

$$M^* := \text{МАКСИМУМЫ1}(M^* \cup \{p^*\},\ d-1)$$

превращается в следующее присваивание $\bar{x}_1 := x_1(p)$. Как операция проверки, так и операция присваивания могут быть выполнены за постоянное время. Для $d = 3$ ситуация естественно является более сложной, но по-прежнему остается управляемой. В этом случае $M^*$ — двумерное множество максимумов (рис. 4.4) — является линейно упорядоченным и может быть организовано в виде сбалансированного по высоте дерева поиска по координате $x_2$. Проверка условия «$p^* \prec M^*$?» превращается в поиск $x_2(p^*)$ в этом дереве: если $x_2(q)$ следует непосредственно за $x_2(p^*)$, то имеет место эквивалентность

$$(p^* \prec M^*) \Leftrightarrow x_1(p^*) \leqslant x_1(q).$$

Корректировка множества $M^*$ в случае $p^* \not\prec M^*$ производится в результате прохождения дерева, начиная с $x_2(p^*)$, в «направлении» уменьшения координаты $x_2$. При этом удаляются все элементы множества до тех пор, пока не будет найдена первая



точка $q'$, для которой $x_1(q') > x_1(p^*)$. Легко найти, что время, затрачиваемое на решение этой задачи в целом, составляет $O(N \log N)$, так как на каждую проверку «$p^* \prec M^*$?» (с возможной вставкой элемента в дерево) требуется время $O(\log N)$, в то время как каждая операция удаления элемента при прохождении дерева в процессе его изменения может быть выполнена за постоянное время путем тривиального изменения дерева поиска (прошиванием). Учитывая, что производятся $N$ операций

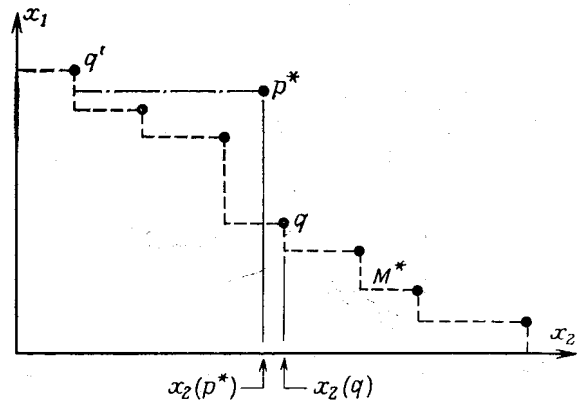



Рис. 4.4. Множество максимумов $M^*$ можно линейно упорядочить. При этом проверка условия «$p^* \prec M^*$?» заключается в сравнении $x_1$-координаты точки $p^*$ с $x_1$-координатой ближайшей к $p^*$ точки множества $M^*$, расположенной справа от нее.

«проверить — вставить» и не более $N$ операций «удалить», получаем указанную оценку. Если вспомнить, что метод заметания требует предварительной сортировки, приходим к выводу, что в случае $d = 2, 3$ время работы алгоритма МАКСИМУМЫ1 является оптимальным и составляет $\theta(N \log N)$.

К сожалению, насколько нам известно, уже для $d = 4$ этот метод имеет оценку $O(N^2)$. Так что необходимо искать иной подход к решению этой задачи. Один из возможных подходов основывается на методе «разделяй и властвуй». Как будет видно из дальнейшего, соответствующий алгоритм предполагает, что множества точек по очереди разбиваются на части в соответствии со значениями координат $x_d, x_{d-1}, \ldots, x_2$. Поэтому, чтобы облегчить выполнение этих часто используемых в алгоритме операций, удобно ввести некоторый шаг предварительной обработки. Этот шаг заключается в сортировке элементов множества $S$ по каждой из координат. Результирующая структура данных представляет многосвязный список ($(d-1)$-кратно прошитый список), каждый узел которого прошит каждым из

односвязных списков, соответствующих выбранным координатам. Для создания такой структуры данных требуется время $O(dN \log N)$. Эта структура позволяет легко и эффективно разбивать множество на части. Избегая тяжеловесных оборотов, будем говорить, что $x_j$ разбивает $S$ на $(S_1, S_2)$, если для каждой $p \in S_1$ имеем $x_j(p) \leqslant x_j$, а для каждой $q \in S_2$ имеем $x_j(q) > x_j$. Будем также говорить, что $S$ разбито пополам по $x_j$ на $(S_1, S_2)$, если $\bar{x}_j$ разбивает $S$ на $(S_1, S_2)$ и $\bar{x}_j$ является медианой множества точек $S$ по координате $x_j$ (так что $|S_1| \simeq |S_2|$).

Первым шагом алгоритма поиска всех максимумов МАКСИМУМЫ2 является разбиение пополам заданного множества $S$ по координате $x_d$ на $(S_1, S_2)$. Затем алгоритм рекурсивно применяется к множествам $S_1$ и $S_2$. В результате получаются два

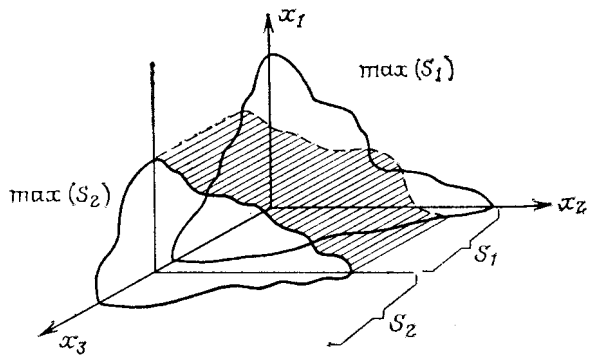



Рис. 4.5. Шаг «слияния» в методе «разделяй и властвуй» для случая $d = 3$.

множества соответствующих им максимумов $\max(S_1)$ и $\max(S_2)$. При объединении результатов двух рекурсивных обращений к процедуре следует учитывать, что все элементы множества $\max(S_2)$ являются также максимумами $S$, в то время как для элементов $\max(S_1)$ это может не иметь места. Действительно, элемент множества $\max(S_1)$ является максимальным элементом $S$ тогда и только тогда, когда над ним не доминирует ни один из элементов множества $\max(S_2)$. Соответствующая ситуация показана на рис. 4.5 для случая $d = 3$. Таким образом, если $U$ и $V$ — два множества точек таких, что для всех $p \in U$ и $q \in V$ имеет место $x_d(p) \leqslant x_d(q)$, то удобно определить новое множество фильтр$(U|V) \subset U$:

$$\text{фильтр } (U|V) \triangleq \{p\colon\ p \in U \text{ и } p \not< V\}.$$

Ясно, что шаг «слияния» в методе «разделяй и властвуй» сводится к вычислению множества фильтр$(\max(S_1)|\max(S_2))$. Таким образом, получаем следующий простой алгоритм:



```
function МАКСИМУМЫ2(S, d)
begin if (|S| = 1) then return S
      else begin(S1, S2) := разбиение S пополам по x_d;
                 M1 := МАКСИМУМЫ2(S1, d);
                 M2 := МАКСИМУМЫ2(S2, d);
                 return M2 ∪ фильтр (M1 | M2)
           end
end.
```

Если обозначить через $T_d(|S|)$ время обработки процедуры МАКСИМУМЫ2$(S)$, а через $F_{d-1}(|S_1|, |S_2|)$ время, необходимое для вычисления множества фильтр$(S_1|S_2)$ [1]), то сразу получаем следующее рекуррентное соотношение (здесь предполагается, что $N$ — четное число):

$$T_d(N) \leqslant 2T_d(N/2) + F_{d-1}(N/2,\ N/2) + O(N), \tag{4.8}$$

где $O(N)$ — время, используемое на разбиение множества $S$ на равные части $S_1$ и $S_2$.

Очевидно, что основные затруднения в рассматриваемом методе связаны с вычислением множества, названного нами фильтром. Предположим снова, что $U$ и $V$ — два множества точек в $E^d$, определенные как это сделано выше, и $|U| = u$, а $|V| = v$. Воспользуемся еще раз методом «разделяй и властвуй». Разделим $V$ пополам по $x_{d-1}$ на $(V_1, V_2)$ и обозначим через $\bar{x}_{d-1}$ наибольшее значение координаты $x_{d-1}$ точек из $V_1$. Величина $\bar{x}_{d-1}$ разбивает $U$ на $(U_1, U_2)$. Рис. 4.6 иллюстрирует эту ситуацию. На этом рисунке показаны проекции множеств $U$ и $V$ на плоскость $(x_d, x_{d-1})$. Заметим, что $|V_1| \simeq v/2$ и $|V_2| \simeq v/2$, в то время как $|U_1| = m$ и $|U_2| = u - m$, где $m$ — некоторое целое число. При таком разбиении множеств исходная задача вычисления множества фильтр$(U|V)$ заменяется четырьмя подзадачами вычисления множеств фильтр$(U_1|V_1)$, фильтр$(U_1|V_2)$, фильтр$(U_2|V_1)$ и фильтр$(U_2|V_2)$. Отметим, однако, что множество фильтр$(U_2|V_1)$ вычисляется тривиально, так как для каждой $p \in U_2$ и $q \in V_1$ имеем $x_d(p) \leqslant x_d(q)$ и $x_{d-1}(p) > x_{d-1}(q)$. Кроме того, каждый элемент из $V_2$ доминирует над каждым элементом из $U_1$ по координатам $x_d$ и $x_{d-1}$, так что подзадача вычисления множества фильтр$(U_1|V_2)$ имеет в действительности размерность $d - 2$. Две оставшиеся подзадачи по-прежнему имеют размерность $d - 1$, но относятся к множествам с меньшим числом элементов. Наглядно процесс вычисления можно

[1]) Следует объяснить индексы, использованные в записи: $T(\,)$ и $F(,)$. Множество $S$ — это множество точек в $E^d$. Однако, так как каждая точка множества $S_2$ доминирует над каждой точкой множества $S_1$ по координате $x_d$, то вычисление множества фильтр $(S_1|S_2)$ является фактически $(d-1)$-мерной задачей.

представить как прохождение путей, порождаемых двойной рекурсией: по размерам обрабатываемых множеств и по размерности. Первая заканчивается, когда мощность одного из двух множеств пары становится маленькой (равной единице), вторая завершается, когда размерность становится настолько маленькой, что применим прямой эффективный метод (это имеет место

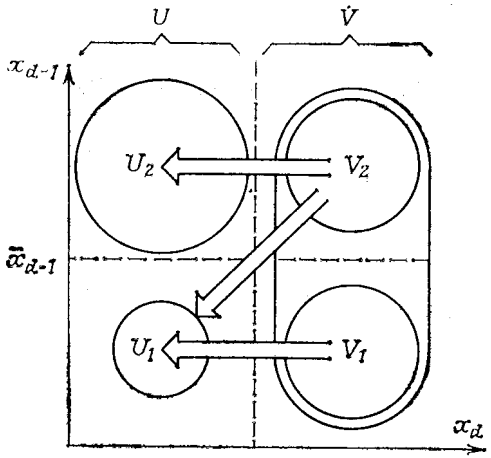



Рис. 4.6. Вычисление множества фильтр$(U|V)$ методом «разделяй и властвуй» (путем сведения к трем более простым подзадачам).

при размерности, равной 3, когда можно воспользоваться модификацией метода заметания по измерению, описанного ранее). Более формально алгоритм описывает следующая функция (заметим, что в алгоритме МАКСИМУМЫ2 множество фильтр$(M_1|M_2)$ вычисляется в результате обращения к ФИЛЬТР$(M_1, M_2;\ d-1)$):

```
        function ФИЛЬТР(U, V; d)
begin if (d = 3) then A := ФИЛЬТР2(U, V);
      if (V = {q}) then A := {p: p ∈ U и p ≮ q};
      if (U = {p}) then
            begin if for each q ∈ V: p ≮ q then A := {p}
                  else A := ∅
            end
      else begin (V1, V2) := разбиение V пополам по x_d;
                  x̃_d = max {x_d(p): p ∈ V1};
                  (U1, U2) := части, на которые x̃_d разбивает U;
                  A := ФИЛЬТР(U2, V2; d) ∪ (ФИЛЬТР(U1,
                  V1; d) ∩ ФИЛЬТР(U1, V2; d − 1))
            end;
      return A
end.
```



Осталось описать процедуру ФИЛЬТР2. Эта процедура использует метод заметания и аналогична процедуре МАКСИМУМЫ1 для $d = 2$. Точки множества $U \cup V$ просматриваются в порядке уменьшения координаты $x_2$ и помещаются в очередь $Q$. Если текущая точка $p$ принадлежит $V$, то $M$ изменяется, в противном случае $p$ либо добавляется к $M$ (как максимальный элемент), либо исключается из рассмотрения. Таким образом, имеем

```
      function ФИЛЬТР2(U, V)
begin Q := упорядочить U ∪ V в порядке убывания
        координаты x_d;
      M := ∅;
      x*_1 := 0;
      while (Q ≠ ∅) do
         begin p ⇐ Q;
               if (x_1(p) > x*_1) then
                  begin if (p ∈ U) then M := M ∪ {p}
                  (* является максимальным элементом *)
                        else x*_1 := x_1(p)
                  end
         end;
      return M
end.
```

Для анализа работы алгоритма обозначим через $F_d(u, v)$ время обработки процедуры ФИЛЬТР$(U, V; d)$. Анализ процедуры ФИЛЬТР2 непосредственно дает $F_2(u, v) = O(u+v)$ (при условии что $U$ и $V$ предварительно упорядочены по значению координаты $x_2$). В общем случае, анализируя процедуру ФИЛЬТР, получаем следующее рекуррентное соотношение:

$$F_d(u, v) \leqslant F_d(m, v/2) + F_d(u-m, v/2) + F_{d-1}(m, v/2), \qquad (4.9)$$

где $m$ — некоторое целое число $0 \leqslant m \leqslant u$. Довольно утомительные вычисления [1]) с учетом оценки для $F_2$ показывают, что максимальное значение правой части соотношения есть $O((u+v)\log u(\log v)^{d-3})$. С учетом этого неравенство (4.8) принимает вид

$$T_d(N) \leqslant 2T_d(N/2) + O\left(N(\log N)^{d-3}\right) + O(N),$$

и, следовательно,

$$T_d(N) = O(N(\log N)^{d-2}), \qquad d \geqslant 2. \qquad (4.10)$$

С учетом времени, затрачиваемого на предварительную сортировку, получаем следующую теорему:

[1]) Соответствующее доказательство приведено в работе [Kung, Luccio, Preparata (1975)].

**Теорема 4.9.** *Максимумы множеств из $N$ точек в $E^d$ ($d \geqslant 2$) могут быть найдены за время $O(N(\log N)^{d-2}) + O(N \log N)$.*

Этот замечательный успех метода «разделяй и властвуй», потерпевшего неудачу в решении задачи о выпуклой оболочке для $d > 3$ (см. разд. 3.4), является ключом к объяснению глубоких различий между двумя задачами. Ключевое в этом вопросе понятие тесно связано, но не в полной мере совпадает с понятием «декомпозируемости», введенным Бентли [Bentley (1979)]. Более конкретно, давайте рассмотрим задачи поиска (см. гл. 2), связанные с задачами о максимумах и выпуклой оболочке, т. е. задачи «проверка на максимум» и «принадлежность выпуклой оболочке» соответственно. Предположим, множество точек $S$

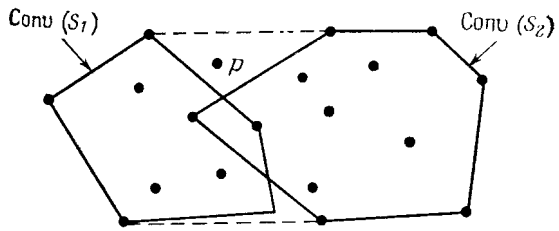



Рис. 4.7. Точка $p$ находится вне $\mathrm{conv}(S_1)$ и $\mathrm{conv}(S_2)$, но внутри выпуклой оболочки их объединения.

произвольным образом разбито на части $\{S_1, S_2\}$. Вопрос «Является ли $p$ максимумом множества $S \cup \{p\}$?» имеет положительный ответ тогда и только тогда, когда получены положительные ответы на вопросы «Является ли $p$ максимумом множества $S_1 \cup \{p\}$?» и «Является ли $p$ максимумом множества $S_2 \cup \{p\}$?». Однако нетрудно разбить $S$ на части таким образом, что ответ на вопрос «Находится ли $p$ вне выпуклой оболочки множества $S$?» является отрицательным, в то время как оба вопроса «Находится ли $p$ вне выпуклой оболочки множества $S_1$?» и «Находится ли $p$ вне выпуклой оболочки $S_2$?» имеют положительные ответы (рис. 4.7).

Прежде чем завершить этот раздел, рассмотрим работу обсуждавшегося алгоритма в среднем случае при довольно общем предположении о распределении вероятностей. Необходимый нам для этого результат дает следующая

**Теорема 4.10** [Bentley, Kung, Schkolnick, Thompson (1978)]. *Среднее число максимумов $\mu(N, d)$ множества $S$ из $N$ точек в $E^d$ равно*

$$\mu(N, d) = O((\log N)^{d-1}) \tag{4.11}$$

*при условии, что координаты каждой точки независимы и выбираются из одного и того же непрерывного распределения.*



Заметим, что при используемом в алгоритме МАКСИМУМЫ2 способе разбиения множества на части сохраняется случайность распределения точек в каждой из частей, так как точки в обеих частях подчинены тому же самому закону распределения. Кроме того, разбиение множества можно выполнить за постоянное время, используя для этого стратегию «передачи указателя», рассмотренную в общих чертах в разд. 4.1.1. С учетом сказанного при анализе среднего случая соотношение (4.8) заменяется следующим соотношением:

$$E[T_d(N)] \leqslant 2E[T_d(N/2)] + E[F_{d-1}(m_1, m_2)] + O(1), \tag{4.12}$$

где $m_1$ и $m_2$ — мощности множеств максимумов множеств $S_1$ и $S_2$, являющихся теперь случайными величинами со средним значением $\mu(N/2, d)$. Из предыдущего обсуждения имеем

$$F_{d-1}(m_1, m_2) \leqslant K_1(m_1 + m_2)\log m_1 (\log m_2)^{d-4},$$

где $K_1$ — некоторая константа.

Так как $m_1$ и $m_2$ ограничены сверху значением $N/2$, то

$$F_{d-1}(m_1, m_2) \leqslant K_1(m_1 + m_2)(\log N)^{d-3},$$

и, следовательно,

$$\begin{aligned} E[F_{d-1}(m_1, m_2)] &\leqslant K_1(\log N)^{d-3}(E[m_1] + E[m_2]) = \\ &= K_1(\log N)^{d-3} 2\mu\left(\frac{N}{2}, d\right) = \\ &= O((\log N)^{d-3}\log(N)^{d-1}) = O((\log N)^{2d-4}). \end{aligned}$$

Подставляя это значение в (4.12), получаем следующий результат:

**Теорема 4.11.** *В предположении, что координаты каждой точки независимы и подчинены одному и тому же непрерывному закону распределения, максимумы множества из $N$ точек в $E^d$ могут быть найдены за время, равное в среднем*

$$E[T_d(N)] = O(N). \tag{4.13}$$

### 4.1.4. Выпуклая оболочка простого многоугольника

Всякий раз, когда на исследуемую задачу накладывается некоторое ограничение, множество рассматриваемых ее индивидуальных экземпляров, как правило, сокращается. В таких случаях наибольший интерес вызывает вопрос, позволит ли это ограничение разработать специально для этого случая существенно более простой метод по сравнению с общим. Конечно, не всегда удается дать убедительный ответ на этот вопрос, разработав более простой алгоритм или установив нижнюю

оценку сложности решения задачи, аналогичную нижней оценке для общего случая. Но во всех случаях исследование этого вопроса заслуживает внимания.

Примером такой «ситуации с ограничением» является задача о выпуклой оболочке простого многоугольника. Заметим, что если имеется множество $S$ из $N$ точек, то его всегда можно рассматривать как многоугольник, упорядочив произвольным образом точки множества и предположив, что между соседними (с учетом цикличности) точками получившейся последовательности имеется ребро. Однако в общем случае получившийся многоугольник не будет простым, т. е. некоторые его ребра могут пересекаться. Но что произойдет, если нам известно, что этот многоугольник простой? (Определение простого многоугольника см. в разд. 1.3.)

Подход, основанный на поиске нижних оценок, в данной ситуации неприменим даже для случая упорядоченной оболочки, так что следует довольствоваться тривиальной нижней оценкой $\Omega(N)$. Поэтому вполне естественно искать алгоритмы, имеющие в худшем случае сложность, меньшую $O(N \log N)$. И ряд таких алгоритмов, каждый со сложностью $O(N)$, был предложен. К сожалению, некоторые из них оказываются неработоспособными в некоторых очень частных случаях [Sklansky (1972); Shamos (1975a)], но другие алгоритмы правильно решают данную задачу. Среди последних следует упомянуть довольно сложный алгоритм Маккаллума — Эйвиса [McCallum, Avis (1979)], использующий два стека. Впоследствии были предложены два новых алгоритма, использующих один стек, — первый Ли [Lee (1983a)], а второй Грэхемом и Яо [Graham, Yao 1983)]. Ниже описывается авторский вариант алгоритма Ли.

Пусть $p_1$ — самая левая вершина заданного простого многоугольника, а $(p_1, p_2, \ldots, p_N)$ — упорядоченная циклическая последовательность его вершин (за вершиной $p_N$ следует $p_1$). Предполагается, что внутренность $P$ находится по правую сторону при обходе его границы в указанном порядке, т. е. циклическая последовательность вершин многоугольника ориентирована по часовой стрелке (рис. 4.8). Пусть $p_M$ — самая правая вершина. Ясно, что $p_1$ и $p_M$ являются граничными точками выпуклой оболочки многоугольника $P$. Кроме того, они разбивают последовательность вершин многоугольника на две цепи: одна от $p_1$ до $p_M$, другая от $p_M$ до $p_1$. Достаточно рассмотреть построение выпуклой оболочки лишь для цепи $(p_1, p_2, \ldots, p_M)$, которую, согласно использовавшейся ранее терминологии (разд. 3.3.2), будем называть *верхней оболочкой*. Пусть *подпоследовательность* $(q_1, q_2, \ldots, q_R)$ последовательности $(p_1, p_2, \ldots, p_M)$, у которой $q_1 = p_1$ и $q_R = p_M$, — искомая выпуклая оболочка многоугольника. Каждое из ребер $\overline{q_i q_{i+1}}$ $(i = 1, \ldots, r-1)$ можно рассматривать как «крышку» «кармана» $K_i$, где карман $K_i$ — это подцепочка последовательности $(p_1, p_2, \ldots, p_M)$, первой и последней вершинами которой являются $q_i$ и $q_{i+1}$ соответственно.



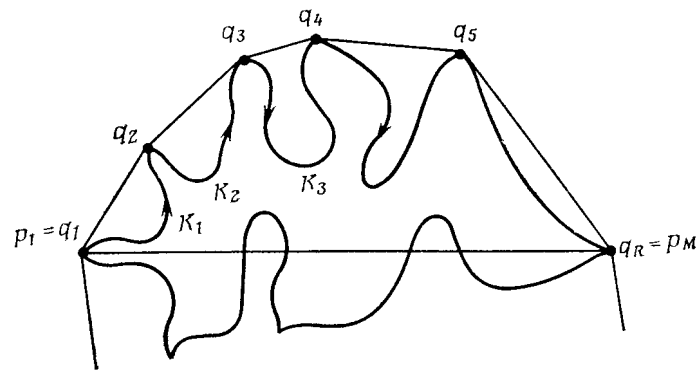



Рис. 4.8. Верхняя оболочка простого многоугольника.

Алгоритм проходит цепь $(p_1, \ldots, p_M)$ и последовательно строит крышки всех карманов. Особым событием в ходе работы алгоритма является обнаружение вершины типа $q$, образующей карман вместе с последней найденной вершиной типа $q$. Такие вершины $q$ будем называть критическими вершинами. Заметим, однако, что не каждая критическая вершина является вершиной окончательной выпуклой оболочки. Они являются лишь кандидатами в число таковых. По сути дела, единственное, что утверждается перед завершением работы алгоритма, это то, что построенная к этому моменту последовательность критических вершин $(q_1, q_2, \ldots)$ согласуется с предположением о простоте многоугольника $P$ (более детальная характеристика будет дана далее). Так что теперь внимание будет сконцентрировано на процедуре перехода от одной критической точки к другой. Эта процедура называется *шагом продвижения*.

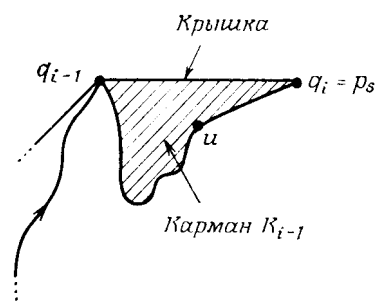



Рис. 4.9. Карман и его крышка.

Предположим, что граница многоугольника просмотрена от вершины $p_1$ до $p_s$ $(s \leqslant M)$ и вершина $p_s = q_i$ является критической. Соответствующая ситуация показана на рис. 4.9, где карман $K_{i-1}$ закрыт крышкой $q_{i-1}q_i$. Обозначим через $u$ вершину

границы $P$, предшествующую $q_i$. В зависимости от положения $u$ относительно ориентированного отрезка $\overrightarrow{q_M q_i}$ имеют место два случая.

1. *Вершина $u$ находится справа от $\overrightarrow{q_M q_i}$ или на нем.* В этом случае в вертикальной полосе, задаваемой вершинами $q_1$ и $q_M$, можно рассмотреть и три области, обозначенные на рис. 4.10(a)

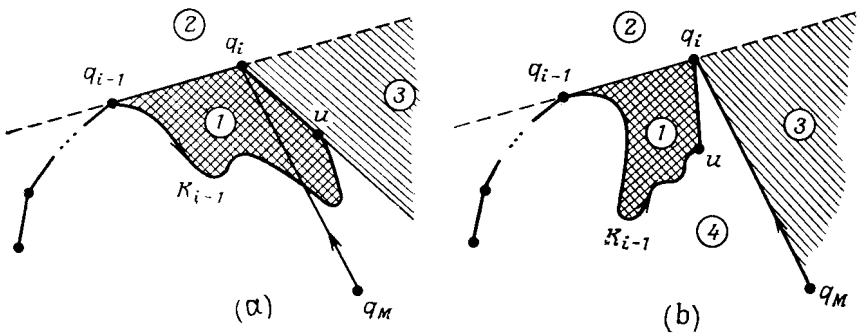



Рис. 4.10. Области, в которых может находиться вершина $v$, следующая за $q_i$ (показаны два случая — в зависимости от относительного положения $u$ и $\overrightarrow{q_M q_i}$).

①, ② и ③. Эти области определяются: прямой, проходящей через вершины $q_{i-1}$ и $q_i$; лучом, являющимся продолжением отрезка $q_i u$, и частью границы многоугольника $P$, соответствующей карману $K_{i-1}$.

2. *Вершина $u$ находится слева от $\overrightarrow{q_M q_i}$.* В этом случае в дополнение к предыдущим определим еще одну область ④, показанную на рис. 4.10(b).

Обозначим через $v$ вершину, следующую за $q_i$ на границе многоугольника $P$. Вершина $v$ находится в одной из областей, определенных выше. Если $v$ находится в областях② или ③, то она является критической вершиной, а если она находится в областях ① или ④, то она не является критической. Рассмотрим четыре указанных случая более подробно.

1. Вершина $v$ принадлежит области ①. В этом случае граница многоугольника заходит в карман. Будем продвигаться по границе до тех пор, пока не достигнем первого ребра границы, одна из вершин которого $w$ находится вне кармана, т. е. в области ② (рис. 4.11(a)). Так как многоугольник $P$ простой, а карман и его крышка также образуют простой многоугольник, то, согласно теореме о жордановой кривой, граница многоугольника $P$ обязательно пересекает крышку кармана. Далее обрабатываем $w$, как вершину $v$ в случае, когда $v$ принадлежит области ②, рассматриваемом ниже.



2. Вершина $v$ принадлежит области ②. Вершина $v$ является критической. Ищется опорная прямая из вершины $v$ к цепи $(q_1, \ldots, q_{i-1})$. Если эта прямая содержит $q_r$ $(r < i)$, то вершины $q_{r+1}, \ldots, q_i$ удаляются, а $v$ берется в качестве новой $q_{r+1}$

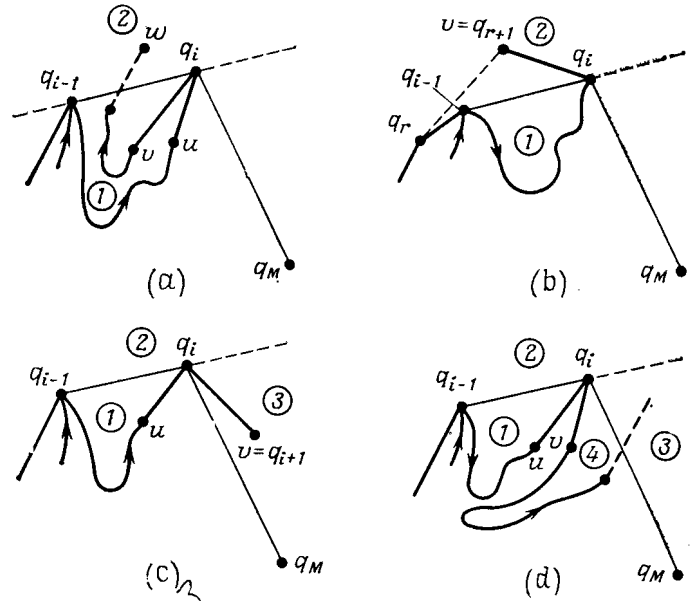



Рис. 4.11. При обходе границы многоугольника $P$, следующей за выделенной вершиной $q_i$, возможны четыре случая.

(рис. 4.11(b)). Ясно, что $v$ является вершиной выпуклой оболочки (критической точкой), так как она является внешней по отношению к текущей оболочке $(q_1, q_2, \ldots, q_M)$. Проведение опорной прямой эквивалентно построению $\mathrm{conv}(q_1, \ldots, q_r, q_{r+1}, q_M)$.

3. Вершина $v$ принадлежит области ③. Вершина $v$ является критической, и она берется в качестве $q_{i+1}$. Действительно (рис. 4.11(c)), $v$ является внешней по отношению к текущей оболочке $(q_1, q_2, \ldots, q_i, q_M)$, и так как она находится справа от прямой, проходящей через $q_{i-1}$ и $q_i$ (и ориентированной в направлении от $q_{i-1}$ к $q_i$), то угол $(q_{i-1} q_i v)$ является выпуклым. (Заметим, что рис. 44.11(c) иллюстрирует случай, соответствующий рис. 4.10(b).)

4. Вершина $v$ принадлежит области ④. В этом случае граница многоугольника заходит внутрь выпуклой оболочки. Как и в случае 1, рассмотренном выше, будем продвигаться по границе многоугольника до тех пор, пока не достигнем первого ребра, обладающего следующим свойством: одна из его вершин является внешней по отношению к области ④ или совпадает с $q_M$. В последнем случае процедура завершается. В первом же случае вершина, являющаяся внешней по отношению к области ④, находится либо в области ③ (такая ситуация обрабатывается способом, указанным в пункте 3 выше), либо в области ② (такая ситуация обрабатывается способом, указанным в пункте 2 выше) (рис. 4.11(d)). Действительно, обозначим через $C$ часть границы многоугольника от $q_i$ до $q_M$. $C$ является простой кривой. Кроме того, прежде чем она пересечет $\overline{q_iq_M}$, она не может выйти ни в область, находящуюся справа от вертикальной прямой, проходящей через $q_M$, ни в область, находящуюся слева от вертикальной прямой, проходящей через $q_1$. Так как многоугольник $P$ простой, то $C$ не может пересечь ни одну из границ карманов $K_1, \ldots, K_{i-1}$, и, следовательно, она не может пересечь ни одну из крышек этих карманов. Отсюда следует, что $C$ может пересечь только $\overline{q_iq_M}$. Если она не пересекает указанный отрезок, то она заканчивается в точке $q_M$.

Предыдущее обсуждение включает доказательство корректности любой реализации, в которой обработка случаев 1—4 производится указанным выше способом. Дадим формальное описание одного из таких алгоритмов. Наиболее подходящими структурами данных являются: (1) очередь $P$, содержащая последовательность $(p_2, p_3, \ldots, p_M)$; (2) стек $Q$ для хранения последовательности $(q_0, q_1, q_2, \ldots)$ — ввиду того, что для вставки и удаления точек выпуклой оболочки используется механизм «последний вошел — первый вышел», где $q_0$ — фиктивная вершина, имеющая ту же абсциссу, что и $p_1 = q_1$, но меньшую ординату и обеспечивающая единообразие обработки. Как было сказано выше, $u$ — это вершина на границе многоугольника $P$, непосредственно предшествующая вершине $q_i$, а $v$ — текущая вершина. Упорядоченная тройка вершин $(stw)$ называется *правым поворотом*, если $w$ находится справа от прямой, проходящей через $s$ и $t$ с учетом направления от $s$ к $t$; в противном случае она называется *левым поворотом*. Кроме того, наряду с обычным использованием операции $U \Leftarrow v$, означающей: «ЗАТОЛКНУТЬ $v$ В $U$», операция ВЫТОЛКНУТЬ $U$ означает: «$v \Leftarrow U$; игнорировать $v$»; $q_i$ обозначает вершину, находящуюся на вершине стека, а ПЕРЕДНИЙ$(P)$ обозначает первый элемент очереди $P$.



```
      procedure ОБОЛОЧКА-МНОГОУГОЛЬНИКА
                (p_1, ..., p_M)
1.  begin P ⇐ (p_2, ..., p_M);
2.        Q ⇐ q_0;
3.        Q ⇐ p_1;
4.        while(P ≠ ∅) do
5.           begin v ⇐ P;
6.                 if((q_{i-1}q_iv) — правый поворот)
                    (* области ①,③,④ *) then
7.                  if((uq_iv) — правый поворот (* области
                      ③,④ *) then
8.                    if((q_Mq_iv) — правый поворот
                        (* область ③ *) then Q ⇐ v
9.                    else (* область ④ *)
10.                        while(ПЕРЕДНИЙ(P) находится
                                  ⟶
                                слева от q_Mq_i или на нем) do
                                ВЫТОЛКНУТЬ P
11.                 else (* область ① *)
12.                   while(ПЕРЕДНИЙ(P) находится слева
                                ⟶
                              от q_iq_{i-1} или на нем) do
                              ВЫТОЛКНУТЬ P
13.               else (* область ② *)
14.                  begin while ((q_{i-1}q_iv) — левый поворот)
                     do ВЫТОЛКНУТЬ Q;
15.                     Q ⇐ v
                     end
           end
    end.
```

Анализ сложности алгоритма ОБОЛОЧКА-МНОГОУГОЛЬНИКА не вызывает затруднений. После инициализации (строки 1—3) каждая вершина границы посещается в точности один раз, прежде чем она будет либо принята (строки 8, 15), либо отвергнута (строки 10, 12). Обработка каждой вершины многоугольника осуществляется за постоянное время. При построении опорной прямой в цикле *while* (строка 14) на каждую операцию удаления затрачивается постоянное время. Учитывая, что последовательности $(p_1, \ldots, p_M)$ и $(q_1, \ldots, q_R)$ содержат $O(M)$ элементов, а рассуждения, аналогичные приведенным, можно применить и для построения нижней оболочки прямоугольника $P$, то в завершение имеем следующую теорему:

**Теорема 4.12.** *Выпуклая оболочка простого многоугольника с $N$ вершинами может быть построена за оптимальное время $\theta(N)$ при использовании памяти объемом $\theta(N)$.*

## 4.2. Приложения в статистике

Между геометрией и статистикой имеется тесная связь, обусловленная тем, что многомерные статистические данные можно рассматривать как точки в евклидовом пространстве. В такой постановке задачи статистики превращаются в чисто геометрические задачи. Например, в задаче о линейной регрессии требуется найти наиболее подходящую в смысле заданной нормы гиперплоскость. Некоторые задачи в теории голосования можно интерпретировать как задачи поиска $k$-мерного полушария, содержащего наибольшее число точек множества. В работе [Shamos (1976)] дан обзор геометрических методов, применяемых в статистике. В некоторых задачах статистики построение выпуклой оболочки является основным моментом, и в нескольких последующих разделах мы рассмотрим некоторые из них.

### 4.2.1. Робастное оценивание

Основной задачей в статистике является оценивание параметров некоторой популяции, таких как среднее значение, на основе лишь небольшой случайной выборки. Будем говорить, что функция $t$ является несмещенной оценкой для параметра $p$, если $E[t]=p$, т. е. математическое ожидание $t$ в точности равно $p$

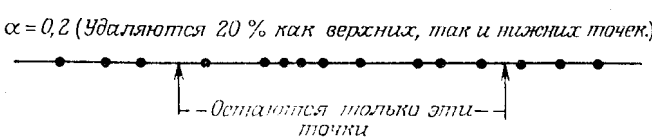



Рис. 4.12. $\alpha$-урезанное среднее.

[Hoel (1971)]. Так как среднее значение выборки является несмещенной оценкой для среднего значения популяции, то оно очень чувствительно к *выбросам* — наблюдениям, существенно выпадающим из основной массы. Желательно уменьшить влияние, оказываемое такими выбросами, так как они часто представляют неверные данные, которые иначе могли бы внести ошибки в анализ данных. Соответствующее свойство, которым должна обладать хорошая оценка, называется *робастностью* (устойчивостью), что означает нечувствительность к отклонениям от предполагаемого распределения популяции. Было предложено много методов, дающих такие оценки [Andrews (1972)]. Важный класс составляют методы оценивания, известные как методы Гэствирта [Gastwirth (1966)]. Эти методы основываются на идее, согласно которой наибольшее доверие оказывается данным, более близким к «центру» выборки.



Рассмотрим $N$ точек на прямой (рис. 4.12). Простой метод устранения предполагаемых выбросов заключается в удалении части точек с левого и правого краев — по $\alpha N$ точек с каждой стороны. Среднее значение вычисляется по оставшимся точкам. Это значение известно как $\alpha$-урезанное среднее и является частным случаем оценки по методу Гэствирта:

$$T=\sum_{i=1}^{N} w_i x_{(i)}, \quad \sum_{i=1}^{N} w_i = 1,$$

где $x_{(i)}$ обозначает $i$-й наименьший из $x_j$. Для $\alpha$-урезанного среднего имеем

$$w_i = 1/[(1-2\alpha)N], \quad \alpha N \leqslant i \leqslant (1-\alpha)N.$$

Любое урезанное среднее можно вычислить за время $O(N)$ (используя алгоритм выборки линейной сложности), а любую оценку по методу Гэствирта можно вычислить за время $O(N \log N)$ (используя сортировку). Теперь возникает вопрос: что представляет собой аналогичная процедура в случае более высокой размерности? Тьюки предложил процедуру, известную под названием «лущение» или «шелушение», состоящую в удалении выпуклой оболочки множества с последующим удалением выпуклой оболочки оставшегося множества, и так до тех пор, пока не останется лишь $(1-2\alpha)N$ точек [Huber (1972)]. Эта процедура приводит нас к следующим определению и задаче.

**Определение 4.2.** Глубиной точки $p$ в множестве $S$ называется число выпуклых оболочек (*выпуклых слоев*), которые должны быть удалены из $S$ прежде, чем будет удалена точка $p$.

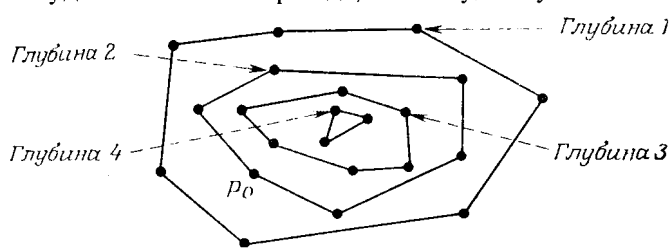



Рис. 4.13. Глубина точки в множестве.

Глубина множества $S$ — это глубина его самой глубокой точки (рис. 4.13).

В связи со сказанным возникает следующая геометрическая задача, представляющая и самостоятельный интерес:

**Задача ВО8** (ГЛУБИНА МНОЖЕСТВА). На плоскости задано множество из $N$ точек. Требуется найти глубину каждой точки.

**Теорема 4.13.** *Любой алгоритм, определяющий глубину каждой точки в множестве, в худшем случае должен выполнить* $\Omega(N \log N)$ *сравнений.*

*Доказательство.* Воспользуемся методом сведения задачи СОРТИРОВКА. Рассмотрим одномерное множество. Зная глубину каждой точки, можно упорядочить множество, выполнив дополнительно лишь $O(N)$ сравнений.

Лишь совсем недавно был разработан алгоритм, достигающий этой нижней оценки. Однако, прежде чем переходить к его описанию, интересно вкратце рассмотреть историю этой задачи.

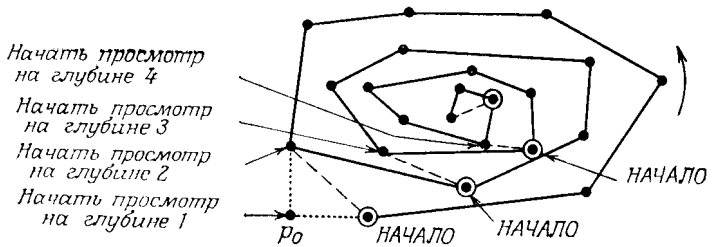



Рис. 4.14. Определение глубины точек множества путем многократного обхода по методу Джарвиса. Вершины, помечаемые последовательно меткой НАЧАЛО, обведены кружком.

Решение задачи методом «грубой силы» сводится к последовательному построению выпуклой оболочки данного множества $S$, затем выпуклой оболочки точек, не вошедших в первую выпуклую оболочку, и т. д., на что потребуется в худшем случае $O(N^2 \log N)$ операций. (Эта оценка достигается в случае, когда множество получаемых выпуклых оболочек представляет $\lfloor N/3 \rfloor$ вложенных треугольников.) В несколько более искусном подходе [Shamos (1978)] многократно используется метод обхода Джарвиса (см. разд. 3.3.3), а достигаемое при этом время работы алгоритма равно $O(N^2)$. Рис. 4.14 иллюстрирует работу этого алгоритма. Для единообразия обработки вводится дополнительная фиктивная точка $p_0 = (x_0, y_0)$, координаты которой равны минимальным значениям абсциссы и ординаты точек множества $S$. Просмотр точек множества начинается с точки $p_0$, и первая достигнутая точка маркируется меткой НАЧАЛО. Последующий обход точек выполняется в направлении против часовой стрелки. Каждая пройденная точка (за исключением НАЧАЛО) удаляется из $S$. Когда вновь достигается точка НАЧАЛО, завершается построение выпуклой оболочки глубины 1. Точка НАЧАЛО удаляется из множества $S$, а роль точки $p_0$ принимает на себя точка, предшествующая точке НАЧАЛО; этот процесс продолжается до тех пор, пока из $S$ не будут удалены все точки.



Более сложный и вместе с тем более эффективный алгоритм дает использование метода Овермарса — Ван Леювена поддержки выпуклой оболочки на плоскости (см. разд. 3.3.7). Напомним лишь, что первичной структурой данных, используемой в этом методе, является двоичное дерево поиска, узлы которого указывают на вторичные структуры данных, представляющие сцепляемые очереди. Кроме того, очередь, на которую указывает корень дерева, дает верхнюю оболочку исходного множества точек, а удаление каждой точки выполняется за время $O((\log N)^2)$. Таким образом, используя одновременно структуры, соответствующие верхней и нижней оболочкам, получаем выпуклую оболочку глубины 1. Затем каждая точка построенной оболочки удаляется из обеих структур, из которых теперь можно получить оболочку глубины 2, и т. д. Оболочки получаются тривиальным образом за линейное время в результате прохождения сцепляемой очереди, а полное время, затрачиваемое на удаление всех вершин, равно $O(N \log^2 N)$.

Впоследствии эта идея была доведена до большого совершенства Чазелле [Chazelle (1983b)]. А именно общий метод Овермарса — Ван Леювена поддержки выпуклой оболочки рассчитан на произвольную последовательность операций вставки и удаления точек. Однако при построении выпуклых слоев последовательность операций удаления полностью управляется разработчиком алгоритма. Чазелле показал, как можно надлежащим образом сгруппировать операции по удалению точек, чтобы порождать слои более эффективно. Отсылаем читателя к первоисточнику, содержащему доказательство следующей интересной теоремы:

**Теорема 4.14.** *Выпуклые слои, а следовательно, и глубина множества из* $N$ *точек на плоскости могут быть найдены за оптимальное время* $\theta(N \log N)$.

### 4.2.2. Монотонная регрессия

Задачу о регрессии можно рассматривать как задачу поиска *наилучшей аппроксимации данных в некотором подпространстве.* Предположим, задано конечное множество точек в $E^d$, рассматриваемых как значения некоторой функции $f$ от $(d-1)$ переменных, и необходимо определить допустимый класс аппроксимирующих функций и норму для измерения ошибки [1]).

[1]) В двумерном случае точки $(x_i, y_i)$ представляют функцию $y_i = f(x_i)$, и $x_i$ называется независимой переменной. В $k$-мерном случае имеем $x_k = f(x_1, \ldots, x_{k-1})$.

Функция регрессии — это некоторая функция $f^*$ от $(d-1)$ переменных, минимизирующая норму $\| f - f^* \|$.

Задача о монотонной регрессии заключается в поиске наилучшей монотонной (т. е. невозрастающей или неубывающей)

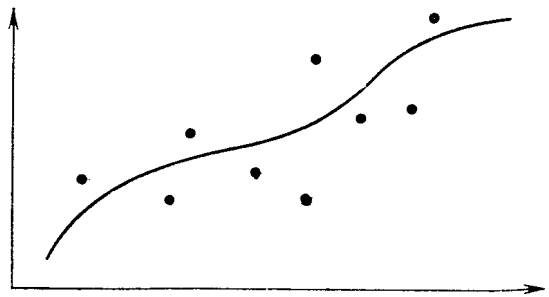


Рис. 4.15. Монотонная функция, аппроксимирующая (не лучшим образом) конечное множество точек.

аппроксимирующей функции для конечного множества точек. Обычно в качестве нормы берется норма пространства $L_2$ или минимум суммы квадратов ввиду связи последней с методом максимального правдоподобия [1]). Другими словами, задано

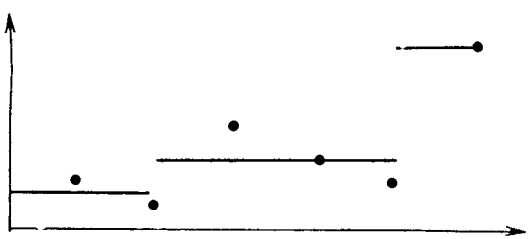


Рис. 4.16. Наилучшее соответствие обеспечивает ступенчатая функция. Для ее задания необходимо определить как число ступенек, так и точки разрыва при переходе с одной ступеньки на другую.

множество точек $\{(x_i, y_i)\colon i = 1, \ldots, N\}$ и необходимо найти монотонную функцию $f$, минимизирующую

$$\sum_{i=1}^{N} (y_i - f(x_i))^2.$$

На рис. 4.15 приведен пример монотонной аппроксимирующей функции.

[1]) Более подробно монотонная регрессия рассматривается в книге [Barlow, Bartholomew, Bremner, Brunk (1972)].



Приведенный выше рисунок вводит в некоторое заблуждение, так как он только усиливает интуитивное представление о том, что функция $f$ должна быть непрерывной. В рамках метода наименьших квадратов, когда минимизируется сумма квадратов отклонений, наилучшее соответствие дает ступенчатая функция, пример которой показан на рис. 4.16. Остается «только» определить точки перехода со ступеньки на ступеньку и высоту каждой ступеньки.

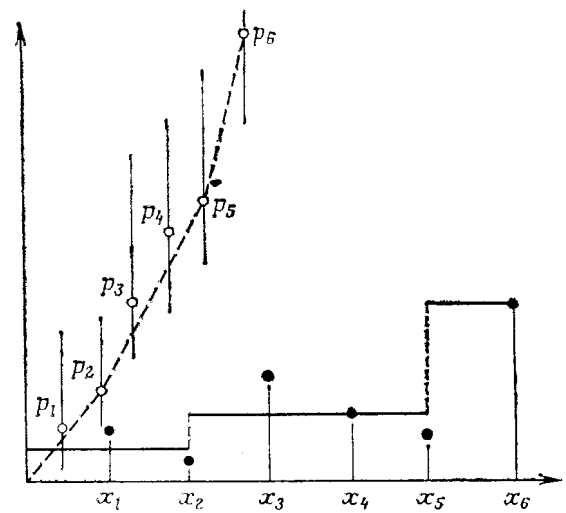



Рис. 4.17. Нижняя выпуклая оболочка ДЧС определяет монотонную аппроксимирующую функцию.

Предположим, что данные упорядочены по значению $x$-координаты. (Обычно на практике нет необходимости производить сортировку данных, так как независимой переменной является время либо точки выбираются в порядке возрастания $x$.) Определим *диаграмму частичных сумм* (ДЧС) как множество точек $p_j = (j, s_j)$, $p_0 = (0, 0)$, где $s_j$ — это частичная сумма ординат точек:

$$s_j = \sum_{i=1}^{j} y_i.$$

Наклон отрезка прямой, соединяющего точки $p_{j-1}$ на $p_j$, как раз равен $y_j$. Предположим теперь, что построена нижняя оболочка $H$ множества точек $\{p_0, p_1, \ldots, p_N\}$ (см. разд. 3.3.2). Между нижней оболочкой $H$ и монотонной функцией регрессии существует замечательная связь: монотонная функция регрессии для

множества точек определяется наклоном отрезков нижней оболочки его диаграммы частичных сумм [Barlow (1972)] (рис. 4.17).

Таким образом, имеет место следующий результат.

**Теорема 4.15.** *Монотонная функция регрессии, определяемая методом наименьших квадратов, для множества из $N$ точек на плоскости может быть найдена за время $O(N \log N)$. Если точки множества уже упорядочены по значению абсциссы, то достаточным оказывается время $O(N)$.*

Если исходные данные уже упорядочены, то, используя метод Грэхема, можно найти нижнюю оболочку за линейное время. Если же точки не упорядочены, то в худшем случае потребуется время $O(N \log N)$.

### 4.2.3. Выделение кластеров (диаметр множества точек)

Следуя работе Хартигана [Hartigan (1975)], кластеризация — это выделение *групп схожих объектов* (кластеров). Кластеризация множества заключается в разбиении его на части таким образом, чтобы минимизировать некоторую меру различия. В книге Хартигана приведен целый ряд таких мер и процедур кластеризации с их использованием. Мы рассмотрим двумерный случай, предполагая, что координаты $x$ и $y$ выбраны таким образом, что мерой различия является обычная метрика евклидова пространства. Мерой «разброса» в кластере является максимальное расстояние между двумя его произвольными точками, называемое *диаметром* кластера. Интуитивно понятно, что кластер с небольшим диаметром содержит элементы, довольно тесно связанные друг с другом, чего нельзя сказать о кластерах с большим диаметром. Приведем формулировку одной из задач кластеризации.

**Задача BO9** ($K$-КЛАСТЕРИЗАЦИЯ С МИНИМАЛЬНЫМ ДИАМЕТРОМ). На плоскости заданы $N$ точек. Требуется разбить их на $K$ кластеров $C_1, \ldots, C_K$ таким образом, чтобы максимальный из диаметров кластеров был минимальным.

Трудно представить, как можно решить эту задачу, не имея по крайней мере алгоритма определения диаметра кластера. Это приводит к следующей задаче:

**Задача BO10** (ДИАМЕТР МНОЖЕСТВА). На плоскости заданы $N$ точек. Найти две наиболее удаленные друг от друга точки.

Эта задача кажется настолько элементарной, что даже трудно представить, что с ней связаны какие-то проблемы. В конце



концов, можно непосредственно вычислить расстояние между каждой из $N(N-1)/2$ пар точек и выбрать максимальное из них, определяющее диаметр множества. Что же здесь еще изучать? Несомненно, следует задать вопрос, является ли эта процедура, имеющая сложность $O(N^2)$, наилучшим из возможных алгоритмов. Вопрос о вычислительной сложности задач такого сорта является основным в вычислительной геометрии, который мы обязаны ставить даже в случае отсутствия в нем практической надобности.

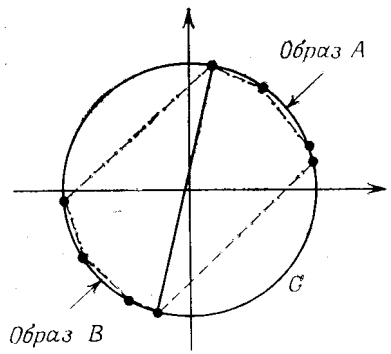


Рис. 4.18. Отображение множеств $A$ и $B$ на единичную окружность $C$.

Метод получения нетривиальной нижней оценки для задачи ДИАМЕТР МНОЖЕСТВА был разработан совсем недавно [1]). Он основан на сведении задачи РАЗДЕЛИМОСТЬ МНОЖЕСТВ, для которой известна нетривиальная нижняя оценка, к задаче ДИАМЕТР МНОЖЕСТВА. Пусть $A = \{a_1, \ldots, a_N\}$ и $B = \{b_1, \ldots, b_N\}$ — два множества действительных неотрицательных чисел. Для проверки того, что множества $A$ и $B$ не содержат одинаковых элементов (т. е. разделимость множеств $A$ и $B$), требуется выполнить $\Omega(N \log N)$ сравнений [Reingold (1972)]. Сведем задачу РАЗДЕЛИМОСТЬ МНОЖЕСТВ к ДИАМЕТР МНОЖЕСТВА, отобразив множества $A$ и $B$ на единичную окружность $C$, при этом элементы множества $A$ отображаются в первый квадрант, а элементы множества $B$ — в третий. Отображение устроено следующим образом: $a_i$ отображается в точку пересечения окружности $C$ с прямой $y = a_i x$, находящуюся в первом квадранте, а $b_i$ отображается аналогичным образом, но только в точку, находящуюся в третьем квадранте (рис. 4.18). Обозначим через $S$ множество, содержащее $2N$ построенных таким образом точек пересечения. Нетрудно понять, что множество $S$ имеет диаметр, равный 2, тогда и только тогда, когда оно содержит две диаметрально противоположные точки окружности $C$, т. е. когда $A \cap B \neq \varnothing$. Таким образом, имеет место следующая теорема, справедливость которой для пространств более высокой размерности очевидна:

---

[1]) Исходную идею преобразования можно найти в [Brown (1979a)]. Очевидно, Добкин и Манро независимо придумали одно и то же отображение.

**Теорема 4.16.** *В рамках модели алгебраических деревьев вычислений для вычисления диаметра конечного множества из $N$ точек в $E^d$ $(d \geqslant 2)$ требуется $\Omega(N \log N)$ операций.*

Вернемся к разработке алгоритма решения задачи ДИАМЕТР МНОЖЕСТВА. Теорема 4.17 дает ключевую идею, позволяющую избежать вычисления расстояния для каждой пары точек:

**Теорема 4.17** [Hocking, Young (1961)]. *Диаметр множества равен диаметру его выпуклой оболочки.* (См. рис. 4.19.)

Конечно, в худшем случае может получиться так, что все исходные точки множества окажутся вершинами выпуклой оболочки, так что, затратив $O(N \log N)$ времени, не удастся удалить ни одной точки.

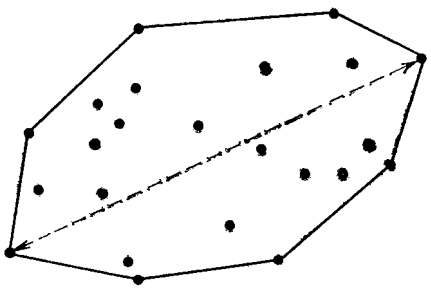


Рис. 4.19. $\mathrm{Diam}(S) = \mathrm{Diam}(\mathrm{CH}(S))$.

Далее, если не оговорено иное, будем рассматривать двумерный случай задачи. В этом случае выпуклая оболочка множества точек является выпуклым многоугольником, а вовсе не конечным множеством точек. Так что имеет место несколько иная задача.

**Задача BO11** (ДИАМЕТР ВЫПУКЛОГО МНОГОУГОЛЬНИКА). Дан выпуклый многоугольник. Найти его диаметр.

Сразу же получаем

ДИАМЕТР МНОЖЕСТВА $\propto_{N \log N}$ ДИАМЕТР ВЫПУКЛОГО МНОГОУГОЛЬНИКА

Если диаметр выпуклого многоугольника можно найти за время, меньшее чем $O(N^2)$, то и диаметр множества можно будет найти быстрее, чем за $O(N^2)$ операций.

Пока у нас нет причин заранее сомневаться в том, что свойство выпуклости поможет нам в решении задачи. Стоит продолжить исследование в этом направлении, прежде чем отказываться от этой идеи. По крайней мере следует рассмотреть несколько иные подходы к определению диаметра. Один из них дает следующая теорема.

**Теорема 4.18.** *Диаметр выпуклой фигуры равен наибольшему из расстояний между двумя параллельными опорными прямыми к этой фигуре* [Yaglom, Boltyanski (1961)].

На примере рис. 4.20 видно, что параллельные прямые можно провести не через *каждую* пару точек. Например, никакие



две опорные прямые, проходящие соответственно через вершины $p_4$ и $p_6$, не являются параллельными. Это значит, что отрезок $p_4p_6$ не является диаметром. Пара точек, через которые можно провести параллельные опорные прямые, называется *противолежащей парой*. В силу теоремы 4.18 необходимо рассматривать лишь противолежащие пары точек. Задача заключается в том, чтобы определить противолежащие пары, не перебирая все возможные пары точек.

Рассмотрим вершину $p_i$ выпуклого многоугольника $P$, вершины которого занумерованы в порядке обхода против часовой

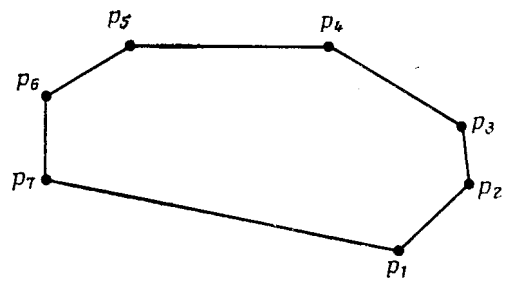



Рис. 4.20. Не все пары вершин являются противолежащими. Через вершины $p_4$ и $p_6$ нельзя провести параллельные опорные прямые. Таким образом, $p_4p_6$ не может быть диаметром.

стрелки (рис. 4.21(а)). Предположим, что мы двигаемся по границе многоугольника $P$ против часовой стрелки. Начав движение в вершине $p_i$, будем двигаться до тех пор, пока не достигнем вершины $q_R^{(i)}$, максимально удаленной от $p_{i-1}p_i$ [1]). В случае неоднозначности выбора, т. е. когда $P$ содержит параллельные ребра, $q_R^{(i)}$ — это первая вершина с заданным свойством, встретившаяся при обходе. Аналогичным образом определяется вершина $q_L^{(i)}$, максимально удаленная от $p_ip_{i+1}$, которая выявляется при обходе границы многоугольника $P$ по часовой стрелке, начиная с вершины $p_i$. Утверждается, что цепь вершин между $q_R^{(i)}$ и $q_L^{(i)}$ (включая эти вершины) дает множество $C(p_i)$, каждая вершина которого образует с $p_i$ противолежащую пару [2]). Действительно, пусть $\alpha_i$ — внешний угол, образуемый $p_{i-1}p_i$ и $p_ip_{i+1}$ $(\alpha_i > \pi)$. Ясно, что $(p_i, p_s)$ является противолежащей

[1]) Точнее сказать, ищется вершина, максимально удаленная от прямой, содержащей отрезок $p_{i-1}p_i$. — *Прим. перев.*

[2]) Вершины $q_R^{(i)}$ и $q_L^{(i)}$ разбивают границу $P$ на две цепи. К $C(p_i)$ относятся вершины той из них, которая не содержит вершину $p_i$. — *Прим. перев.*

парой тогда и только тогда, когда существует прямая, попадающая в пересечение углов $\alpha_s$ и $\alpha_i$ (рис. 4.21(b)). Так как $C(p_i)$ — это выпуклая цепь, то каждая вершина $p_s \in C(p_i)$ обладает этим свойством. Более того, для каждой вершины $P$, не принадлежащей $C(p_i)$, справедливо противоположное высказывание.

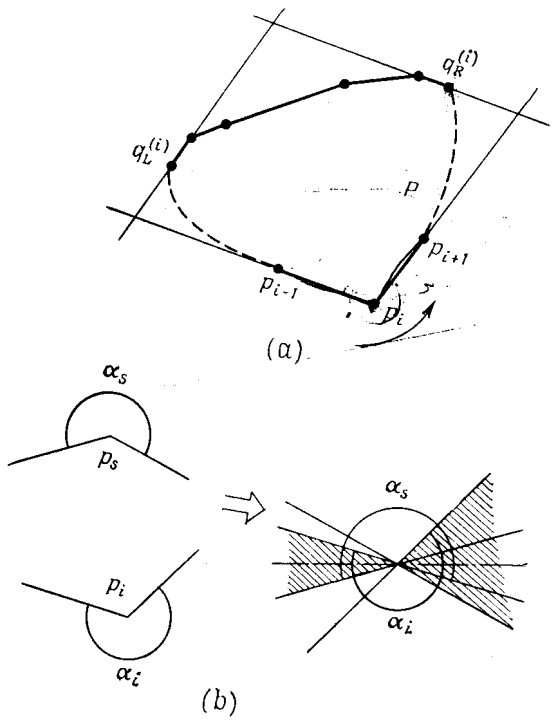



Рис. 4.21. Характеризация противолежащих пар. Пересечение углов $\alpha_i$ и $\alpha_s$ представляет собой два клина с совпадающими вершинами.

Это свойство немедленно подсказывает алгоритм порождения всех противолежащих пар. Основная операция, необходимая для этого, заключается в проверке условия максимальной удаленности точки от прямой, определяемой отрезком $\overline{p_i p_{i+1}}$. Это легко сделать, вычислив площадь (с учетом знака) треугольника $(p_i, p_{i+1}, p)$ (см. разд. 2.2.1). Теперь можно описать простой алгоритм. В этом алгоритме предполагается, что вершины многоугольника $P$, составляющие при обходе против часовой стрелки последовательность $(p_1, p_2, \ldots, p_N)$ (в которой за $p_N$ следует $p_1$), представлена списком с указателем СЛЕДУЮЩИЙ[ ], дающим следующий элемент списка.



```
      procedure ПРОТИВОЛЕЖАЩИЕ-ПАРЫ
1.    begin p := p_N;
2.          q := СЛЕДУЮЩИЙ[p];
3.          while (Площадь(p, СЛЕДУЮЩИЙ[p],
                    СЛЕДУЮЩИЙ[q]) > Площадь
                    (p, СЛЕДУЮЩИЙ[p], q)) do
                    q := СЛЕДУЮЩИЙ[q];
            (* продвигаться по границе P до тех пор, пока
            не будет достигнута вершина, максимально
            удаленная от p СЛЕДУЮЩИЙ[p] *)
4.          q_0 := q;
5.          while (q ≠ p_0) do
6.           begin p := СЛЕДУЮЩИЙ[p];
7.                 печать (p, q);
8.                 while(Площадь(p, СЛЕДУЮЩИЙ[p],
                       СЛЕДУЮЩИЙ[q])) >
                       Площадь(p, СЛЕДУЮЩИЙ[p], q)) do
9.                    begin q := СЛЕДУЮЩИЙ[q];
10.                         if((p, q) ≠ (q_0, p_0)) then печать(p, q)
                      end;
11.                if (Площадь(p, СЛЕДУЮЩИЙ[p],
                    СЛЕДУЮЩИЙ[q])) =
                    ПЛОЩАДЬ(p, СЛЕДУЮЩИЙ[p], q))
                      then
12.                         if(p, q) ≠ (q_0, p_N)) then
                            печать(p, СЛЕДУЮЩИЙ[q])
                            (* обработка параллельных ребер *)
           end
      end.
```

Рис. 4.22 иллюстрирует работу приведенного выше алгоритма. Строки 1—3 алгоритма описывают поиск вершины $q_R^{(1)}$, если использовать обозначения, введенные ранее (в алгоритме эта вершина обозначена $q_0$). Последующий цикл **while**, в котором формируется множество $C(p_i)$, использует два указателя $p$ и $q$. Эти два указателя двигаются по границе многоугольника $P$ в направлении против часовой стрелки, при этом $p$ продвигается от $p_N$ до $q_R^{(1)}$, а $q$ — от $q_R^{(1)}$ до $p_N$. Противолежащая пара порождается каждый раз, когда либо происходит продвижение любого из двух упомянутых указателей (строки 5—7 и 9—10), либо встречается пара параллельных ребер многоугольника $P$ (строки 11—12). Очевидно, что первой порожденной парой будет $(p, q) = (p_0, q_0)$ (при первом выполнении строки 7). Утверждается, что последней парой будет $(p, q) = (q_0, p_N)$, так как $p_N$ — это последнее значение, принимаемое $q$ (иначе основной цикл **while** в строке 5 завершается). Таким образом, каждая про-

тиволежащая пара порождается лишь один раз. И так как при выполнении цикла **while** указатель $p$ продвигается от $p_0$ до $q_0$, а указатель $q$ — от $q_0$ до $p_N$, то произойдет всего $N$ продвижений,

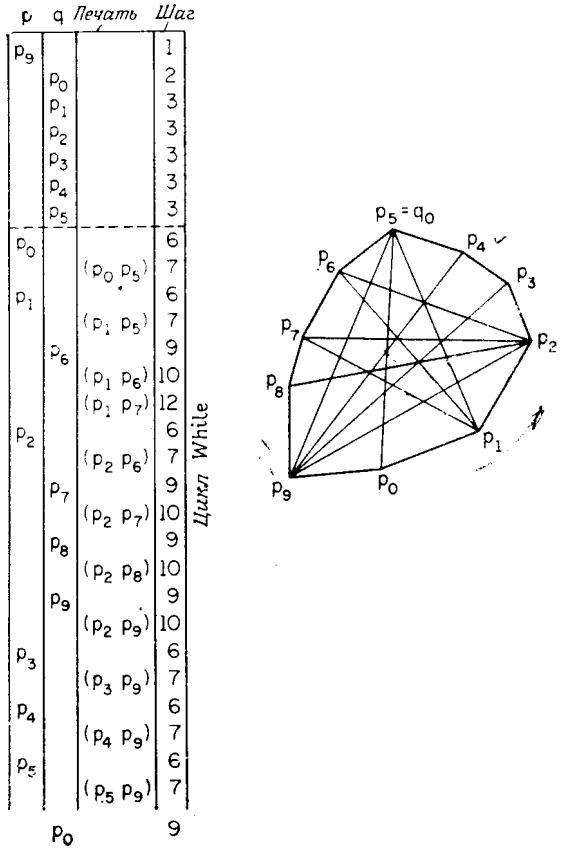






Рис. 4.22. Иллюстрация работы процедуры ПРОТИВОЛЕЖАЩИЕ-ПАРЫ. Обратите внимание на то, что ребра $p_1p_2$ и $p_6p_7$ параллельны.

и, следовательно, при отсутствии параллельных ребер будет порождено $N$ противолежащих пар. Если имеются пары параллельных ребер, то их число не может превышать $\lfloor N/2 \rfloor$, и, следовательно, полное число противолежащих пар, порождаемых алгоритмом, не превосходит $3N/2$.

Алгоритм можно изменить таким образом, что он будет выдавать лишь такие противолежащие пары (их в точности $N$), которые являются кандидатами на место диаметра. Действительно, каждый раз, когда два ребра многоугольника параллельны, необходимо рассматривать лишь диагонали определяемой ими трапеции.



Так как для определения диаметра многоугольника вполне достаточно знать все его противолежащие пары (согласно теореме 4.18), то имеет место следующий результат.

**Теорема 4.19.** *Диаметр выпуклого многоугольника может быть найден за линейное по числу его вершин время.*

**Следствие 4.1.** *Диаметр множества из $N$ точек на плоскости может быть найден за оптимальное время $\theta(N \log N)$.*

**Теорема 4.20.** *Диаметр множества из $N$ точек, выбранных из $N^p$-распределения на плоскости* (см. разд. 4.1.1), *может быть найден в среднем за время $O(N)$.*

*Доказательство.* Согласно теореме 4.5, выпуклую оболочку можно найти в среднем за линейное время. Применяя теорему 4.12, получаем необходимый результат.

Теорема 4.19 в сочетании с теоремой 4.12 (см. разд. 4.1.4) позволяют получить следующий интересный результат.

**Следствие 4.2.** *Диаметр простого многоугольника может быть найден за линейное время.*

На первый взгляд может показаться, что при прохождении *всех* противолежащих пар для определения диаметра выполняется лишняя работа. Легко убедиться, что при отсутствии параллельных ребер этого не происходит. Ключевым в этом вопросе является понятие *диаметральной пары вершин* в многоугольнике $P$, т. е. такой пары вершин в многоугольнике $P$, расстояние между которыми в точности равно диаметру множества вершин. Так как диаметральная пара является также противолежащей парой, то в множестве вершин имеется не более $N$ диаметральных пар. Классический результат, полученный Эрдёшем [Erdös (1964)], состоит в следующем:

**Теорема 4.21.** *В множестве из $N$ точек на плоскости может быть не более $N$ пар точек, на которых достигается максимальное расстояние между точками множества.*

Очевидно, что данная оценка достигается на правильных $N$-угольниках ($N \geqslant 3$, нечетное).

## 4.3. Замечания и комментарии

Оптимальное решение задачи вычисления диаметра множества точек на плоскости было получено методом, обсуждавшимся в разд. 4.2.3. Довольно соблазнительно расширить этот подход на пространства более высокой размерности. В слу-

чае пространства $E^3$ еще одним фактором, стимулирующим интерес к этому вопросу, является следующий результат [Grünbaum (1956)] (известный как гипотеза Васони):

**Теорема 4.22.** *В множестве из $N$ точек в трехмерном пространстве число пар точек, на которых достигается максимальное расстояние, не превышает* $2N - 2$.

К сожалению, большое сходство теорем 4.21 и 4.22 скрывает тот факт, что, в то время как в случае плоскости число диаметральных пар и число противолежащих пар растут как $O(N)$, в трехмерном пространстве для них имеют место оценки $O(N)$ и $O(N^2)$ соответственно. В самом деле, не составляет труда конструктивно продемонстрировать тот факт, что число противолежащих пар в трехмерном пространстве может расти как $O(N^2)$. Соответствующий пример приведен на рис. 4.23, где нужная конфигурация получается в результате замены двух несмежных ребер тетраэдра двумя «цепями» по $N/2$ вершин в каждой. Любая пара вершин — по одной вершине из каждой цепи — образует противолежащую пару. Таким образом, процедура порождения всех противолежащих пар требует времени $O(N^2)$ и, возможно, является более трудоемкой по сравнению с методом «грубой силы», основанным на вычислении расстояния между каждой парой точек. Хотя диаметральные пары составляют подмножество противолежащих пар, до сих пор не найден эффективный способ выделения таких пар. Несмотря на ее внешнюю простоту, задача вычисления диаметра множества точек в трехмерном пространстве явилась для многих исследователей преградой, о которую разбивались все их усилия.

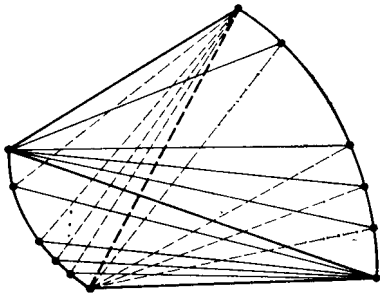


Рис. 4.23. Конструкция множества из $N$ точек в трехмерном пространстве с числом противолежащих пар, равным $O(N^2)$.

В $d$-мерном пространстве диаметр множества из $N$ точек всегда можно определить за время $O(dN^2)$, вычисляя расстояние между каждой парой точек.

Следующая теорема сужает круг возможностей сделать это более эффективно:

**Теорема 4.23** [Erdös (1960)]. *В множестве из $N$ точек в $d$-мерном ($d > 3$) пространстве число пар точек, на которых достигается максимальное расстояние, может достигать* $N^2/(2 - \lfloor d/2 \rfloor) + O(N^{2-\varepsilon})$, *где* $\varepsilon > 0$ *— некоторое число.*



Яо [Yao (1982)] предложил несколько иной подход к задаче вычисления диаметра множества. Этот подход позволил получить алгоритм со сложностью $o(N^2)$. А именно оценка времени работы алгоритма равна

$$T(N, d) = O\left(N^{2-a(d)}(\log N)^{1-a(d)}\right),$$

где $a(d) = 2^{-(d+1)}$. Для $d = 3$ эту оценку можно улучшить до $O((N \log N)^{1,8})$. Вопрос о том, можно ли уменьшить различие между $T(N, d)$ и $\Omega(N \log N)$, остается открытым.

## 4.4. Упражнения

1. Пусть $S$ — множество из $N$ точек на плоскости. Обе координаты каждой точки являются целыми числами, не превосходящими $m$ (*точки, заданные на решетке*). Разработать алгоритм построения выпуклой оболочки $\mathrm{CH}(S)$ со временем обработки $O(N + m)$.

2. Пусть $S$ — множество из $N = m^2$ точек в $E^3$, определенное следующим образом:

$$S = \{p_{ij}: x(p_{ij}) = i,\ y(p_{ij}) = j,\ z(p_{ij}) > 0,\ 0 < i,\ j \leqslant m\}.$$

Разработать специальный алгоритм построения верхней оболочки множества $S$.

3. *Проверка на максимум в $E^2$.* На плоскости задано множество $S$ из $N$ точек. Разработать метод, проверяющий за время $O(\log N)$, является ли некоторая точка $q$ максимумом множества $S \cup \{q\}$. (Предполагается режим многократных запросов — см. разд. 2.1.) Время, затрачиваемое на создание структуры данных, используемой для поиска, не должно превышать $O(N \log N)$, а занимаемый ею объем памяти — $O(N)$.

4. *Проверка на максимум в $E^3$.* В пространстве $E^3$ задано множество $S$ из $N$ точек. Разработать метод, проверяющий за время $O(\log N)$, является ли некоторая точка $q$ максимумом множества $S \cup \{q\}$. (Предполагается режим многократных запросов.) Время, затрачиваемое на создание структуры данных, используемой для поиска, не должно превышать $O(N \log N)$, а занимаемый ею объем памяти — $O(N)$. (Указание: использовать метод локализации точки в некотором подходящем разбиении плоскости.)

5. *Степень доминирования.* На плоскости задано множество $S$ из $N$ точек. *Доминирующим подмножеством* множества $S$ называется множество $\max(S)$ максимумов множества $S$. Степенью доминирования точки $p$ в множестве $S$ называется число доминирующих подмножеств, которые необходимо удалить для удаления точки $p$. Степень доминирования множества $S$ равна максимальной степени доминирования его точек.

(a) Разработать алгоритм со сложностью $O(N \log N)$ для вычисления степени доминирования каждой точки множества $S$.

(b) Оптимален ли этот алгоритм?

ГЛАВА 5

# Близость: основные алгоритмы

В главе 4 был приведен алгоритм двух наиболее удаленных друг от друга точек множества на плоскости, имеющей сложность $O(N \cdot \log N)$. Можно было бы предположить, что алгоритм поиска двух ближайших друг к другу точек множества можно получить простым обобщением указанного алгоритма, но это не так. Две наиболее удаленные точки множества с необходимостью являются вершинами выпуклой оболочки этого множества, и для разработки быстрого алгоритма можно использовать свойство выпуклости. Две ближайшие точки совсем не обязательно имеют какое-либо отношение к выпуклой оболочке, так что для решения этой задачи должен быть разработан новый метод. Этому и будет посвящена данная глава. Мы рассмотрим большой класс задач, связанных с определением близости точек на плоскости. Наша цель будет состоять в том, чтобы решать все эти не связанные на первый взгляд между собой задачи, используя единый алгоритм, который выявляет, обрабатывает и запоминает компактным образом всю информацию относительно близости точек. Чтобы добиться этого, мы возродим классический математический объект, называемый диаграммой Вороного, представив ее в виде эффективной вычислительной структуры, позволяющей добиться огромного преимущества по сравнению с лучшими из ранее известных алгоритмов. В этой главе для штурма этого большого и трудного класса задач — *задач нахождения ближайших точек* или, иначе говоря, *задач определения близости* — будут использованы некоторые обсуждавшиеся ранее методы решения геометрических задач, такие как построение выпуклой оболочки и геометрический поиск.

Как уже говорилось выше, состояние дел в этой области вычислительной геометрии не является исключением по отношению к сложившейся в настоящее время практике: в случае плоскости имеются мощные и элегантные методы, в то время как для трехмерного пространства — и тем более для пространств более высокой размерности — известно очень мало и на каждом шагу исследователя подстерегают труднопреодолимые препятствия, не сулящие ничего хорошего.



Большинство задач, изученных в предыдущих главах (за исключением задачи о наиболее удаленной паре точек), основываются исключительно на свойствах инцидентности рассматриваемых геометрических объектов. Так что соответствующие результаты остаются справедливыми при применении к объектам полной группы линейных преобразований (разд. 1.3.2). Задачи, рассматриваемые в двух последующих главах, основываются на метрических свойствах, и, таким образом, справедливость результата ограничивается группой евклидовых преобразований (группой движений твердого тела).

## 5.1. Набор задач

Мы начнем с перечисления различных на первый взгляд задач, относящихся к определению близости, но которые, как мы увидим далее, близки с точки зрения процедуры их решения.

**Задача Б.1** (БЛИЖАЙШАЯ ПАРА). На плоскости заданы N точек. Найти две из них, расстояние между которыми наименьшее [1]).

Эта задача настолько просто формулируется и настолько важна, что ее можно рассматривать как один из основных вопросов вычислительной геометрии как с точки зрения ее приложений, так и с чисто теоретической точки зрения. Например, одно из приложений этой задачи относится к системе реального времени, управляющей движением самолетов: с некоторыми упрощениями можно считать, что наибольшей опасности столкнуться подвергаются два самолета, находящиеся на наименьшем расстоянии друг от друга.

Центральный вопрос разработки алгоритма решения этой задачи состоит в том, нужно ли перебирать каждую пару точек, чтобы найти определенное таким образом минимальное расстояние. Это может быть сделано за время $O(dN^2)$ в случае $d$-мерного пространства ($d$ — произвольное целое число). В одномерном случае существует более быстрый алгоритм, использующий тот факт, что каждая пара ближайших точек должна состоять из последовательных точек множества при упорядочении его по значению координаты. Таким образом, можно упорядочить $N$ заданных действительных чисел за $O(N \log N)$ шагов, а затем

[1]) Может оказаться, что таких пар несколько. Будем считать, что для решения задачи достаточно найти одну такую пару точек.

осуществить линейный просмотр упорядоченной последовательности $(x_1, x_2, \ldots, x_N)$ за время $O(N)$, вычисляя при этом $x_{i+1} - x_i$, $i = 1, \ldots, N-1$. Как будет показано далее, этот очевидный алгоритм является оптимальным.

**Задача Б.2** (ВСЕ БЛИЖАЙШИЕ СОСЕДИ). На плоскости заданы $N$ точек. Найти ближайшего соседа для каждой точки множества.

«Ближайший сосед» — это отношение на множестве точек $S$, определяемое следующим образом: точка $b$ является ближайшим соседом точки $a$ (обозначается $a \to b$), если

$$\operatorname{dist}(a, b) = \min_{c \in S-a} \operatorname{dist}(a, c).$$

Пример графа этого отношения изображен на рис. 5.1. Заметим, что отношение «$\to$» не обязательно является симметричным, т. е. из $a \to b$ вовсе не обязательно следует, что $b \to a$. Отметим также, что точка может быть ближайшим соседом нескольких точек[1]) (к тому же отношение «$\to$» не обязательно является функцией). Решением задачи Б.2 является совокупность упорядоченных пар $(a, b)$, где $a \to b$.

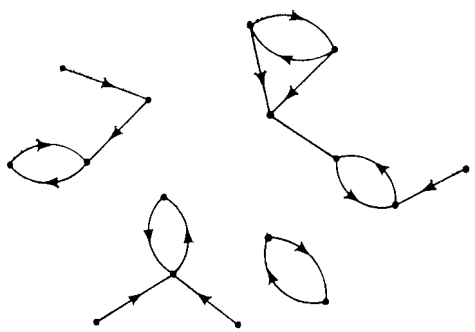


Рис. 5.1. Отношение «ближайший сосед» («$\to$») на множестве точек.

Пара точек, удовлетворяющая условию симметричности отношения (имеют место $a \to b$ и $b \to a$), называется *взаимной парой*. Если процедура выбора точек на плоскости является пуассоновским процессом, то математическое ожидание доли взаимных пар точек, согласно Пиелоу [Pielou (1977)], равно

$$6\pi/(8\pi + 3\,(3)^{1/2}) \sim 0.6215.$$

В математической экологии эта величина используется для выявления тенденции особей некоторого биологического вида к образованию изолированных пар. Для этого вычисляется фактическое число взаимных пар и получившееся потом отношение сравнивается с указанной величиной. Другие задачи, включающие вычисление ближайших соседей, возникают при исследовании расселения видов. В этих задачах, так же как в географии (см. [Kolars, Nystuen (1974)]) и в физике твердого тела, представляет интерес распределение величин расстояний между ближайшими соседями. Очевидно, что в одномерном случае тот же самый алгоритм, основанный на сортировке и решающий задачу БЛИЖАЙШАЯ ПАРА, позволяет найти все пары ближайших соседей. В дальнейшем мы рассмотрим трудности, присущие этим двум задачам в пространствах большей размерности.

[1]) Хотя точка может иметь в качестве ближайшего соседа каждую из других точек множества, она может быть ближайшим соседом самое большее шести точек в двумерном пространстве и не более двенадцати точек в трехмерном пространстве. Это максимальное число точек, для которых некоторая точка является ближайшим соседом в пространстве размерности $d$, совпадает с максимальным числом единичных сфер, которые могут быть расположены таким образом, чтобы все они касались заданной единичной сферы. (Саати [Saaty (1970)] указывает, что это число не известно для $d$, бо́льших 12.)



**Задача Б.3** (ЕВКЛИДОВО МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО). На плоскости заданы $N$ точек. Построить дерево, вершинами которого являются все заданные точки и суммарная длина всех ребер которого минимальна.

Решением задачи является список, содержащий $N-1$ пару точек. Каждая пара представляет ребро дерева. На рис. 5.2 показан пример такого дерева.

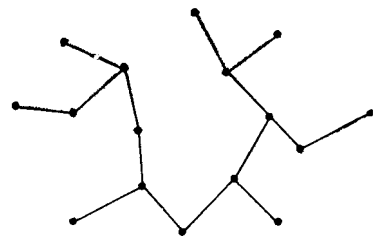


Рис. 5.2. Минимальное остовное дерево множества точек на плоскости.

Задача построения евклидова минимального остовного дерева (ЕМОД) возникает в приложениях, касающихся различного рода сетей. Если необходимо организовать систему связи между $N$ пунктами, соединив их кабелем, то использование ЕМОД даст сеть минимальной стоимости. Любопытно, что федеральный закон придает данной проблеме дополнительную значимость: федеральные тарифы требуют, чтобы тариф за пользование междугородной линией связи был пропорционален длине минимального остовного дерева, соединяющего конечные пункты абонентов. Это расстояние должно измеряться на стандартной плоской проекции поверхности Земли[1]). Это имеет место несмотря на то, что Земля не является плоской и что телефонная компания может организовать реальную сеть не обязательно в виде МОД. Тем не менее расчет за услуги основывается на решении реальной задачи построения евклидова остовного дерева, которую приходится решать сотни раз в течение дня, поскольку конфигурации телефонной сети постоянно изменяются.

[1]) Авторы благодарны Стефану Барру, сообщившему эту информацию.

Этот закон представляет «Соломонов компромисс» между тем, что требуется вычислять, и тем, что вычисляется на практике, так как минимальное остовное дерево не соответствует наикратчайшей из возможных сетей, соединяющих заданные точки, при условии, что к исходному множеству не запрещено добавлять новые точки. Если такое ограничение действительно отсутствует, то дерево, имеющее наикратчайшую длину, называется деревом Штейнера (рис. 5.3) [1]).

Как было показано Гэри, Грэхемом и Джонсоном [Garey, Graham, Johnson (1976)], задача построения дерева Штейнера является *NP*-трудной, и, используя современные средства, мы

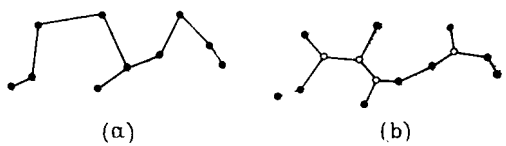



Рис. 5.3. Суммарная длина ребер дерева Штейнера (b) может быть меньше, чем у МОД (a).

не в состоянии решать задачи более чем для 20—25 точек. Поэтому неразумно требовать, чтобы тариф за использование линий связи вычислялся в соответствии с деревом Штейнера.

Минимальное остовное дерево используется при кластеризации [Cower, Ross (1969), Johnson (1967), Zahn (1971)], при вычислении характерной размерности множеств точек [Schwartzmann, Vidal (1975)] и при распознавании образов [Osteen, Lin (1974)]. Оно также использовалось для минимизации длины проводников при компоновке электрической схемы ЭВМ [Lobermann, Weinberger (1957)]. Минимальное остовное дерево дает начальное приближение для многих алгоритмов решения задачи о коммивояжере, которая обсуждается в разд. 6.1.

Задача о минимальном остовном дереве обычно формулируется как задача теории графов: задан взвешенный граф с $N$ вершинами и $E$ ребрами, требуется найти кратчайшее поддерево графа $G$, включающее все его вершины. Эта задача независимо была решена Дейкстрой [Dijkstra (1959)], Краскалом [Kruskal (1956)] и Примом [Prim (1957)], а существование полиномиального алгоритма (который был представлен каждым из них) является в значительной степени неожиданным, так как граф с $N$ вершинами может иметь $N^{N-2}$ остовных деревьев [Moon (1967)] [2]). В попытке найти быстрый алгоритм для этой

[1]) Дерево Штейнера на рис. 5.3 взято из работы [Melzak (1973)].
[2]) Впервые это было доказано Кэли в 1889 г.



общей задачи было затрачено много усилий [Nijenhuis, Wilf (1975), Yao (1975)] и лучший на сегодняшний день результат состоит в том, что задача может быть решена за время $O(E)$, если $E > N^{1+\varepsilon}$ [Cheriton, Tarjan (1976)]. (См. также разд. 6.1.)

В случае евклидовой задачи о минимальном остовном дереве $N$ вершин определяются $2N$ координатами точек на плоскости, а соответствующий граф содержит ребра, соединяющие каждую пару вершин. Вес ребра равен расстоянию между точками, соединяемыми ребром. В этом случае использование лучшего из известных алгоритмов решения задачи о МОД потребует времени $\theta(E) = \theta(N^2)$. И так как МОД *всегда содержит самое короткое ребро графа* $G$ [1]), то легко доказать, что приведенное значение является нижней оценкой для произвольного графа. Действительно, так как в произвольном графе значения весов ребер ничем не ограничены, то алгоритм решения задачи о МОД, требующий времени меньше $O(N^2)$, мог бы быть использован для поиска минимума среди $N^2$ значений за время $O(N^2)$, что невозможно. Из этого следует, что любой алгоритм, рассматривающий решение задачи о евклидовом минимальном остовном дереве как выделение минимального остовного дерева в полном графе с $N$ вершинами, с необходимостью имеет квадратичную сложность по времени. Какие же соображения заставляют нас предполагать, что для решения задачи достаточно меньшего времени? Прежде всего, евклидов вариант задачи о МОД имеет лишь $2N$ входных величин (координат точек), в то время как в случае произвольного графа имеется $N(N-1)/2$ входных величин (длин ребер). Таким образом, евклидов вариант задачи содержит сильные ограничения по сравнению с общим случаем, и при разработке быстрого алгоритма можно было бы использовать метрические свойства.

**Задача Б.4** (ТРИАНГУЛЯЦИЯ). На плоскости заданы $N$ точек. Соединить их непересекающимися отрезками таким образом, чтобы каждая область внутри выпуклой оболочки этого множества точек являлась треугольником. (См. разд. 1.3.1.)

Граф триангуляции множества из $N$ точек, являясь планарным, имеет не более $3N-6$ ребер. Результатом решения сформулированной выше задачи должен быть по крайней мере список этих ребер. На рис. 5.4 приведен пример триангуляции.

Задача триангуляции возникает в методе конечных элементов [Strang, Fix (1973); Cavendish (1974)] и при интерполяции функции от двух переменных, когда заданы значения функции в $N$ произвольным образом расположенных точках $(x_i, y_i)$ и требуется аппроксимировать ее в некоторой новой точке $(x, y)$.

[1]) Это показали Краскал и Прим.

Один из возможных подходов основывается на кусочно-линейной интерполяции, при которой поверхность, определяемая функцией, аппроксимируется «сетью», состоящей из плоских треугольных граней. Проекция каждой точки $(x, y)$ принадлежит лишь одной из треугольных граней, и соответствующее значение функции $f(x, y)$ вычисляется в результате определения интерполирующей плоскости, проходящей через три вершины грани. Процесс триангуляции заключается в выборе троек точек, которые и будут образовывать грани. Для определения «качества» получаемой триангуляции было предложено немало критериев [George (1971)], включающих, в частности, максимизацию наименьшего угла или минимизацию полной длины ребер. Выбор указанных условий объясняется их удобством для получения оценки ошибки интерполяции, а не тем, что они позволяют получить наилучшую триангуляцию.

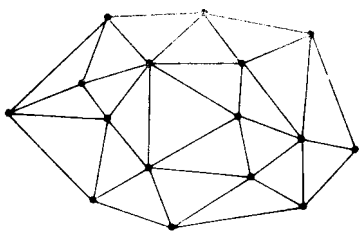


Рис. 5.4. Триангуляция множества точек.

Все четыре предшествующие задачи (Б.1—Б.4) относятся к числу задач, решение которых ищется лишь один раз и состоит в построении некоторого геометрического объекта (ближайшей пары точек, всех ближайших соседей, евклидова минимального остовного дерева и триангуляции). Далее рассматриваются две задачи поиска (см. гл. 2), которые приходится решать многократно и которые вследствие этого допускают предварительную доработку исходных данных.

**Задача Б.5** (ПОИСК БЛИЖАЙШЕГО СОСЕДА). На плоскости заданы $N$ точек. Как быстро можно найти ближайшего соседа для некоторой новой точки $q$, при условии что допускается предварительная обработка [Knuth (1973)]?

В пространстве размерности $d$ эту задачу можно решить за время $O(dN)$. Но нас интересует, как, используя предварительную обработку, сделать процедуру поиска более быстрой. Существует множество прикладных задач, в которых может быть использован такой быстрый поиск. Наиболее важной из них, по-видимому, является задача классификации. Один из методов классификации основывается на правиле ближайшего соседа [Duda, Hart (1973)]. В соответствии с этим правилом, если необходимо классифицировать объект на принадлежность к одному из заданных классов, то его следует отнести к классу, соответствующему его ближайшему соседу. Например, неизвестная точка $U$ на рис. 5.5 была бы классифицирована как принадлежащая классу $B$.



Похожая ситуация имеет место в информационных системах при выборке информации, когда выбирается запись, наилучшим образом соответствующая записи-запросу [Burkhard, Keller (1973), Friedman, Bentley, Finkel (1977)]. В случае, когда требуется обрабатывать большое количество объектов, решая либо задачи классификации [Duda, Hart (1973)] (при распознавании речи, идентификации элементарных частиц и т. п.), либо задачи

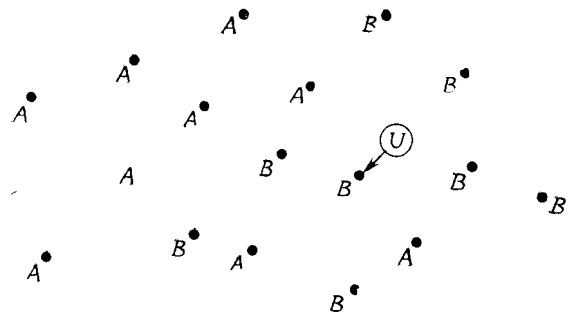



Рис. 5.5. Правило ближайшего соседа.

выборки данных (выборка наиболее подходящего элемента), необходимо достаточно быстро выполнять поиск ближайшего соседа.

**Задача Б.6** ($k$-БЛИЖАЙШИХ СОСЕДЕЙ). На плоскости заданы $N$ точек. Как быстро можно найти $k$ точек, ближайших к некоторой новой точке $q$, при условии что допускается предварительная обработка?

Процедура поиска $k$ ближайших соседей использовалась при интерполяции и выделении контуров [Davis (1975)] и при классификации (правило, использующее $k$ ближайших соседей, является более устойчивым ко всякого рода случайностям по сравнению с тем, когда решение принимается с учетом лишь единственного соседа). Хотя последняя задача представляется более сложной, чем задача Б.5, далее будет показано, что обе они могут быть решены с использованием аналогичных геометрических структур (разд. 6.3.1).

## 5.2. Задача о единственности элементов

Изучая вычислительную сложность некоторой задачи, т. е. получая нижние оценки для некоторых важных параметров, таких как время, необходимое для решения задачи, и используемый объем памяти, мы часто пытаемся преобразовать (или,

иначе говоря, свести) некоторую хорошо изученную задачу, для которой известны нетривиальные нижние оценки, к рассматриваемой (разд. 1.4).

Примеры такого подхода стали теперь классическими, и мы уже сталкивались с ними при изучении вычислительной геометрии. Напомним хотя бы о сведении задачи сортировки (в рамках модели деревьев вычислений) к задаче построения упорядоченной выпуклой оболочки множества точек. Другим известным примером является задача ВЫПОЛНИМОСТЬ, часто используемая при доказательстве $NP$-полноты задач [Garey, Johnson (1982)]. Вполне естественно называть такие фундаментальные задачи, играющие роль базовых для классов задач, задачами-прототипами или *вычислительными прототипами*.

Для первых четырех задач, представленных в предыдущем разделе, довольно легко разработать прямые алгоритмы с полиномиальной оценкой для времени. И поэтому вполне естественно попробовать в качестве возможного прототипа задачу сортировки. Хотя мы не можем исключить возможность того, что сортировка может оказаться подходящим прототипом, но до сих пор для всех четырех задач Б.1—Б.4 соответствующее преобразование не найдено. Однако, к счастью, можно использовать другой прототип — ЕДИНСТВЕННОСТЬ ЭЛЕМЕНТОВ [Dobkin, Lipton (1979)]. Эта задача формулируется следующим образом.

**Задача** (ЕДИНСТВЕННОСТЬ ЭЛЕМЕНТОВ). Даны $N$ действительных чисел. Определить, имеются ли среди них хотя бы два равных.

Получим для задачи ЕДИНСТВЕННОСТЬ ЭЛЕМЕНТОВ нижнюю оценку временной сложности в рамках модели алгебраических деревьев решений.

Множество $\{x_1, \ldots, x_N\}$ из $N$ действительных чисел можно рассматривать как точку $(x_1, \ldots, x_N)$ в $E^N$. Используя терминологию, введенную в разд. 1.4, обозначим через $W \subseteq E^N$ множество истинности для задачи ЕДИНСТВЕННОСТЬ ЭЛЕМЕНТОВ на множестве $\{x_1, \ldots, x_N\}$ (т. е. $W$ содержит все точки, любая пара координат которых различна). Утверждается, что $W$ имеет $N!$ компонент связности. Действительно, любой перестановке $\pi$ множества $\{1, 2, \ldots, N_N\}$ соответствует множество точек в $E^N$

$$W_\pi = \{(x_1, \ldots x_N) : x_{\pi(1)} < x_{\pi(2)} < \ldots < x_{\pi(N)}\}.$$

Ясно, что

$$W = \bigcup_{\text{по всем } \pi} W_\pi$$

и при этом $W_\pi$ являются компонентами связности, а $\#(W) = N!$. Отсюда как следствие теоремы 1.2 получаем



**Следствие 5.1.** *В рамках модели алгебраических деревьев вычислений любой алгоритм, определяющий, являются ли элементы множества из $N$ действительных чисел различными, требует $\Omega(N \log N)$ проверок.*

Этот важный результат будет использован в следующем разделе.

**Замечание.** В вычислительной геометрии используются три важные задачи-прототипа, имеющие небольшую сложность $\Omega(N \log N)$. Это — сортировка, определение крайних точек и проверка единственности элементов множества. Сведение этих задач друг к другу — дело, далеко не простое. Но все они обладают одной общей и важной чертой: нижняя оценка сложности этих задач получена в результате анализа мощности множества перестановок из $N$ элементов (симметрической группы $S_N$). И хотя довольно соблазнительно найти общего «предка» этих задач-прототипов — нечто вроде ИДЕНТИФИКАЦИЯ ПЕРЕСТАНОВКИ, — но мы не видим пути, ведущего к естественной формулировке подобной задачи.

## 5.3. Нижние оценки

Как обычно, прежде чем приступить к изучению алгоритмов решения задач, перечисленных в разд. 5.1, обсудим их сложность. Начнем с задач поиска ПОИСК БЛИЖАЙШЕГО СОСЕДА и $k$-БЛИЖАЙШИХ СОСЕДЕЙ.

В качестве вычислительного прототипа возьмем задачу ДВОИЧНЫЙ ПОИСК. Легко можно показать, что

ДВОИЧНЫЙ ПОИСК $\propto_{O(1)}$ БЛИЖАЙШИЙ СОСЕД.

Пусть заданы $N$ действительных чисел $x_1, x_2, \ldots, x_N$. В результате двоичного поиска определяется число $x_i$, ближайшее к числу $q$, задаваемому в запросе на поиск. (При этом допускается предварительная обработка!) Но эту же самую задачу можно представить и в геометрической формулировке, сопоставив каждому числу $x_i$ точку на плоскости с координатами $(x_i, 0)$. И таким образом, поиск ближайшего соседа приведет к тому же ответу, что и двоичный поиск. Используя один из основных фактов теории информации, получаем следующую теорему:

**Теорема 5.1.** *Для поиска ближайшего соседа точки в пространстве произвольной размерности необходимо выполнить $\Omega(\log N)$ сравнений (в худшем случае).*

Если, оставаясь в рамках модели деревьев решений, предположить, что число $q$ с равной вероятностью может принадлежать любому из $N + 1$ интервалов, определяемых числами $x_i$, то

теорема 5.1 дает оценку поведения в среднем для любого алгоритма поиска ближайшего соседа.

Что касается задачи $k$-БЛИЖАЙШИХ СОСЕДЕЙ, то, положив $k = 1$, немедленно получаем сведение ПОИСК БЛИЖАЙШЕГО СОСЕДА $\propto$ $k$-БЛИЖАЙШИХ СОСЕДЕЙ. Таким образом, теорема 5.1 применима и к задаче $k$-БЛИЖАЙШИХ СОСЕДЕЙ.

Разобравшись с задачами поиска, обратимся теперь к задачам Б.1—Б.4. На рис. 5.6 показана диаграмма сводимости задач (на диаграмме символ $\propto$ заменен стрелкой).

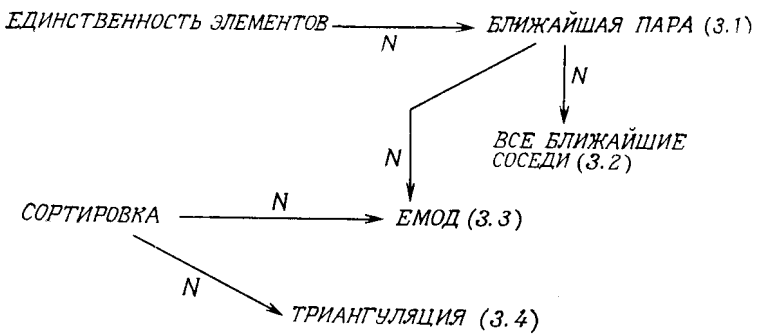



Рис. 5.6. Связь задач о близости с основными задачами, используемыми в качестве вычислительных прототипов.

Начнем с рассмотрения сводимости ЕДИНСТВЕННОСТЬ ЭЛЕМЕНТОВ $\propto_N$ БЛИЖАЙШАЯ ПАРА. Пусть задано множество действительных чисел $\{x_1, x_2, \ldots, x_N\}$. Будем рассматривать их как точки на прямой $y = 0$, пытаясь найти ближайшую пару точек. Если расстояние между точками, образующими ближайшую пару, не равно нулю, то все точки множества различны. Так как множество точек, заданное в одномерном пространстве, всегда можно вложить в пространство размерности $k$, то естественным образом получаем обобщение этого сведения.

Сведение БЛИЖАЙШАЯ ПАРА $\propto_N$ ВСЕ БЛИЖАЙШИЕ СОСЕДИ очевидно, так как одна из пар, полученных в результате решения последней задачи, будет ближайшей и она может быть определена с помощью $O(N)$ сравнений.

Рассмотрим теперь задачу ЕМОД (Б.3). В предыдущем разделе уже говорилось о том, что ЕМОД содержит кратчайшее ребро евклидова графа на множестве из $N$ точек, и, следовательно, задача БЛИЖАЙШАЯ ПАРА тривиальным образом сводится к ЕМОД за линейное время. Однако можно также показать, что

СОРТИРОВКА $\propto_N$ ЕМОД.



Действительно, рассмотрим множество из $N$ действительных чисел $\{x_1, \ldots, x_N\}$. Рассматривая каждое число $x_i$ как точку $(x_i, 0)$ на плоскости, построим соответствующее множество ЕМОД. В получившемся ЕМОД вершины, соответствующие числам $x_i$ и $x_j$, соединены ребром тогда и только тогда, когда $x_i$ и $x_j$ образуют пару последовательных чисел в упорядоченном множестве. Решением задачи ЕМОД является список, содержащий $N - 1$ пар $(i, j)$, каждая из которых определяет ребро дерева. Не составляет труда преобразовать этот список в упорядоченный список чисел $x_i$, затратив на это время $O(N)$. (Читателю предоставляется сделать это в качестве простого упражнения.)

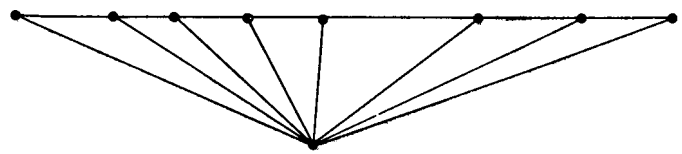


Рис. 5.7. Иллюстрация к получению нижней оценки сложности решения задачи ТРИАНГУЛЯЦИЯ.

И наконец, рассмотрим задачу ТРИАНГУЛЯЦИЯ (Б.4) и покажем, что

СОРТИРОВКА $\propto_N$ ТРИАНГУЛЯЦИЯ.

Рассмотрим множество из $N$ точек $\{x_1, x_2, \ldots, x_N\}$, показанное на рис. 5.7. $N - 1$ точек множества лежат на одной прямой, а одна точка находится вне этой прямой. Триангуляция этого множества может быть выполнена единственным способом, как это показано на рисунке. Список ребер, порождаемый алгоритмом триангуляции, можно использовать для получения упорядоченного списка чисел $x_i$, затратив на это дополнительно $O(N)$ операций. Таким образом, необходимо сделать $\Omega(N \log N)$ сравнений [1]).

Проведенный анализ позволил установить сводимость задач, представленную на рис. 5.6. Учитывая, что в рамках модели деревьев вычислений обе задачи ЕДИНСТВЕННОСТЬ ЭЛЕМЕНТОВ и СОРТИРОВКА — на множестве из $N$ элементов имеют нижнюю оценку сложности $\Omega(N \log N)$, имеет место следующая теорема.

**Теорема 5.2.** *В рамках модели деревьев вычислений любой алгоритм, решающий одну из задач* — БЛИЖАЙШАЯ ПАРА,

[1]) Эквивалентное сведение — УПОРЯДОЧЕННАЯ ОБОЛОЧКА $\propto_N$ ТРИАНГУЛЯЦИЯ — основывается на том, что триангуляция множества $S$ является планарным графом (уложенным на плоскости), внешняя граница которого является выпуклой оболочкой множества $S$.

ВСЕ БЛИЖАЙШИЕ СОСЕДИ, ЕМОД, ТРИАНГУЛЯЦИЯ, — *требует* $\Omega(N \log N)$ *операций.*

В последующих разделах мы займемся исследованием алгоритмов решения этих задач, стараясь найти оптимальные алгоритмы.

## 5.4. Решение задачи о ближайшей паре методом «разделяй и властвуй»

Нижняя оценка, даваемая теоремой 5.2, стимулирует нас на поиск алгоритма, решающего задачу БЛИЖАЙШАЯ ПАРА и имеющего сложность $\Omega(N \log N)$. Для достижения этой цели представляются разумными два пути, которые и стоит попробовать: непосредственное использование сортировки и применение метода «разделяй и властвуй». Можно сразу

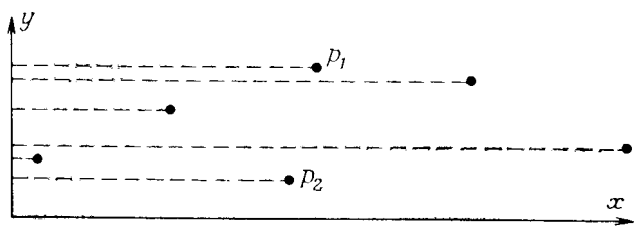



Рис. 5.8. Пример, показывающий неприменимость метода, основанного на проецировании. Точки $p_1$ и $p_2$, образующие ближайшую пару, имеют наибольшее расстояние по $y$-координате.

же отвергнуть первый поход, так как сортировка оказывается полезной лишь в условиях наличия полной упорядоченности. Единственный разумный способ получить полное упорядочение заключается, по-видимому, в проецировании всех точек на некоторую прямую. Но, к сожалению, при проецировании теряется существенная в данном случае информация. Это демонстрирует рис. 5.8, на котором точки $p_1$ и $p_2$ образуют ближайшую пару, но при этом дают максимальное расстояние при проецировании на ось $y$.

Второй путь к достижению сложности $\theta(N \log N)$ заключается в разбиении задачи на две подзадачи, решения которых можно объединить за линейное время, получая решение исходной задачи [Bentley, Shamos (1976); Shamos (1980)]. Непосредственное применение метода «разделяй и властвуй» в данном случае не принесет успеха, и довольно поучительно разобраться в причинах неудачи. Хорошо было бы разбить множество на два подмножества $S_1$ и $S_2$ по $N/2$ точек в каждом и рекурсивно найти в каждом из них ближайшую пару точек. Но возникает вопрос:



как использовать полученную информацию? Не исключена возможность, что одна из точек, образующих ближайшую пару, принадлежит подмножеству $S_1$, а вторая — $S_2$. При этом не видно способа, позволяющего избежать $N^2/4$ дополнительных сравнений. Обозначим через $P(N, 2)$ время работы алгоритма, ищущего ближайшую пару точек на плоскости. Предыдущие рассуждения дают следующее рекуррентное соотношение:

$$P(N, 2) = 2P(N/2, 2) + O(N^2).$$

Решением этого соотношения является $P(N, 2) = O(N^2)$. Попытаемся внести некоторые изменения, чтобы преодолеть возникшее затруднение. Для этого рассмотрим одномерный случай.

В одномерном случае известен лишь единственный алгоритм, решающий эту задачу и имеющий сложность $\theta(N \log N)$. Этот

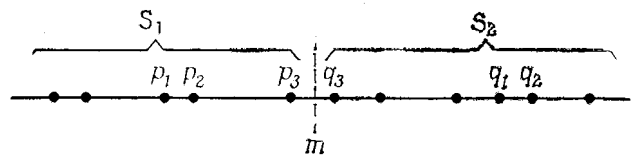



Рис. 5.9. Метод «разделяй и властвуй» в одномерном случае.

алгоритм упорядочивает точки множества, а затем просматривает его за линейное время. Поскольку, как уже отмечалось выше, сортировка не допускает обобщения на двумерный случай, попытаемся разработать для одномерного случая метод типа «разделяй и властвуй», допускающий обобщение на двумерный случай. Предположим, что точка $m$ разбивает множество на два подмножества $S_1$ и $S_2$ и при этом $p < q$ для всех $p \in S_1$ и $q \in S_2$. Решив раздельно рекурсивным образом задачу о ближайшей паре для множеств $S_1$ и $S_2$, получим две пары точек $\{p_1, p_2\}$ и $\{q_1, q_2\}$, представляющие ближайшие пары для $S_1$ и $S_2$ соответственно. Обозначим $\delta$ наименьшее расстояние, найденное на текущий момент (рис. 5.9):

$$\delta = \min(|p_2 - p_1|, |q_2 - q_1|).$$

Ближайшей парой во всем множестве является либо $\{p_1, p_2\}$, либо $\{q_1, q_2\}$, либо $\{p_3, q_3\}$, где $p_3 \in S_1$, $q_3 \in S_2$. Обратим внимание на следующий ключевой момент: для того чтобы расстояние, даваемое парой $\{p_3, q_3\}$, было меньше $\delta$, $p_3$ и $q_3$ должны находиться на расстоянии, не превышающем $\delta$ от точки $m$. (Ясно, что $p_3$ должна быть самой правой точкой множества $S_1$, а $q_3$ — самой левой точкой множества $S_2$, но такая характеристика точек не имеет особого смысла в пространствах более высокой размерности, и поэтому мы предпочли нечто более об-

щее.) Сколько точек множества $S_1$ могут находиться в интервале $(m-\delta, m]$? Так как каждый полуоткрытый интервал длиной $\delta$ содержит не более одной точки множества $S_1$, то интервал $(m-\delta, m]$ содержит не более одной точки. Аналогично интервал $[m, m+\delta)$ содержит самое большее одну точку. Таким образом, число сравнений, которые необходимо сделать для точек, выбираемых из различных подмножеств, не превышает одного. Очевидно, что все точки, попадающие в интервалы $(m-\delta, m]$ и $[m, m+\delta)$, можно определить, просмотрев множество за линейное время. Таким образом, получаем следующий алгоритм, имеющий сложность $O(N \log N)$:

**function** БПАРА1($S$)

Входные данные: X[1: $N$], $N$ точек множества $S$ в одномерном пространстве.

Выходные данные: $\delta$ — расстояние между ближайшей парой точек.

```
begin if (|S|=2)then δ:=|X[2]−X[1]|
      else if(|S|=1) then δ:=∞
           else begin m:=Медиана(S);
                Построить(S1,S2)(* S1={p : p≤m},
                                   S2={p : p>m} *);
                δ1:=БПАРА1(S1);
                δ2:=БПАРА1(S2);
                p:=max(S1);
                q:=min(S2);
                δ:=min(δ1,δ2,q−p)
           end;
     return δ
end.
```

Хотя этот алгоритм, очевидно, сложнее алгоритма, использующего сортировку с последующим просмотром, но он обеспечивает искомый переход к двумерному случаю.

Обобщение на двумерный случай можно выполнить самым непосредственным способом. Разобьем множество точек на плоскости $S$ на два подмножества $S_1$ и $S_2$ так, чтобы каждая точка $S_1$ лежала левее любой точки $S_2$. То есть множество разбивается на части вертикальной прямой $l$, определяемой медианой множества $S$ по $x$-координате. Решив рекурсивно задачу для $S_1$ и $S_2$, получим числа $\delta_1$ и $\delta_2$ — минимальные расстояния для множеств $S_1$ и $S_2$ соответственно. Положим $\delta = \min(\delta_1, \delta_2)$. (См. рис. 5.10.)

Если ближайшую пару образуют точки $p \in S_1$ и $q \in S_2$, то, очевидно, расстояния от $p$ и $q$ до $l$ не превышают $\delta$. Таким образом, если обозначить через $P_1$ и $P_2$ две вертикальные полосы



шириной $\delta$, расположенные соответственно слева и справа от $l$, то $p \in P_1$ и $q \in P_2$. Здесь возникает затруднение, которого не было в одномерном случае. На прямой было не более одного

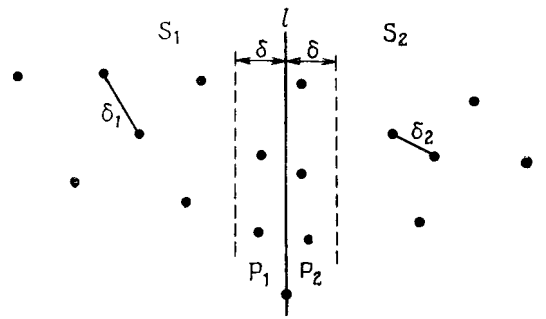



Рис. 5.10. Метод «разделяй и властвуй» в случае плоскости.

кандидата [1]) для $p$ и $q$. На плоскости таким кандидатом может быть любая точка, если она находится на расстоянии, не превышающем $\delta$, от прямой $l$. На рис. 5.11 приведен пример такого

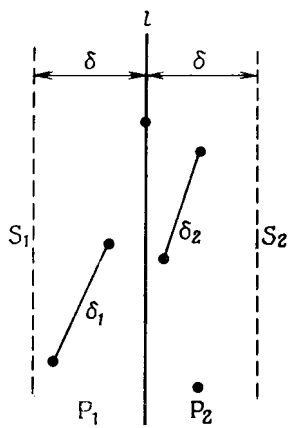



Рис. 5.11. Все точки могут находиться на расстоянии, не превышающем $\delta$ от прямой $l$.

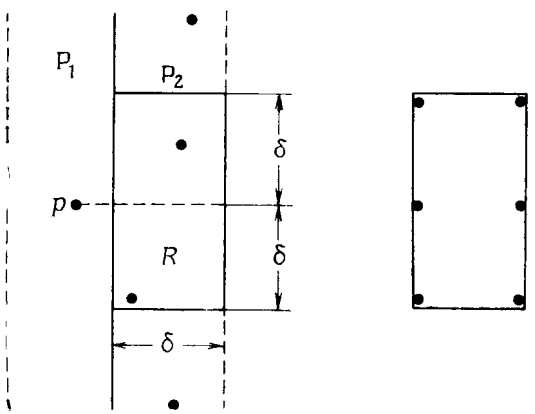



Рис. 5.12. Для каждой точки, лежащей в $P_1$, необходимо проверить не более шести точек из $P_2$.

множества. Может показаться, что для определения ближайшей пары вновь потребуется выполнить $N^2/4$ сравнений расстояний между точками. Но мы сейчас покажем, что в действительности

[1]) В процедуре БПАРА1 имеется в точности один кандидат для $p$: $p = \max(S_1)$.

точки, попадающие в полосы шириной $\delta$, расположенные по обе стороны от прямой $l$, обладают особой структурой.

Обращаясь к рис. 5.12, рассмотрим в полосе $P_1$ произвольную точку $p$. Мы должны найти все точки $q$ в $P_2$, удаленные от $p$ не более чем на $\delta$. Но сколько может быть таких точек? Все они должны находиться в прямоугольнике $R$ размером $\delta \times 2\delta$. Кроме того, известно, что никакая пара точек в $P_2$ не может находиться на расстоянии, меньшем $\delta$ друг от друга [1]). Как показано на рисунке, максимальное число точек, которые можно поместить в такой прямоугольник так, чтобы расстояние между ними было не меньше $\delta$, равно шести. Это значит, что для каждой точки из $P_1$ необходимо исследовать лишь не более шести точек из $P_2$, а вовсе не $N/2$ точек. Следовательно, на шаге слияния решений подзадач нужно выполнить не более $6 \times N/2 = 3N$ сравнений расстояний вместо $N^2/4$.

Но пока мы еще не получили алгоритм со сложностью $O(N \log N)$, так как хотя и известно, что для каждой точки из $P_1$ необходимо исследовать лишь шесть точек из $P_2$, но не известно, какие именно точки нужно исследовать! Чтобы ответить на этот вопрос, предположим, что точка $p$ и все точки из $P_2$ спроецированы на прямую $l$. Для определения точек из $P_2$, попадающих в $R$, можно рассмотреть лишь проекции точек, находящихся на расстоянии не более $\delta$ от проекции точки $p$ (их не более шести). Если точки упорядочены по $y$-координате, то для всех точек из $P_1$ «кандидаты» на место их ближайшего соседа из $P_2$ могут быть найдены за один проход этого упорядоченного списка. Ниже дан набросок алгоритма в том виде, как он разработан на данный момент.

**procedure** БПАРА2($S$)

1. Разбить $S$ на два подмножества $S_1$ и $S_2$ вертикальной прямой $l$, делящей множество пополам.
2. Рекурсивно найти расстояния для ближайших пар $\delta_1$ и $\delta_2$.
3. $\delta := \min(\delta_1, \delta_2)$.
4. Пусть $P_1$ — множество точек из $S_1$, лежащих в полосе на удалении $\delta$ от разделяющей прямой $l$, а $P_2$ — аналогичное подмножество в $S_2$. Спроецировать $P_1$ и $P_2$ на $l$ и упорядочить проекции по $y$-координате. Пусть $P_1^*$ и $P_2^*$ — соответствующие упорядоченные последовательности.
5. «Слияние» можно выполнить, просматривая $P_1^*$ и для каждой точки из $P_1^*$, изучая точки из $P_2^*$, находящиеся на расстоянии, не превышающем $\delta$. Пока указатель продвигается по последовательности $P_1^*$, указатель на $P_2^*$ может перемещаться вперед-назад, оставаясь в интервале шириной $2\delta$. Пусть $\delta_l$ — минимальное расстояние между парой точек, найденное в ходе этой процедуры.
6. $\delta_S := \min(\delta, \delta_l)$.

[1]) Это принципиальное соображение принадлежит Стронгу (частное сообщение, 1974).



Если обозначить через $T(N)$ время обработки алгоритмом множества из $N$ точек, то время, затрачиваемое на обработку на шагах 1 и 5, равно $O(N)$, на шагах 3 и 6 затрачивается постоянное время, а шаг 2 требует времени $2T(N/2)$. Если бы сортировку нужно было производить при каждом выполнении шага 4, то на это требовалось бы время $O(N \log N)$. Однако можно прибегнуть к стандартному приему, называемому *предварительной сортировкой*, когда один раз создается упорядоченный по $y$-координате список всех точек, а затем при выполнении шага 4 из этого списка за время $O(N)$ выделяются в упорядоченном виде необходимые точки [1]). Эта небольшая хитрость позволяет записать для времени обработки $P(N, 2)$ алгоритма поиска ближайшей пары в двумерном случае следующее рекуррентное соотношение:

$$P(N, 2) = 2P(N/2, 2) + O(N) = O(N \log N). \tag{5.1}$$

На основании этого соотношения получаем теорему 5.3:

**Теорема 5.3.** *Кратчайшее расстояние, определяемое $N$ точками на плоскости, может быть найдено за время $\theta(N \log N)$ и является оптимальным.*

Главные особенности описанного подхода на основе метода «разделяй и властвуй», обеспечивающие возможность его обобщения на случай более высокой размерности, приведены ниже:

1. Шаг, на котором происходит объединение решений подзадач, производится в пространстве на единицу меньшей размерности (от плоскости переходим к прямой).
2. Два множества точек, объединяемые после решения подзадач, обладают свойством *разреженности*, т. е. для каждого из множеств справедливо условие: расстояние между любыми двумя точками не может быть меньше некоторой заданной константы. Свойство разреженности имеет ключевое значение для шага объединения решений. Введем формальное определение.

**Определение 5.1.** Пусть заданы действительное число $\delta > 0$ и целое число $c \geqslant 1$. *Для заданного* $\delta$ множество точек $S$ в $d$-мер-

[1]) При анализе нерекурсивной реализации этого алгоритма, разработанной Хоуи и Шеймосом, была обнаружена любопытная ситуация: число выполняемых вычислений расстояния всегда было строго меньше $N$. Далее в этом разделе будет показано, что это имеет место всегда. Конечно, время работы алгоритма по-прежнему определяется шагом сортировки.

ном пространстве имеет *разреженность* $c$, если любой гиперкуб со стороной $2\delta$ содержит не более $c$ точек множества $S$.

Здесь разреженность определяется как (монотонная неубывающая) функция от $\delta$. Как будет показано, разреженность можно ввести, взяв в качестве $\delta$ минимальное расстояние между парами точек. Как видно из рис. 5.12, в рассматриваемом случае множество точек, находящихся на расстоянии не более $\delta$ от разделяющей прямой $l$, является разреженным и при этом $c = 12$. (Заметим, что $c = 12 = 4 \times 3$ для $d = 2$.)

В случае $d \geqslant 2$ часть пространства $E^d$, заключенная между двумя гиперплоскостями, ортогональными одной из координатных осей и удаленных друг от друга на расстояние $2\delta$, называется $\delta$-*слоем*. Заметим, что, если разреженность $\delta$-слоя при заданном $\delta$ равна $c$, эта разреженность сохраняется при проецировании $\delta$-слоя на гиперплоскость, делящую $\delta$-слой пополам ($(d-1)$-мерное многообразие). Действительно $d$-мерный куб со стороной $2\delta$ проецируется в $(d-1)$-мерный куб также со стороной $2\delta$, содержащий ровно столько же точек, сколько и исходный куб (рис. 5.13).

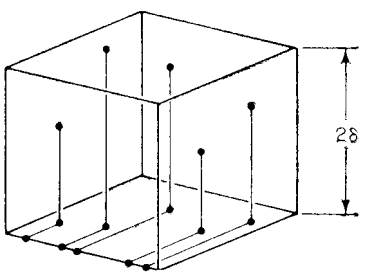



Рис. 5.13. Свойство разреженности множества точек сохраняется при ортогональном проецировании (при $d \geqslant 2$).

Предположим теперь, что в $E^d$ задано множество $S$, содержащее $N$ точек. По рассмотренной схеме, приведем следующий алгоритм:

**procedure** БПАРА$d(S)$

1. Выбрав подходящим образом гиперплоскость $l$ (называемую *разделяющей гиперплоскостью*), ортогональную одной из координатных осей, разбить множество $S$ на два подмножества $S_1$ и $S_2$ (затратив на это время $O(N)$), содержащих соответственно $\alpha N$ и $(1-\alpha)N$ точек (где $0 < \alpha_0 \leqslant \alpha \leqslant 1 - \alpha_0$ будет определено далее).

2. Рекурсивно применить алгоритм к множествам $S_1$ и $S_2$, получив минимальные расстояния $\delta_1$ и $\delta_2$ между точками в множествах $S_1$ и $S_2$ соответственно. Положить $\delta = \min(\delta_1, \delta_2)$.

3. Объединить полученные результаты, выполнив обработку точек, попавших в $\delta$-слой пространства $E^d$, имеющий ширину $2\delta$ и разделяемый гиперплоскостью $l$ пополам.

Рассмотрим более подробно шаг 3 алгоритма. В качестве рабочей гипотезы предположим, что множество точек в $\delta$-слое, обозначаемое $S_{12}$, имеет разреженность $c$ при заданном $\delta$ (позднее мы убедимся, что такое $c$ существует). Обозначим через $S_{12}^*$ проекцию множества $S_{12}$ на гиперплоскость $l$ ($(d-1)$-мерное пространство). Очевидно, что необходимым условием для того, чтобы пара точек из $S_{12}$ была ближайшей парой в множестве $S$, является следующее: расстояние между проекциями этих точек на гиперплоскость $l$ не должно превышать $\delta$. Таким образом, имеется разреженное множество $S_{12}^*$, в котором необходимо найти все пары точек, удаленных друг от друга на расстояние, не превышающее некоторую величину ($<\delta$). Это составляет предмет следующей задачи, представляющей самостоятельный интерес.



**Задача Б.7** (ПОИСК БЛИЖАЙШИХ СОСЕДЕЙ В ОКРЕСТНОСТИ ФИКСИРОВАННОГО РАДИУСА В РАЗРЕЖЕННОМ МНОЖЕСТВЕ). Пусть заданы действительное число $\delta$ и множество из $M$ точек в $E^d$, имеющее разреженность $c$ для заданного $\delta$. Требуется перечислить все пары точек, расстояние между которыми меньше $\delta$.

Полагая $|S_{12}^*| = M$ и обозначив через $F_{\delta, c}(M, d)$ время решения задачи Б.7, на основании проведенного анализа получаем следующее рекуррентное соотношение:

$$P(N, d) = P(\alpha N, d) + P((1-\alpha)N, d) + O(N) + F_{\delta, c}(M, d-1). \tag{5.2}$$

Наша цель заключается в минимизации $F_{\delta, c}(M, d-1)$. Это достигается как за счет выбора разделяющей гиперплоскости, минимизирующей $M$, так и за счет разработки эффективного алгоритма решения задачи Б.7. Обе эти подцели достижимы благодаря следующей интересной теореме:

**Теорема 5.4** [Bentley, Shamos (1976)]. *Пусть в $E^d$ задано множество $S$, содержащее $N$ точек. Существует гиперплоскость $l$, перпендикулярная одной из координатных осей и обладающая следующими свойствами*:

(1) *каждое из подмножеств $S_1$ и $S_2$ множества $S$, расположенных по обе стороны от $l$, содержит не менее $N/(4d)$ точек*;

(2) *$\delta$-слой пространства $E^d$, расположенный около гиперплоскости $l$, содержит не более $dbN^{1-1/d}$ точек, где $\delta = \min(\delta_1, \delta_2)$, $\delta_i$ — минимальное расстояние между точками множества $S_i$ ($i = 1, 2$), а $b$ — верхняя граница числа точек, содержащихся в гиперкубе со стороной $2\delta$ и удаленных друг от друга на расстояние не менее $\delta$.*

*Доказательство.* Проведем доказательство для $d = 2$, так как в этом случае можно непосредственно опереться на интуи-

цию. Доказательство для $d$-мерного случая проводится аналогичным образом. С несущественной потерей общности можно считать, что $N$ кратно 8. Определим по оси $x$ интервал $[x_1, x_2]$ таким образом, чтобы слева от $x_1$ и справа от $x_2$ находилось по $N/8$ точек множества $S$. Обозначим через $C_x \subseteq S$ множество точек с абсциссами между $x_1$ и $x_2$. Заметим, что $|C_x| = 3N/4$. Аналогичным образом определим интервал $[y_1, y_2]$ по оси $y$. Обозначим через $R$ декартово произведение $[x_1, x_2] \times [y_1, y_2]$ (рис. 5.14).

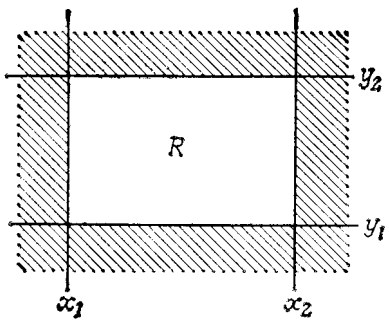



Рис. 5.14. Определение прямоугольника $R = [x_1, x_2] \times [y_1, y_2]$.

Определим на оси $x$ наибольший интервал, целиком попадающий в $[x_1, x_2]$ и содержащий проекции $2bN^{1/2}$ точек. Аналогичный интервал определим на оси $y$. Обозначим через $\gamma$ длину наибольшего из этих двух интервалов. Не теряя общности, будем считать, что таким интервалом является $[x_1', x_2'] \times \times ([x_1', x_2'] \subset [x_1, x_2])$. Утверждается, что прямая $l$, задаваемая

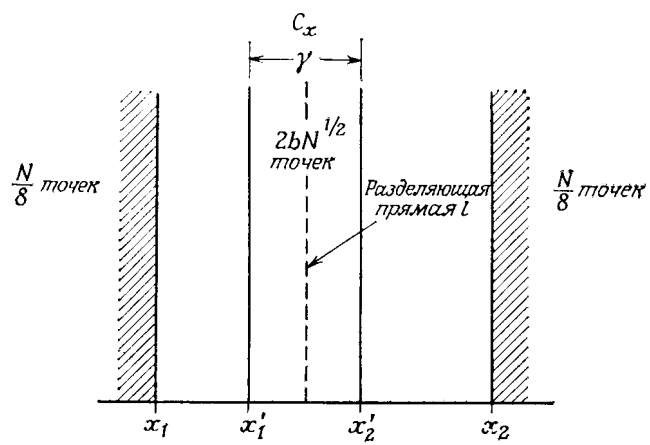



Рис. 5.15. Взаимосвязь между $[x_1, x_2]$ и $[x_1', x_2']$.

уравнением $x = (x_1' + x_2')/2$ (прямая, перпендикулярная оси $x$ и делящая отрезок $\overline{x_1'x_2'}$ пополам), удовлетворяет условиям (1) и (2) теоремы (рис. 5.15).

Для доказательства этого утверждения покажем сначала, что неравенство $\gamma < 2\delta$ не может иметь места. А затем покажем, что условие $\gamma \geqslant 2\delta$ обеспечивает необходимый результат.



Условие $\gamma < 2\delta$ означает, что любая горизонтальная или вертикальная полоса шириной $2\delta$, проекция которой попадает внутрь $[y_1, y_2]$ или $[x_1, x_2]$ соответственно, содержит больше чем $2bN^{1/2}$ точек.

Разделим интервал $[x_1, x_2]$ на подынтервалы длиной $2\delta$. Каждая вертикальная полоса, проецируемая на такой подынтервал, содержит более $2bN^{1/2}$ точек. Так как имеется $\lfloor (x_2 - x_1)/2\delta \rfloor$ таких интервалов, то получаем

$$\lfloor (x_2 - x_1)/2\delta \rfloor \cdot 2bN^{1/2} < |C_x| = 3N/4,$$

или

$$\lfloor (x_2 - x_1)/2\delta \rfloor < 3N^{1/2}/8b.$$

Аналогичные рассуждения для интервала $[y_1, y_2]$ приводят к неравенству

$$\lfloor (y_2 - y_1)/2\delta \rfloor < 3N^{1/2}/8b.$$

Рассмотрим теперь прямоугольник $R = [x_1, x_2] \times [y_1, y_2]$. Этот прямоугольник содержит не более $\lceil (x_2 - x_1)/2\delta \rceil \cdot \lceil (y_2 - y_1)/2\delta \rceil$ квадратов со стороной $2\delta$. Так как сторона каждого такого квадрата равна $2\delta$, то по условию теоремы каждый квадрат содержит не более $b$ точек. Отсюда следует, что число точек множества $S$, содержащихся в $R$, не превосходит

$$\left\lfloor \frac{x_2 - x_1}{2\delta} \right\rfloor \cdot \left\lfloor \frac{y_2 - y_1}{2\delta} \right\rfloor \cdot b \leqslant \left( \left\lfloor \frac{x_2 - x_1}{2\delta} \right\rfloor + 1 \right) \left( \left\lfloor \frac{y_2 - y_1}{2\delta} \right\rfloor + 1 \right) \cdot b < \left( \frac{3N^{1/2}}{8b} + 1 \right)^2 \cdot b.$$

Так как $b$ — это небольшая по величине константа, бóльшая или равная 1, то правая часть этого неравенства принимает максимальное значение при $b = 1$. Таким образом, $R$ содержит не более $2(3/8)^2 N \simeq 0{,}28N$ точек при условии, что $N \geqslant 8$.

С другой стороны, вертикальные полосы, находящиеся вне интервала $[x_1, x_2]$, содержат $2 \cdot N/8$ точек, и то же самое справедливо для горизонтальных полос, находящихся вне интервала $[y_1, y_2]$ (см. рис. 5.15). В соответствии с принципом включения-исключения, число точек вне $R$ не превышает $4N/8 = N/2$, и, следовательно, $R$ содержит *не менее* $N/2$ точек. Полученное противоречие указывает на то, что неравенство $\gamma < 2\delta$ не выполняется.

Так как $\gamma \geqslant 2\delta$, то сразу же получаем требуемый результат: выполнение свойства (1) обеспечивается способом построения интервала $[x_1, x_2]$, а свойство (2) выполняется в силу

того, что δ-слой около прямой $l$ содержит не более $2bN^{1/2}$ точек. Таким образом, прямая $l$ удовлетворяет условиям теоремы.

Мы используем эту теорему для того, чтобы показать прежде всего, что на шаге 1 алгоритма БПАРА$d(S)$ можно так выбрать разделяющую гиперплоскость $l$, что будет иметь место неравенство $1/4d \leqslant \alpha \leqslant 1-1/4d$, а $S_{12}$ — множество точек в δ-слое около $l$ — будет иметь мощность не более $cdN^{1-1/d}$, где $c = 4\cdot 3^{d-1}$. Действительно, если $\delta = \min(\delta_1, \delta_2)$, то гиперкуб со стороной 2δ, находящийся в δ-слое, содержит не более $c = 4\cdot 3^{d-1}$ точек, в чем можно легко убедиться с помощью индукции по $d$. Индукция начинается со значения $d = 2$, для которого, как уже было показано, гиперкуб со стороной 2δ содержит не более 12 точек. Тем самым доказывается, что разделяющая гиперплоскость $l$ существует. Для ее определения выполним предварительно сортировку множества $S$ по всем координатам, затратив на это время $O(dN\log N)$. Путем простого просмотра за время $O(dN)$ получившихся в результате сортировки упорядоченных последовательностей определяем интервалы такие, что по обе стороны от них содержится по $N/4d$ точек. В каждом из этих интервалов, опять-таки используя простой просмотр, можно за время $O(N)$ определить максимальные длины окон (подынтервалов), содержащих в точности $\lfloor cdN^{1-1/d}\rfloor$ точек. Наибольший из этих подынтервалов определяет единственную координатную ось и разделяющую гиперплоскость, которая делит его пополам. Заметим, что исключительно благодаря предварительной сортировке разделяющую гиперплоскость можно найти за линейное по размеру множества $S$ время.

Таким образом, мы получили *разреженное множество* $S_{12}$, *мощность которого не превосходит* $cdN^{1-1/d}$ (с разреженностью $c = 4\cdot 3^{d-1}$ для вычисленного δ). Затем решаем задачу Б.7 для множества $S^*_{12}$, являющегося проекцией множества $S_{12}$ на полученную ранее разделяющую гиперплоскость $l$. Для этой задачи, имеющей размерность $d-1$, вновь применяется метод «разделяй и властвуй». В частности, ищется разделяющая гиперплоскость $l'$, существование которой гарантируется теоремой 5.4, при этом δ считается заданным (каким оно получено на шаге 2 процедуры БПАРА$d$), а $c$ равна разреженности множества $S^*_{12}$, только что полученной нами. Отсюда следует, что мощность множества точек в δ-слое около разделяющей гиперплоскости $l'$ ограничена величиной $(d-1)\cdot cM^{1-1/(d-1)}$, т. е. $o(M)$. В заключение решается задача Б.7, имеющая размерность $d-1$, для чего приходится определять разделяющую гиперплоскость, затратив на это время $O(M)$, после чего рекурсивно решать две аналогичные задачи на множествах с мощностью $\alpha M$ и $(1-\alpha)M$, и наконец решается задача, имеющая размерность $d-2$ на множестве мощностью $o(M)$. Это дает следующее рекуррентное соотношение:

$$F_{\delta,c}(M, d-1) = F_{\delta,c}\left(\frac{M}{4d}, d-1\right) + F_{\delta,c}\left(M\left(1-\frac{1}{4d}\right), d-1\right) + O(M) + F_{\delta,c}(o(M), d-2). \quad (5.3)$$



Легко убедиться, что это соотношение имеет решение $F_{\delta,c}(M, d-1) = O(M\log M)$. Этот факт интересен сам по себе, чтобы представить его в виде следующей теоремы:

**Теорема 5.5.** *В разреженном множестве из $M$ точек в $E^d$ (с разреженностью $c$ для заданного δ) все пары точек, расстояния между которыми меньше δ, можно найти за время $O(M\log M)$.*

Возвращаясь к исходной задаче о ближайшей паре, напомним, что $M$ — это мощность множества $S_{12}$, равная в силу способа построения $O(N^{1-1/d})$, т. е. $o(N|\log N)$. Отсюда следует, что $F_{\delta,c}(M, d-1) = O(M\log M) = O(N)$. И как следствие, рекуррентное соотношение (5.2) имеет решение $P(N, d) = O(N\log N)$. Таким образом, время выполнения основных вычислений совпадает по порядку с временем, затрачиваемым на предварительную сортировку. В результате получаем следующую теорему:

**Теорема 5.6.** *Ближайшая пара множеств из $N$ точек в $E^d$ может быть определена за время $\theta(N\log N)$, что является оптимальным.*

## 5.5. Решение задач о близости методом локусов; диаграмма Вороного

Хотя использование метода «разделяй и властвуй» при решении задачи о ближайшей паре выглядит довольно обнадеживающе, но уже при решении задачи ВСЕ БЛИЖАЙШИЕ СОСЕДИ, являющейся на первый взгляд лишь простым обобщением задачи о ближайшей паре, это метод оказывается неприемлемым. Действительно, если попытаться аналогичным образом использовать для решения задачи ВСЕ БЛИЖАЙШИЕ СОСЕДИ рекурсию, то обнаружится, что при естественном способе разбиения на подзадачи свойство разреженности уже не имеет места и при этом не видно способа выполнить шаг слияния быстрее, чем за квадратичное время. С другой стороны, довольно ценный эвристический прием, используемый при разработке геометрических алгоритмов, состоит в том, чтобы выделить области точек, обладающие требуемым свойством

(локусы), и попытаться организовать их в виде некоторой структуры данных. Так, в случае плоскости мы хотим решить следующую задачу:

**Задача Б.8** (ОБЛАСТИ БЛИЗОСТИ). На плоскости задано множество $S$, содержащее $N$ точек. Требуется для каждой точки $p_i$ множества $S$ определить локус точек $(x, y)$ на плоскости, для которых расстояние до $p_i$ меньше, чем до любой другой точки множества $S$.

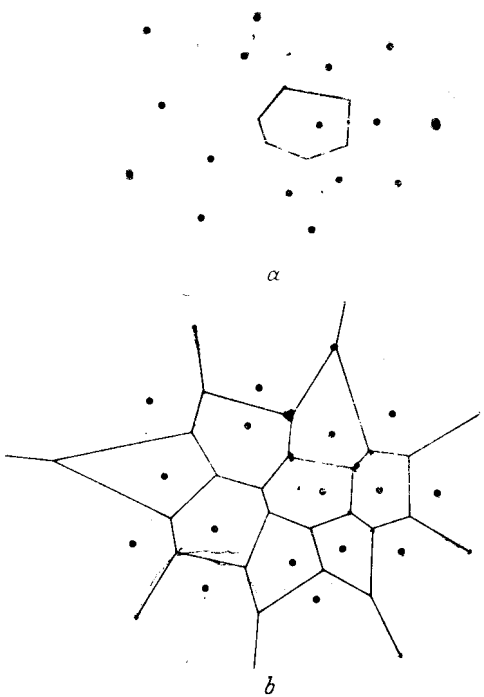


Рис. 5.16. (a) — многоугольник (ячейка) Вороного; (b) — диаграмма Вороного.

Отметим, что интуитивно решение этой задачи представляется как построение разбиения плоскости на области, каждая из которых является локусом точек $(x, y)$, более близких к некоторой точке множества $S$, чем к любой другой точке $S$. Заметим также, что если такое разбиение плоскости известно, то, применив процедуру поиска, определяющую какой из областей разбиения принадлежит некоторая точка $q$, можно непосредственно получить решение задачи ПОИСК БЛИЖАЙШЕГО СОСЕДА (задачи Б.5). Исследуем теперь структуру этого разбиения плоскости. Если имеются две точки $p_i$ и $p_j$, то множество точек, более близких к $p_i$, чем к $p_j$, есть не что иное, как полуплоскость, определяемая прямой, перпендикулярной отрезку $\overline{p_i p_j}$ и делящей его пополам, и содержащая точку $p_i$. Обозначим эту полуплоскость $H(p_i, p_j)$. Множество точек, более близких к $p_i$, чем к любой другой точке, которое будем обозначать $V(i)$, получается в результате *пересечения* $N-1$ *полуплоскостей*. Это множество является выпуклой многоугольной областью (см. разд. 1.3.1), имеющей не более $N-1$ сторон. Таким образом,

$$V(i) = \bigcap_{i \neq j} H(p_i, p_j).$$



Область $V(i)$ называется *многоугольником Вороного, соответствующим точке* $p_i$. На рис. 5.16(a) приведен пример многоугольника Вороного [Rogers (1964)][1]).

Получаемые таким образом $N$ областей образуют разбиение плоскости, представляющее некоторую сеть, называемую *диаграммой Вороного*. Диаграмму Вороного множества точек $S$ будем обозначать $\mathrm{Vor}(S)$. На рис. 5.16(b) приведен пример диаграммы Вороного. Вершины многоугольников определяют *вершины диаграммы Вороного*, а соединяющие их отрезки — *ребра диаграммы Вороного*.

Каждая из $N$ исходных точек множества принадлежит в точности одному многоугольнику Вороного. Поэтому, если $(x, y) \in V(i)$, то $p_i$ является ближайшим соседом точки $(x, y)$. Диаграмма Вороного содержит всю информацию о близости точек соответствующего множества.

### 5.5.1. Свойства диаграммы Вороного

Здесь будет приведен ряд важных свойств диаграммы Вороного. Хотя диаграмму Вороного можно определить для пространства любой размерности, последующее обсуждение будет вестись для двумерного случая. На то имеются две причины: во-первых, стремление обеспечить связь проводимых рассуждений с интуицией; во-вторых, желание сосредоточить обсуждение лишь на изученных ситуациях, для которых разработаны эффективные алгоритмы решения возникающих задач.

В этом разделе мы будем считать справедливым следующее предположение:

**Предположение А.** *Никакие четыре точки исходного множества S не лежат на одной окружности.*

Если отказаться от этого предположения, то в формулировки теорем и их доказательства потребуется внести несущественные, но довольно длинные уточнения.

Для начала заметим, что каждое ребро диаграммы Вороного является отрезком прямой, перпендикулярной отрезку, соединяющему некоторую пару точек множества $S$, и делящей этот отрезок пополам. Таким образом, ребро принадлежит в точности двум многоугольникам. Имеет место следующая теорема:

[1]) Такие многоугольники впервые были глубоко изучены русским математиком Г. Вороным (1868—1908), использовавшим их в работе по квадратичным формам [Voronoi (1908)]. Иногда их также называют многоугольниками (ячейками) Дирихле, многоугольниками Тиссена, ячейками Вигнера — Зейтца. Хоуи предложил более образный (и более подходящий) термин «многоугольники близости».

**Теорема 5.7.** *Каждая вершина диаграммы Вороного является точкой пересечения в точности трех ребер диаграммы.*

*Доказательство.* Предположим, что вершина $v$ является точкой пересечения некоторого множества ребер $e_1, e_2, \ldots, e_k$ ($k \geqslant 2$), перечисленных в порядке обхода вокруг вершины по часовой стрелке (рис. 5.17). Ребро $e_i$ является общим для многоугольников $V(i-1)$ и $V(i)$, где $i = 2, \ldots, k$, а ребро $e_1$ является общим для $V(k)$ и $V(1)$. Заметим, что так как $v$ принадлежит ребру $e_i$, то она одинаково удалена от точек $p_{i-1}$ и $p_i$. С другой стороны, аналогичным образом получаем, что

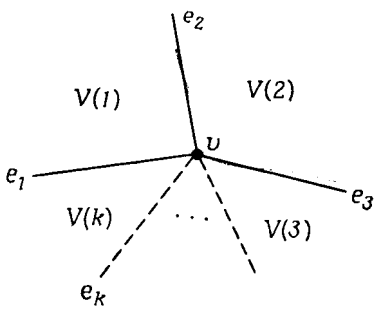



Рис. 5.17. Вершина диаграммы Вороного с инцидентными ей ребрами.

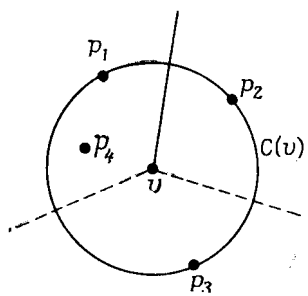



Рис. 5.18. Окружность $C(v)$ не содержит ни одной точки множества $S$.

она равноудалена от точек $p_i$ и $p_{i+1}$ и т. д. Таким образом, вершина $v$ равноудалена от точек $p_1, p_2, \ldots, p_k$. Но это значит, что точки $p_1, p_2, \ldots, p_k$ лежат на одной окружности, и тем самым в случае $k \geqslant 4$ нарушается принятое предположение А. Следовательно, $k \leqslant 3$. Предположим теперь, что $k = 2$. В этом случае оба ребра $e_1$ и $e_2$ принадлежат многоугольникам $V(2)$ и $V(1)$, поскольку оба они принадлежат перпендикуляру, делящему пополам отрезок $\overline{p_1p_2}$. А так как они не пересекаются в $v$, то вновь получаем противоречие.

Теорема 5.7 эквивалентна следующему утверждению: вершины диаграммы Вороного являются центрами окружностей, каждая из которых определяется тремя точками исходного множества, а сама диаграмма Вороного является регулярной [1]) со степенью вершин, равной трем. Обозначим через $C(v)$ упомянутую выше окружность, соответствующую вершине $v$. Эти окружности обладают следующим интересным свойством:

[1]) В том смысле, как это обычно понимается в теории графов; граф является регулярным, если все вершины имеют одну и ту же степень.



**Теорема 5.8.** *Для каждой вершины $v$ диаграммы Вороного множества $S$ окружность $C(v)$ не содержит никаких других точек множества $S$.*

*Доказательство.* Докажем теорему методом «от противного». Пусть $p_1$, $p_2$ и $p_3$ — три точки множества $S$, определяющие окружность $C(v)$ (рис. 5.18). Если окружность $C(v)$ содержит еще некоторую точку множества $S$, например $p_4$, то вершина $v$ находится ближе к $p_4$, чем к любой из точек $p_1$, $p_2$ или $p_3$, и в этом случае в соответствии с определением диаграммы Вороного вершина $v$ должна находиться в $V(4)$, а не в каком-либо из многоугольников $V(1)$, $V(2)$ или $V(3)$. Но это противоречит тому, что вершина $v$ принадлежит одновременно $V(1)$, $V(2)$ и $V(3)$.

**Теорема 5.9.** *Каждый ближайший сосед точки $p_i$ в $S$ определяет ребро в многоугольнике Вороного $V(i)$.*

*Доказательство.* Пусть $p_j$ является ближайшим соседом $p_i$, а $v$ — середина соединяющего их отрезка. Предположим, что $v$ не лежит на границе $V(i)$. Тогда отрезок $\overline{p_iv}$ пересекает некоторое ребро многоугольника $V(i)$, например равноудаленное от

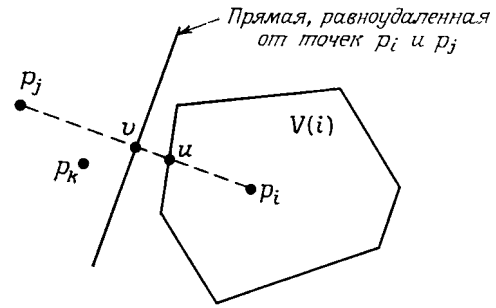



Рис. 5.19. Каждый ближайший сосед точки $p_i$ определяет ребро многоугольника $V(i)$.

$p_i$ и $p_k$, в некоторой точке $u$ (рис. 5.19). Тогда длина $(\overline{p_iu}) <$ $<$ длина $(\overline{p_iv})$ и поэтому длина $(\overline{p_ip_k}) \leqslant 2$ длина $(\overline{p_iu}) < 2$ длина $(\overline{p_iv}) =$ длина $(\overline{p_ip_j})$, откуда следует, что $p_k$ ближе к $p_i$, чем $p_j$, что противоречит условию теоремы.

**Теорема 5.10.** *Многоугольник $V(i)$ является неограниченным тогда и только тогда, когда точка $p_i$ лежит на границе выпуклой оболочки множества $S$.*

*Доказательство.* Если точка $p_i$ не лежит на границе выпуклой оболочки множества $S$, то, согласно теореме 3.4, она является внутренней точкой некоторого треугольника $p_1p_2p_3$. Рас-

смотрим окружности $C_{12}$, $C_{13}$ и $C_{23}$, определяемые точкой $p_i$ и тремя парами точек $\{p_1, p_2\}$, $\{p_1, p_3\}$ и $\{p_2, p_3\}$ соответственно (рис. 5.20). Каждая из этих окружностей имеет конечный радиус. Обозначим через $A_{12}$ внешнюю дугу окружности $C_{12}$, т. е. дугу окружности между точками $p_1$ и $p_2$, не содержащую точку $p_i$. Можно непосредственно показать, что любая точка $A_{12}$ ближе к $p_1$ или к $p_2$, чем к $p_i$ (аналогичное утверждение имеет место и для $C_{13}$ и $C_{23}$). Пусть $C$ — окружность, охватывающая окружности $C_{12}$, $C_{13}$ и $C_{23}$. Утверждается, что любая точка $x$, лежащая вне $C$, ближе к одной из точек $p_1$, $p_2$ или $p_3$, чем к $p_i$. Действительно, рассмотрим отрезок $\overline{xp_i}$. По теореме о жордановой кривой отрезок $\overline{xp_i}$ пересекает одну из сторон треугольника $p_1p_2p_3$, например $\overline{p_1p_2}$. Значит, он также пересекает и $A_{12}$ в некоторой точке $u$. Но $u$ ближе к одной из точек $p_1$ или $p_2$, чем к $p_i$, что и подтверждает сделанное утверждение. Так как точка $x$ ближе к $p_1$, $p_2$ или $p_3$, чем к $p_i$, то $V(i)$ целиком содержится внутри $C$ и, следовательно, является ограниченным.

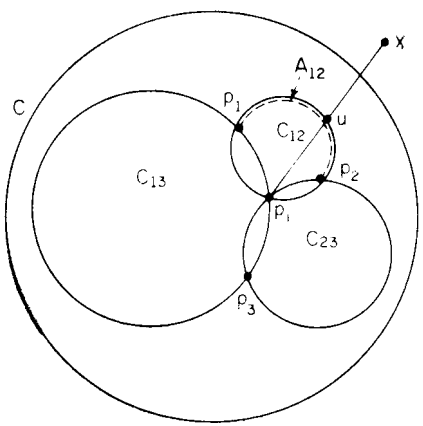



Рис. 5.20. К доказательству теоремы 5.10.

Обратно, предположим, что многоугольник $V(i)$ ограничен, и пусть $e_1, e_2, \ldots, e_k$ $(k \geqslant 3)$ — последовательность ребер, образующих его границу. Каждое ребро $e_h$ $(h = 1, \ldots, k)$ принадлежит прямой, перпендикулярной отрезку $\overline{p_ip'_h}$, $p'_h \in S$ и делящей его пополам. Отсюда сразу следует, что $p_i$ является внутренней точкой многоугольника $p'_1p'_2 \ldots p'_k$, т. е. $p_i$ не лежит на границе выпуклой оболочки множества $S$.

Так как лишь неограниченные многоугольники могут иметь в качестве ребер лучи, то лучи диаграммы Вороного соответствуют парам смежных точек множества $S$, лежащих на границе выпуклой оболочки.

Рассмотрим теперь граф, двойственный диаграмме Вороного, т. е. граф, уложенный на плоскости и получаемый в результате соединения отрезками каждой пары точек множества $S$, многоугольники Вороного которых имеют общее ребро. В результате получается граф с вершинами в исходных $N$ точках (рис. 5.21).



На первый взгляд граф, двойственный диаграмме Вороного, может показаться довольно необычным, так как ребро диаграммы может даже не пересекаться с двойственным ему ребром (рассмотрите, например, ребра, соединяющие последовательные вершины выпуклой оболочки). Важность двойственного графа во многом обусловлена следующей теоремой Делоне [Delaunay (1934)]:

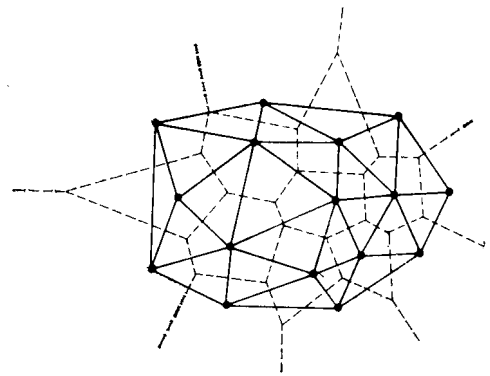


Рис. 5.21. Прямолинейный граф, двойственный диаграмме Вороного.

**Теорема 5.11.** *Граф, двойственный диаграмме Вороного, является триангуляцией множества* $S$ [1]).

(Отсюда следует, что диаграмма Вороного может быть использована для решения задачи триангуляции Б.4, но эта теорема имеет значительно более важные следствия.)

*Доказательство.* Чтобы доказать, что граф, двойственный диаграмме Вороного, является триангуляцией, необходимо показать, что выпуклая оболочка множества $S$ разбивается на треугольники, определяемые точками множества $S$. Для этого будет показано, что можно выделить множество треугольников $\mathcal{T} = \{T(v)\colon v$ — вершина диаграммы Вороного$\}$ так, что никакие два из них не пересекаются и каждая точка внутри $\mathrm{conv}(S)$ принадлежит одному из этих треугольников (и, следовательно, в точности одному такому треугольнику).

Соответствующие треугольники строятся следующим образом. Пусть $v$ — вершина диаграммы Вороного, принадлежащая одновременно многоугольникам $V(1)$, $V(2)$ и $V(3)$ (см. теорему 5.7). $T(v)$ — это треугольник с вершинами в точках $p_1$, $p_2$

[1]) В такой простой формулировке эта теорема оказывается неверной, если имеется подмножество, содержащее четыре или более точек, лежащих на одной окружности. Однако в этом случае возникающие изменения в триангуляции очевидны.

и $p_3$. Утверждается, что если $T(v)$ имеет непустое пересечение с внутренностью $\mathrm{conv}(S)$, то треугольник $T(v)$ невырожденный (т. е. точки $p_1$, $p_2$ и $p_3$ не лежат на одной прямой). Действительно, предположим обратное, т. е. что точки $p_1$, $p_2$ и $p_3$ лежат на одной прямой. Тогда все три отрезка $p_1p_2$, $p_1p_3$ и $p_2p_3$ принадлежат одной и той же прямой $l$, а перпендикулярные этим отрезкам прямые, делящие их пополам, являются параллельными. Так как вершина $v$ является точкой пересечения этих прямых, то $v$ — бесконечно удаленная точка. Отсюда следует, что многоугольники $V(1)$, $V(2)$ и $V(3)$ являются неограниченными. Согласно теореме 5.10, это значит, что точки $p_1$, $p_2$ и $p_3$ принадлежат границе выпуклой оболочки. Это противоречит предположению, что $T(v)$ имеет непустое пересечение с *внутренностью* $\mathrm{conv}(S)$. Тем самым утверждение доказано.

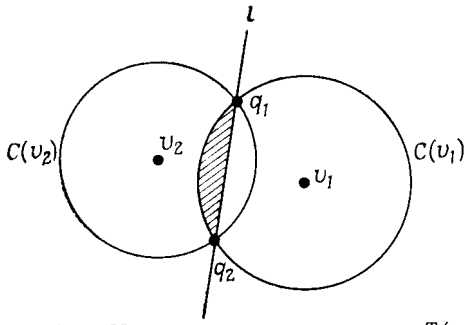



Рис. 5.22. Ни одна точка треугольника $T(v_1)$ не попадает внутрь заштрихованной области.

Рассмотрим два треугольника $T(v_1)$ и $T(v_2)$ $(v_1 \neq v_2)$ и соответствующие им окружности $C(v_1)$ и $C(v_2)$ (заметим, что окружность $C(v_i)$ является описанной вокруг треугольника $T(v_i)$). Если окружности $C(v_1)$ и $C(v_2)$ не пересекаются, то не пересекаются и треугольники $T(v_1)$ и $T(v_2)$. Поэтому предположим, что $C(v_1)$ и $C(v_2)$ имеют общие внутренние точки. Отметим, что в силу теоремы 5.8 ни одна из окружностей не может целиком содержать другую, поэтому окружности $C(v_1)$ и $C(v_2)$ пересекаются в двух точках $q_1$ и $q_2$, которые определяют прямую $l$ (рис. 5.22), отделяющую точки $v_1$ и $v_2$ друг от друга. Покажем, что $l$ также отделяет треугольник $T(v_1)$ от треугольника $T(v_2)$. Предположим обратное: в полуплоскости, определяемой $l$ и содержащей вершину $v_2$, имеются точки треугольника $T(v_1)$ (т. е. в области, заштрихованной на рис. 5.22, так как $T(v_1) \subset C(v_1)$). Отсюда следует, что в заштрихованную область, а следовательно, и в $C(v_2)$ попадает вершина треугольника $T(v_1)$, что противоречит теореме 5.8. Таким образом, $T(v_1)$ и $T(v_2)$ не имеют общих внутренних точек.

Наконец, рассмотрим произвольную точку $x$ в $\mathrm{conv}(S)$ и предположим, что она не принадлежит ни одному из треугольников $T(v)$, где $v$ — вершина диаграммы Вороного. Отсюда следует, что существует достаточно маленький круг $\gamma$ с центром в $x$, все точки которого обладают тем же свойством, что и точка $x$



(рис. 5.23). Пусть $q$ — некоторая точка в $T(v)$, где $v$ — произвольная вершина диаграммы. В круге $\gamma$ можно выбрать точку $y \in \gamma$, такую, что прямая $l$, проходящая через $q$ и $y$, не проходит ни через одну из точек множества $S$. Прямая $l$ пересекает треугольники семейства $\mathcal{T}$ по некоторому множеству замкнутых интервалов. Обозначим через $t$ конечную точку ближайшего к $x$ интервала и будем считать, что $t$ принадлежит ребру $p_1p_2$ треугольника $T(v_1)$, где $v_1$ — некоторая вершина диаграммы Воро-

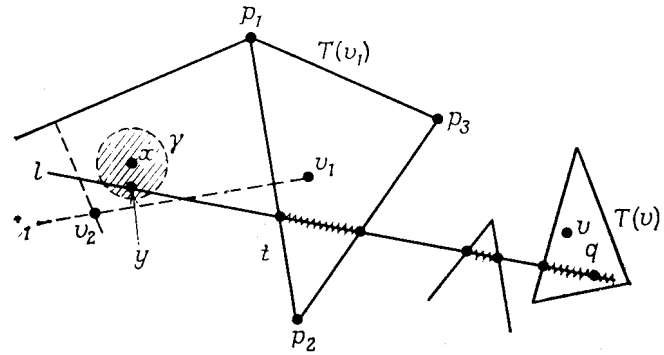



Рис. 5.23. Каждая точка внутри выпуклой оболочки множества принадлежит некоторому треугольнику.

ого. Пусть $p_3$ — третья вершина треугольника $T(v_1)$. Рассмотрим перпендикуляр $l_1$ из $v_1$ к $p_1p_2$. Начальная часть $l_1$ совпадает ребром, ограничивающим $V(1)$ и $V(2)$. Так как по предположению точка $x$ находится в $\mathrm{conv}(S)$, а $x$ и $p_3$ находятся по разные стороны отрезка $p_1p_2$, то отрезок $p_1p_2$ не принадлежит границе выпуклой оболочки множества $S$. Отсюда следует, что ребро, ежащее на $l_1$, *не является* лучом (по теореме 5.10), а представляет отрезок, заканчивающийся в некоторой вершине $v_2$. ледовательно, $p_1$ и $p_2$ являются вершинами треугольника $(v_2)$, и так как $T(v_1) \cap T(v_2) = \varnothing$ (как было показано выше), $v_1$ и $v_2$ лежат по разные стороны от отрезка $p_1p_2$. Значит, является *внутренней* точкой $T(v_1) \cup T(v_2)$, и это противоречит необоснованному) предположению, что точки отрезка $\bar{t}y$ не принадлежат ни одному треугольнику из $\mathcal{T}$.

Предыдущая теорема имеет непосредственное следствие:

**Следствие 5.2.** *Диаграмма Вороного множества из $N$ точек имеет не более $2N-5$ вершин и $3N-6$ ребер.*

*Доказательство.* Каждому ребру графа, двойственного диаграмме Вороного, соответствует единственное ребро диаграммы. двойственный граф является триангуляцией, а значит, планар-

ным графом с $N$ вершинами. В соответствии с формулой Эйлера он имеет не более $3N-6$ ребер и $2N-4$ граней. Следовательно, диаграмма Вороного имеет не более $3N-6$ ребер. Однако лишь ограниченные грани (их не более $2N-5$) соответствуют вершинам диаграммы Вороного при отображении двойственности.

Используя введенное понятие двойственности, диаграмму Вороного можно также применить для решения задачи ТРИАНГУЛЯЦИЯ (задача Б.4).

Так как диаграмма Вороного является планарным графом, то для ее представления достаточно *линейного* по числу точек объема памяти. Это позволяет представить информацию о близости в чрезвычайно компактной форме. Любой конкретный многоугольник Вороного может иметь до $N-1$ ребер, но полное число ребер не превосходит $3N-6$, при этом каждое ребро принадлежит в точности двум многоугольникам. Это значит, что *среднее* число ребер многоугольника Вороного не превосходит шести.

В двух последующих разделах рассмотренные свойства будут использованы для быстрого построения диаграммы Вороного и рассмотрено ее применение для решения задач о близости точек.

### 5.5.2. Построение диаграммы Вороного

Хотя мы будем использовать диаграммы Вороного для решения других задач, следует отметить, что в целом ряде приложений конечной целью является именно построение диаграммы Вороного. В археологии многоугольники Вороного используются для нанесения на карту ареала применения орудий труда в древних культурах и для изучения влияния соперничающих центров торговли [Hodder, Orton (1976)]. В экологии возможности организма на выживание зависят от числа соседей, с которыми он должен бороться за пищу и свет. Использование диаграммы Вороного, отражающей картину расселения животных и распределения жизненно важных ресурсов, помогает исследовать эффект перенаселенности [Pielou (1977)]. Совместное влияние электрических и близкодействующих сил, для изучения которых строятся сложные диаграммы Вороного, помогает определять структуру молекул.

Под задачей построения диаграммы Вороного $\mathrm{Vor}(S)$ на множестве точек $S$, для ссылки на которую мы будем использовать название ДИАГРАММА ВОРОНОГО, будем понимать порождение описания диаграммы как планарного графа, уложенного на плоскости. Это описание должно включать следующие элементы (см. разд. 1.2.3.2):



1. Координаты вершин диаграммы Вороного.
2. Множество ребер диаграммы, каждое из которых представляется парой вершин диаграммы. Для каждого ребра указываются два других ребра, следующих за ним при обходе против часовой стрелки в каждой его концевой точке (реберный список с двойными связями, см. разд. 1.2.3.2).

Такая организация неявным образом дает циклическую упорядоченность против часовой стрелки ребер, инцидентных любой вершине диаграммы, а также циклическую упорядоченность по часовой стрелке ребер любой грани.

Теперь, как обычно, рассмотрим сначала вопрос о нижних оценках для времени, необходимого для построения диаграммы Вороного. Ответ на этот вопрос дает следующая простая теорема:

**Теорема 5.12.** *В рамках модели алгебраических деревьев вычислений для построения диаграммы Вороного множества из $N$ точек требуется $\Omega(N \log N)$ операций в худшем случае.*

*Доказательство.* Далее мы увидим, что все задачи о близости точек можно свести за линейное время к задаче ДИАГРАММА ВОРОНОГО, и, следовательно, существует много различных способов доказательства этой теоремы. Но здесь мы ограничимся очень простым доказательством. В *одномерном* случае диаграмму Вороного множества, состоящего из $N$ точек, представляет последовательность из $N-1$ точек, отделяющих друг от друга последовательные точки исходного множества и расположенных посередине между ними. Имея эту последовательность, можно за линейное время получить упорядоченный список точек исходного множества. Таким образом, задача СОРТИРОВКА может быть сведена за линейное время к задаче ДИАГРАММА ВОРОНОГО.

Простой (и достаточно грубый) подход к построению диаграммы Вороного заключается в поочередном построении многоугольников, входящих в диаграмму. Так как каждый многоугольник Вороного получается в результате пересечения $N-1$ полуплоскостей, то, используя метод, который будет рассмотрен в гл. 7, можно построить этот многоугольник за время $O(N \log N)$, что дает для полного времени построения всех многоугольников оценку $O(N^2 \log N)$. Далее будет показано, что вся диаграмма целиком может быть построена за оптимальное время $\theta(N \log N)$, а это значит, что асимптотически задача построения всей диаграммы является не более сложной, чем задача построения лишь одного из многоугольников этой диаграммы!

В действительности, несмотря на кажущуюся сложность, задача ДИАГРАММА ВОРОНОГО прекрасно подходит для применения метода «разделяй и властвуй». Успех применяемого нами метода зависит от различных структурных свойств диаграммы, использование которых позволяет произвести объединение решений подзадач за линейное время.

Попытаемся в общих чертах описать алгоритм.

**procedure** ДИАГРАММА-ВОРОНОГО (предварительный вариант)

*Шаг* 1. Разделить множество $S$ на два приблизительно равных подмножества $S_1$ и $S_2$.

*Шаг* 2. Рекурсивно построить $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$.

*Шаг* 3. Объединить $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$ и таким образом получить $\mathrm{Vor}(S)$.

Предположим, что шаг 1 алгоритма можно выполнить за время $O(N)$ (позже мы докажем это). Если обозначить через $T(N)$ полное время работы алгоритма, то на выполнение шага 2 потребуется время, равное примерно $2T(N/2)$. Таким образом, если удастся объединить $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$ за линейное время, получив в результате диаграмму Вороного $\mathrm{Vor}(S)$ всего исходного множества, то мы получим оптимальный алгоритм с временем работы $\theta(N \log N)$. Но прежде чем браться за более детальную разработку алгоритма, попытаемся ответить на следующий вопрос: на каком основании можно считать, что $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$ как-то связаны с $\mathrm{Vor}(S)$?

Чтобы ответить на этот вопрос, определим геометрическую конструкцию, играющую ключевую роль в рассматриваемом подходе.

**Определение 5.2.** Пусть для заданного разбиения $\{S_1, S_2\}$ множества $S$ $\sigma(S_1, S_2)$ обозначает множество ребер диаграммы Вороного, общих для пар многоугольников $V(i)$ и $V(j)$ диаграммы $\mathrm{Vor}(S)$, где $p_i \in S_1$ и $p_j \in S_2$.

Совокупность ребер $\sigma(S_1, S_2)$ обладает следующими свойствами.

**Теорема 5.13.** *Совокупность* $\sigma(S_1, S_2)$ *является множеством ребер некоторого подграфа диаграммы* $\mathrm{Vor}(S)$ *и обладает следующими свойствами:*

(1) $\sigma(S_1, S_2)$ *состоит из циклов и цепей, не имеющих общих ребер. Если некоторая цепь содержит лишь одно ребро, то это ребро является прямой линией; иначе два крайних ребра цепи являются лучами.*

(2) *Если множества* $S_1$ *и* $S_2$ *линейно разделимы*[1]*, то* $\sigma(S_1, S_2)$ *состоит из единственной монотонной цепи* (см. разд. 2.2.2.2).

[1]) Если разделяющей прямой принадлежат более одной точки, то всех их следует отнести к одному и тому же множеству разбиения.



*Доказательство.* (1) Если представить, что каждый из многоугольников $\{V(i): p_i \in S_1\}$ покрашен в красный цвет, а каждый из многоугольников $\{V(j): p_j \in S_2\}$ — в зеленый цвет, то $\mathrm{Vor}(S)$ превратится в двухцветную карту. Хорошо известно, что границы между многоугольниками различных цветов представляют циклы и цепи, не имеющие общих ребер [Bollobás (1979)]. (Заметим, что две компоненты множества $\sigma(S_1, S_2)$ могут иметь общую вершину лишь в случае, когда степень этой вершины не меньше четырех. Поэтому, если все вершины диаграммы Вороного имеют степень три, то компоненты множества $\sigma(S_1, S_2)$ не пересекаются также и по вершинам.) Каждая компонента множества $\sigma(S_1, S_2)$ разбивает плоскость на две части. Таким образом, цепь либо содержит единственное ребро, представляющее прямую линию, либо ее первое и последнее ребра являются лучами. Что доказывает свойство (1).

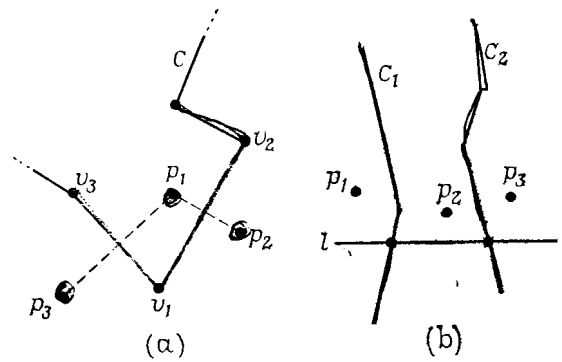



Рис. 5.24. Множества $S_1$ и $S_2$ разделимы вертикальной прямой; (a) — каждая компонента $\sigma(S_1, S_2)$ является монотонной по вертикали цепью; (b) — $\sigma(S_1, S_2)$ состоит из единственной цепи.

(2) Не теряя общности, предположим, что множества $S_1$ и $S_2$ разделены вертикальной прямой $m$, и пусть $C$ — компонента множества $\sigma(S_1, S_2)$. Будем двигаться по $C$, начав движение из некоторой точки $q$ ребра компоненты $C$ в направлении уменьшения $y$-координаты до тех пор, пока не достигнем вершины диаграммы $v_1$, в которой $C$ поворачивает вверх (рис. 5.24(a)). Ребро $\overline{v_1v_2}$ перпендикулярно отрезку $\overline{p_1p_2}$ и делит его пополам, а ребро $\overline{v_1v_3}$ перпендикулярно отрезку $\overline{p_1p_3}$ и также делит его пополам. Так как $y(v_3) > y(v_1)$ и $y(v_2) > y(v_1)$, то $x(p_3) \leqslant x(p_1) \leqslant x(p_2)$. Однако, учитывая форму $C$, точки $p_2$ и $p_3$ принадлежат одному и тому же множеству (либо $S_1$, либо $S_2$).

Возникающая ситуация противоречит предположению о разделимости $S_1$ и $S_2$ вертикальной прямой. Значит, вершина, подобная $v_1$, не может существовать, а $C$ является монотонной относительно вертикальной прямой. Отсюда следует, что компонента $C$ является цепью.

Предположим теперь, что $\sigma(S_1, S_2)$ содержит по крайней мере две монотонные относительно вертикальной прямой цепи $C_1$ и $C_2$. Произвольная горизонтальная прямая $l$ пересекает каждую из цепей $C_1$ и $C_2$ в единственной точке (в силу монотонности $C_1$ и $C_2$). Предположим, что точка пересечения прямой $l$ с $C_1$ находится левее точки пересечения $l$ с $C_2$ (рис. 5.24(b)). Тогда в $S$ существуют три точки $p_1$, $p_2$ и $p_3$ такие, что $x(p_1) < x(p_2) < x(p_3)$. При этом $p_1$ и $p_3$ принадлежат одному и тому же множеству разбиения $\{S_1, S_2\}$, что противоречит предположению о разделимости этих двух множеств вертикальной прямой. Поэтому $\sigma(S_1, S_2)$ содержит лишь одну компоненту, являющуюся монотонной цепью.

В том случае, когда множества $S_1$ и $S_2$ являются линейно разделимыми, единственную цепь, содержащуюся в $\sigma(S_1, S_2)$, будем обозначать $\sigma$. Если отделяющая прямая $m$ является вертикальной, то можно недвусмысленно говорить о том, что $\sigma$ разбивает плоскость на левую $\pi_L$ и правую $\pi_R$ части. Имеет место следующее важное свойство.

**Теорема 5.14.** *Если множества $S_1$ и $S_2$ линейно разделимы вертикальной прямой и при этом $S_1$ находится слева от $S_2$, то диаграмма Вороного* $\mathrm{Vor}(S)$ *представляет собой объединение* $\mathrm{Vor}(S_1) \cap \pi_L$ *и* $\mathrm{Vor}(S_2) \cap \pi_R$ [1]).

*Доказательство.* Все точки множества $S$, расположенные справа от $\sigma$, принадлежат $S_2$. Кроме того, все ребра диаграммы $\mathrm{Vor}(S)$, расположенные справа от $\sigma$, разделяют многоугольники $V(i)$ и $V(j)$, где обе точки $p_i$ и $p_j$ принадлежат $S_2$. Отсюда следует, что каждое ребро диаграммы $\mathrm{Vor}(S)$, попадающее в $\pi_R$, либо совпадает с некоторым ребром $\mathrm{Vor}(S_2)$, либо является его частью. Аналогичные утверждения справедливы и для $\pi_L$.

Предыдущая теорема дает ответ на вопрос о том, какая связь имеется между $\mathrm{Vor}(S_1)$, $\mathrm{Vor}(S_2)$ и $\mathrm{Vor}(S)$. А именно если множества $S_1$ и $S_2$ линейно разделимы (а это полностью в нашей власти и может быть реализовано на шаге 1 алгоритма), то теорема дает метод, позволяющий выполнить объединение $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$. Пересмотренный вариант алгоритма приведен ниже.

[1]) Диаграмма Вороного $\mathrm{Vor}(S)$ представляет собой объединение $\mathrm{Vor}(S_1) \cap \pi_L$, $\mathrm{Vor}(S_2) \cap \pi_R$ и $\sigma$. — *Прим. перев.*



**procedure** ДИАГРАММА-ВОРОНОГО

*Шаг* 1. Разделить множество $S$ на два приблизительно равных подмножества $S_1$ и $S_2$, использовав для этого медиану по $x$-координате.

*Шаг 2.* Рекурсивно построить $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$.

*Шаг 3′.* Построить ломаную $\sigma$, разделяющую $S_1$ и $S_2$.

*Шаг 3″.* Удалить все ребра диаграммы $\mathrm{Vor}(S_2)$, расположенные слева от $\sigma$, и все ребра $\mathrm{Vor}(S_1)$, расположенные справа от $\sigma$. В результате получаем $\mathrm{Vor}(S)$ — диаграмму Вороного для множества в целом.

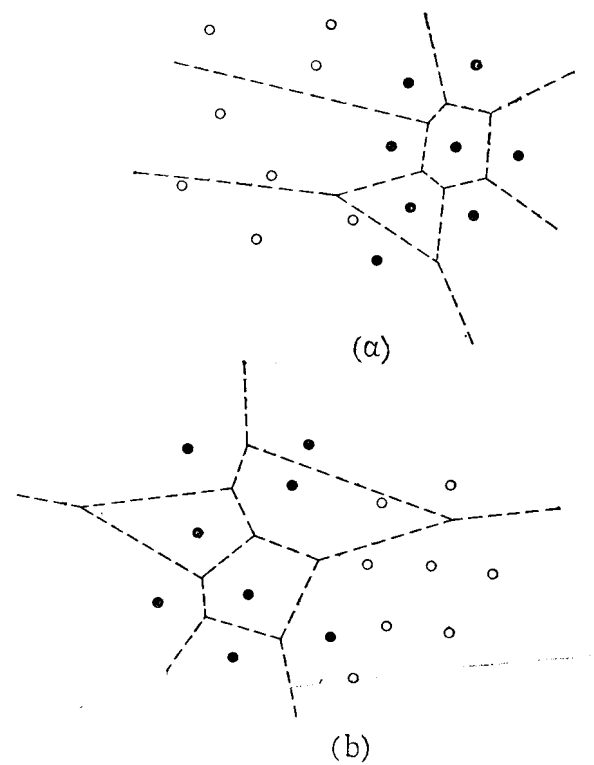



Рис. 5.25. Диаграмма Вороного левого (a) и правого (b) множеств.

Очевидно, что успешное выполнение этой процедуры зависит от того, насколько быстро мы сможем построить $\sigma$, так как шаг 3″ не вызывает затруднений. Для наглядности на рис. 5.25(a) и 5.25(b) отдельно показаны $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$ соответственно, а на рис. 5.26 они представлены вместе с $\sigma$.

Если говорить о времени обработки, то начальное разбиение множества $S$ с использованием медианы по $x$-координате

может быть выполнено за время $O(N)$ с помощью стандартного алгоритма поиска медианы. Кроме того, шаг 3″ может быть выполнен за время $O(|S_1|+|S_2|)=O(N)$. Остается лишь найти эффективный способ построения разделяющей цепи σ.

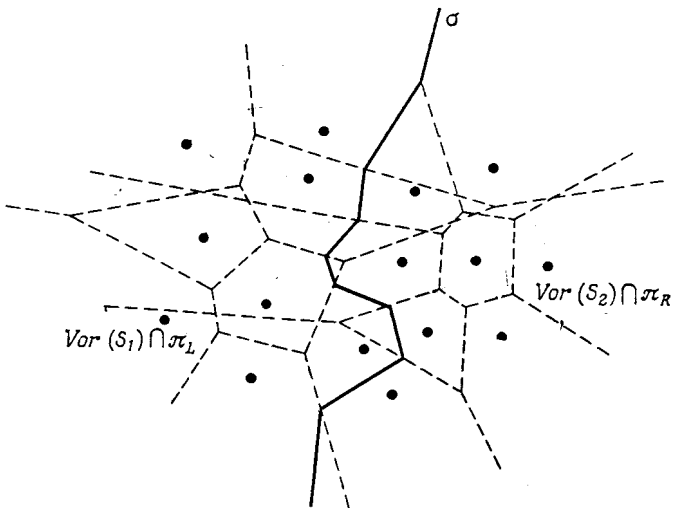



Рис. 5.26. Наложение $\mathrm{Vor}(S_1)$, $\mathrm{Vor}(S_2)$ и σ.

Рассмотрим теперь решение этой задачи [Shamos, Hoey (1975), Lee (1978), (1980a)].

### 5.5.2.1. *Построение разделяющей цепи*

Первый шаг в построении разделяющей цепи σ заключается в определении двух ее лучей. Заметим, что каждый луч цепи σ перпендикулярен опорному отрезку (каждый своему) к $\mathrm{CH}(S_1)$ и $\mathrm{CH}(S_2)$ и делит его пополам. Отметим также, что так как по предположению $S_1$ и $S_2$ линейно разделимы, то имеются в точности два опорных отрезка к $\mathrm{CH}(S_1)$ и $\mathrm{CH}(S_2)$ (тем самым подтверждается, что $\sigma(S_1, S_2)$ состоит лишь из одной цепи σ). Если теперь (по индукции) предположить, что уже построены эти две выпуклые оболочки, то два опорных отрезка, которые мы обозначим $t_1$ и $t_2$, строятся (самое большее) за линейное время (см. разд. 3.3.5) после чего легко определяются лучи цепи σ (рис. 5.27). Заметим, что в качестве побочного результата мы получаем также $\mathrm{CH}(S)$, обеспечивая тем самым наличие выпуклых оболочек, необходимых для шага индукции.



Найдя луч цепи σ, мы последовательно строим ребра цепи до тех пор, пока не достигнем второго луча. Для лучшего понимания полезно обратиться к примеру на рис. 5.28, на котором для простоты каждая точка $p_j$ представлена своим номером $j$. В приведенном примере верхний луч цепи σ перпендикулярен отрезку, соединяющему точки 7 и 14, и делит его пополам. Представим точку $z$, двигающуюся по верхнему лучу из бесконечности сверху вниз. Первоначально точка $z$ находится в некоторых многоугольниках диаграмм $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$. Точка $z$ будет двигаться по лучу до тех пор, пока не пересечет ребро одного из этих многоугольников. С этого момента точка начнет двигаться в другом направлении. В нашем примере точка $z$ первой пересечет ребро диаграммы $\mathrm{Vor}(S_2)$. Это значит, что теперь $z$ стала ближе к точке 11, чем к точке 14, и поэтому она должна двигаться вдоль прямой, перпендикулярной отрезку, соединяющему точки 7 и 11, и делящей его пополам.

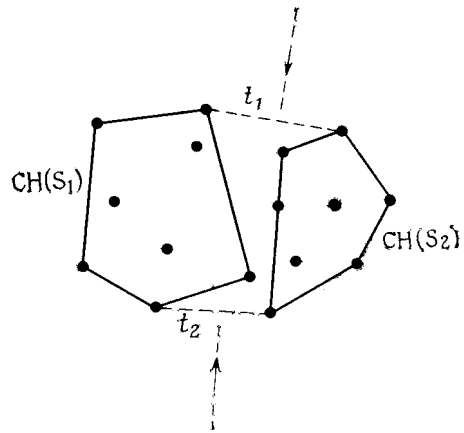



Рис. 5.27. Определение лучей цепи σ. Лучи цепи σ перпендикулярны опорным отрезкам к $\mathrm{CH}(S_1)$ и $\mathrm{CH}(S_2)$ и делят их пополам.

Это продолжается до тех пор, пока не будет достигнута прямая, перпендикулярная отрезку, соединяющему точки 6 и 7, делящая его пополам. Теперь точка $z$ двигается по прямой, перпендикулярной отрезку, соединяющему точки 6 и 11, и делящей его пополам. В конце концов она пересечет прямую, перпендикулярную отрезку 10—11, делящую его пополам, и движение продолжится по прямой, перпендикулярной отрезку 6—10 и делящей его пополам. Такое зигзагообразное движение по ломаной будет продолжаться до тех пор, пока не будет достигнут нижний луч цепи σ.

Предположим, что продвижение по σ осуществляется в направлении уменьшения $y$-координаты. Механизм продвижения по σ обеспечивает переход от текущего ребра $e$ и текущей вершины $v$ (являющейся верхним концом ребра $e$) к следующим ребру $e'$ и вершине $v'$. Если ребро $e$ разделяет многоугольники $V(i)$ и $V(j)$, где $p_i \in S_1$ и $p_j \in S_2$, то вершина $v'$ является либо точкой пересечения $e$ с границей $V(i)$ в $\mathrm{Vor}(S_1)$, либо точкой пересечения $e$ с границей $V(j)$ в $\mathrm{Vor}(S_2)$ в зави-

симости от того, какая из них ближе к $v$. Таким образом, просмотрев границы многоугольников $V(i)$ в $\mathrm{Vor}(S_1)$ и $V(j)$ в $\mathrm{Vor}(S_2)$, можно легко определить вершину $v'$. К сожалению, цепь $\sigma$ может, оставаясь внутри некоторого многоугольника $V(i)$, многократно менять свое направление, прежде чем она пересечет какое-либо ребро многоугольника $V(i)$. И если каждый раз приходится просматривать все многоугольники $V(i)$,

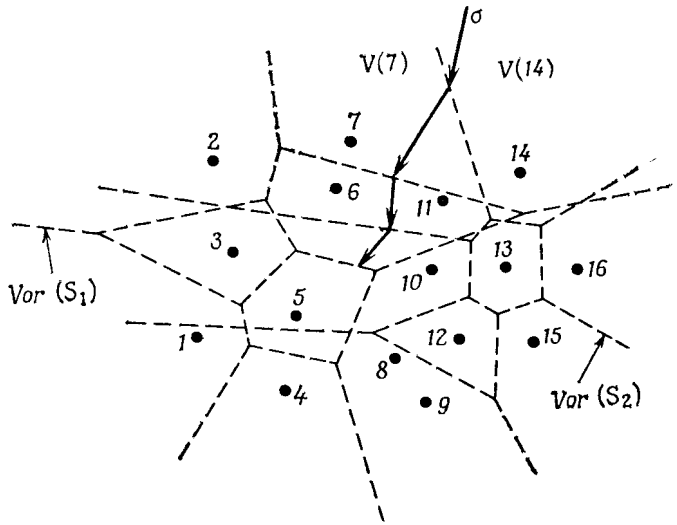



Рис. 5.28. Зигзагообразный путь, проходимый при построении σ.

то может потребоваться проверить слишком много ребер, и в общем случае необходимое для этого время может оказаться квадратичным. Однако использование особенностей структуры диаграммы Вороного позволяет разработать более эффективную стратегию просмотра.

Действительно, предположим, что последовательность ребер $e_1, e_2, \ldots, e_k$ цепи σ содержится в $V(i)$, где для конкретности $p_i \in S_1$ (см. пример на рис. 5.29, где $k = 3$). Продолжим каждое ребро $e_h$ за его конечную точку $v_h$, получая луч $r_h$, и рассмотрим точку пересечения $q_h$ луча $r_h$ с границей многоугольника $V(i)$ в $\mathrm{Vor}(S_1)$. Рассмотрим подцепь $C_1 = e_1, \ldots, e_k$ цепи σ и часть границы многоугольника $V(i)$ в $\mathrm{Vor}(S_1)$, расположенную справа от σ, которую обозначим $C_2$. Подцепь $C_2$ — это часть границы (возможно, неограниченного) выпуклого многоугольника $V(i)$ в $\mathrm{Vor}(S_1)$; $C_1$ также является выпуклой и целиком содержится в $V(i)$. Так как все углы между отрезками $\overline{v_h q_h}$ и $\overline{v_h q_{h+1}}$, где $h = 1, 2, \ldots, k-1$ (с учетом направления от $\overline{v_h q_h}$ к $\overline{v_h q_{h+1}}$), имеют один и тот же знак, то точки пересечения $q_1, q_2, \ldots, q_k$ оказываются *упорядоченными* на $C_2$. Отсюда следует, что при определении точек пересечения $q_1, q_2, \ldots$ цепь $C_2$, т. е. границу многоугольника $V(i)$ в $\mathrm{Vor}(S_1)$, необходимо просматривать *без возвратов в порядке обхода по часовой стрелке*. Аналогичные рассуждения показывают, что границу любого многоугольника $V(j)$ в $\mathrm{Vor}(S_2)$ следует просматривать *без возвратов в порядке обхода против часовой стрелки*.



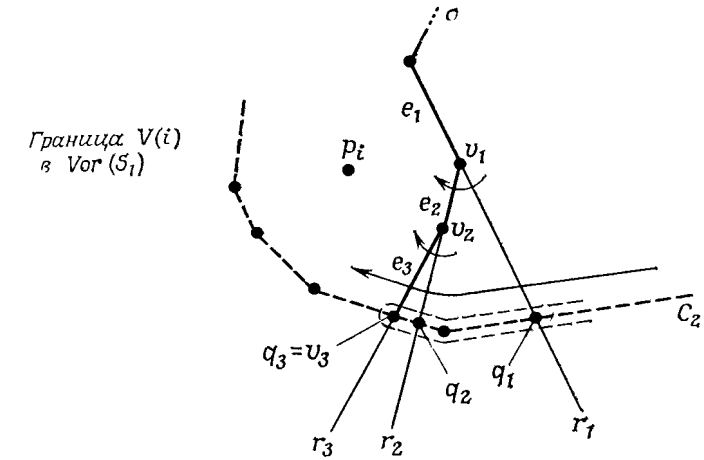



Рис. 5.29. Точки пересечения $q_1, q_2, q_3$ упорядочены на $C_2$.

В частности, будем считать, что как $\mathrm{Vor}(S_1)$, так и $\mathrm{Vor}(S_2)$ представлены в виде РСДС (см. разд. 1.2.3.2), при этом циклы граней ориентированы по часовой стрелке в $\mathrm{Vor}(S_1)$ и против часовой стрелки в $\mathrm{Vor}(S_2)$. Для представления РСДС диаграммы $\mathrm{Vor}(S_1)$ используется массив указателей СЛЕДУЮЩИЙ$_1$[ ], используемый для обхода границы грани по часовой стрелке. Аналогичный массив СЛЕДУЮЩИЙ$_2$[ ] используется для представления $\mathrm{Vor}(S_2)$. В работе алгоритма используются три ребра: два ребра $e_L$ и $e_R$, принадлежащие соответственно $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$, и «текущее ребро» $e$ цепи σ (в действительности не само ребро, а прямая, содержащая это ребро). Кроме ребра $e$ используется также его начальная точка $v$ (для первого ребра $e^*$ цепи σ, являющегося лучом, в качестве $v^*$ берется точка, лежащая на $e^*$ и имеющая достаточно большую ординату). И наконец, используются две точки множества $S$, $p_L \in S_1$ и $p_R \in S_2$, образующие отрезок $\overline{p_L p_R}$, перпендикулярный текущему ребру $e$, которое делит его пополам. Пусть $I(e, e')$ обозначает пересечение отрезков $e$ и $e'$, а запись

$I(e, e') = \Lambda$ означает, что отрезки $e$ и $e'$ не пересекаются. Как и ранее, $t_1$ и $t_2$ — это два опорных отрезка и при этом $t_1 = \overline{pq}$. Ниже представлена реализация шага 3′.

```
1.  begin p_L := p;
2.        p_R := q;
3.        e := e*;
4.        v := v*;
5.        e_L := первое ребро (открытой) границы V(p_L);
6.        e_R := первое ребро (открытой) границы V(p_R);
7.        repeat
8.          while(I(e,e_L) = Λ) do e_L := СЛЕДУЮЩИЙ_1[e_L]
                  (* посмотреть границу V(p_L) *);
9.          while(I(e,e_R) = Λ) do e_R := СЛЕДУЮЩИЙ_2[e_R]
                  (* посмотреть границу V(p_R) *);
10.         if(v ближе к I(e,e_L), чем к I(e,e_R)) then
11.           begin v := I(e,e_L);
12.                 p_L := точка множества S, находящаяся
                    по другую сторону от e_L;
13.                 e := прямая, перпендикулярная p_L p_R и
                    делящая его пополам;
14.                 e_L := обратное к e_L (* новое e_L является
                    ребром V(p_L) *)
              end
15.         else begin v := I(e,e_R);
16.                p_R := точка множества S, находящаяся
                   по другую сторону от e_R;
17.                e := прямая, перпендикулярная p_L p_R и
                   делящая его пополам;
18.                e_R := обратное к e_R
              end
19.       until(p_L p_R = t_2)
    end.
```

После инициализации (строки 1—6) начинается продвижение по цепи σ (строки 7—19). Фактический механизм подвижения по цепи описывают строки 8—18, при этом строки 8 и 9 описывают просмотр без возврата границ многоугольников $V(p_L)$ и $V(p_R)$ соответственно. Строки 11—14 и 15—18 отражают два взаимоисключающих случая изменения текущих значений параметров $\{v, p_L, p_R, e_L, e_R\}$, на выполнение каждого из которых требуется постоянное время. Так как диаграммы $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$ вместе содержат не более $3N-6$ ребер, а цепь σ содержит $O(N)$ вершин, то полное время, необходимое для построения цепи σ, является линейным. Подробное описание этой процедуры приведено в работе Ли [Lee (1978)].



Напомним, что для получения диаграммы Вороного необходимо удалить все ребра диаграммы $\mathrm{Vor}(S_1)$, расположенные справа от σ, и все ребра $\mathrm{Vor}(S_2)$, лежащие слева от σ. Операции по удалению ребер выполняются при обходе границ многоугольников по часовой стрелке и против часовой стрелки в процессе построения цепи σ. Из всего сказанного следует, что для объединения $\mathrm{Vor}(S_1)$ и $\mathrm{Vor}(S_2)$ требуется линейное время. Завершим этот раздел следующей теоремой:

**Теорема 5.15.** *Диаграмму Вороного для множества из $N$ точек на плоскости можно построить за время $O(N \log N)$, что является оптимальным.*

*Доказательство.* Время, необходимое для выполнения рекурсивной процедуры объединения диаграмм, определяется рекуррентным соотношением $T(N) = 2T(N/2) + O(N) = O(N \log N)$. Оптимальность этой процедуры следует из теоремы 5.12.

## 5.6. Решение задач о близости с помощью диаграммы Вороного

В разделе 5.1 уже говорилось, что все представленные там задачи о близости можно эффективно решить с помощью диаграммы Вороного. В этом разделе мы более подробно обсудим это утверждение на примере задач Б.1, Б.2, Б.4 и Б.5.

Начнем с задачи ВСЕ БЛИЖАЙШИЕ СОСЕДИ (Б.2), для которой справедлива следующая теорема:

**Теорема 5.16.** *Задача ВСЕ БЛИЖАЙШИЕ СОСЕДИ сводима за линейное время к задаче ДИАГРАММА ВОРОНОГО и, следовательно, может быть решена за время $O(N \log N)$, что является оптимальным.*

*Доказательство.* Согласно теореме 5.9, каждый ближайший сосед точки $p_i$ определяет ребро многоугольника $V(i)$. Чтобы найти ближайшего соседа точки $p_i$, достаточно лишь просмотреть каждое из ребер многоугольника $V(i)$. Так как каждое ребро принадлежит двум многоугольникам Вороного, то ни одно из ребер не будет просматриваться более двух раз. Таким образом, имея диаграмму Вороного, можно найти всех ближайших соседей за линейное время.

Учитывая, что задача БЛИЖАЙШАЯ ПАРА (Б.1) сводима за линейное время к задаче ВСЕ БЛИЖАЙШИЕ СОСЕДИ, то очевидно, что диаграмма Вороного может быть также использована для оптимального решения задачи Б.1.

Решая задачу поиска ближайшего соседа (задачу Б.5), мы стремимся выполнить предварительную обработку исходного множества точек, с тем чтобы по возможности быстро осуществлять поиск ближайшего соседа точки $q$, задаваемой в запросе на поиск. Но поиск ближайшего соседа точки $q$ эквивалентен поиску многоугольника Вороного, содержащего эту точку. Значит, предварительная обработка должна заключаться в построении диаграммы Вороного! Так как диаграмма Вороного является планарным прямолинейным графом, то для поиска на ней можно использовать любой из методов, представленных в разд. 2.2.2.

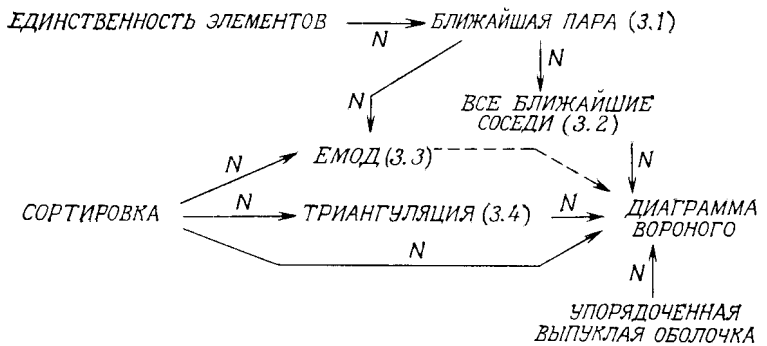



Рис. 5.30. Связь задач о близости с основными задачами-прототипами (дополненный рис. 5.6).

**Теорема 5.17.** *Поиск ближайшего соседа можно выполнить за оптимальное время $O(\log N)$, использовав для этого память объемом $O(N)$ и затратив на предварительную обработку время $O(N \log N)$.*

*Доказательство.* На построение диаграммы Вороного затрачивается время $O(N \log N)$, затем применяется теорема 2.7.

Ранее уже говорилось о том, что граф, двойственный диаграмме Вороного, является триангуляцией (триангуляция Делоне, см. теорему 5.11). Справедлива следующая теорема:

**Теорема 5.18.** *Триангуляция, в которой окружность, описанная вокруг любого треугольника, не содержит других точек, может быть найдена за время $\theta(N \log N)$, что является оптимальным для любой триангуляции.*

Хотя диаграмма Вороного может быть использована для получения в оптимальное время триангуляции множества точек и, следовательно, для решения задачи Б.4, задача о планарной триангуляции в своей общей формулировке является



значительно более широкой, и мы еще вернемся к ней в разд. 6.2. Кроме того, в одном из следующих разделов будет рассмотрена интересная взаимосвязь задач ЕВКЛИДОВО МИНИМАЛЬНОЕ ОСТОВНОЕ ДЕРЕВО и ДИАГРАММА ВОРОНОГО.

Здесь уместно вновь вернуться к диаграмме, представленной на рис. 5.6 и отражающей взаимосвязи между задачами о близости точек. На рис. 5.30 эта диаграмма показана с учетом результатов, приведенных в данном разделе. На этой диаграмме также отражен известный факт о том, что задача СОРТИРОВКА сводима за линейное время к задаче ДИАГРАММА

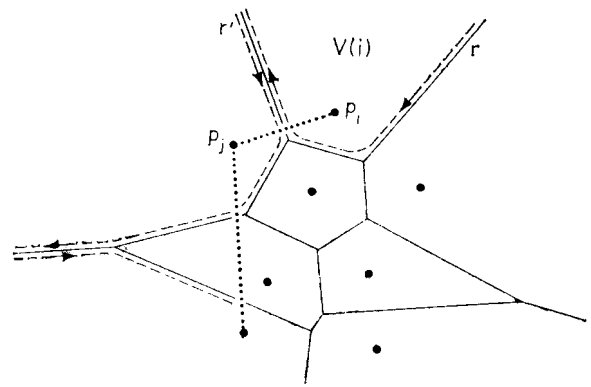



Рис. 5.31. Построение выпуклой оболочки по диаграмме Вороного.

ВОРОНОГО (теорема 5.12), и показано новое сведение задачи ВЫПУКЛАЯ ОБОЛОЧКА к задаче ДИАГРАММА ВОРОНОГО, устанавливаемое следующей теоремой:

**Теорема 5.19.** *Если имеется диаграмма Вороного множества из $N$ точек на плоскости, то выпуклая оболочка этого множества может быть найдена за линейное время.*

*Доказательство.* Пусть диаграмма Вороного представлена РСДС с ориентацией обхода циклов, составляющих грани, например, против часовой стрелки. Будем перебирать ребра диаграммы до тех пор, пока не найдется луч (рис. 5.31). Если считать $r$ ориентированным в направлении от бесконечности к его началу и при этом многоугольник $V(i)$ расположен справа от луча $r$, то точка $p_i$ является вершиной выпуклой оболочки (теорема 5.10). Будем просматривать ребра многоугольника $V(i)$ до тех пор, пока не встретится другой луч $r'$. Изменив на обратную ориентацию луча $r$, определим следующую вершину выпуклой оболочки $p_j$ и теперь будем просматривать ребра

многоугольника $V(j)$ и т. д., пока вновь не вернемся к многоугольнику $V(i)$. Ребро диаграммы просматривается лишь в том случае, если выполняется обход ребер одного из многоугольников, содержащих это ребро. Так как каждое ребро принадлежит в точности двум многоугольникам, то ни одно ребро не будет просмотрено более двух раз и время обработки является линейным.

Действительно, замечательно, что такая разнородная совокупность задач может быть решена с использованием единственной структуры. Однако возможности применения диаграммы Вороного не ограничиваются рассмотренными. При обсуждении в последующих разделах других приложений мы не перестанем удивляться многообразию использования диаграммы Вороного.

## 5.7. Замечания и комментарии

Результаты, обсуждавшиеся в предыдущих разделах этой главы, относились к задачам о близости точек в пространствах с евклидовой метрикой. Евклидова метрика может быть легко обобщена до понятия $L_p$-метрики (метрики Минковского), определяемой следующим образом:

**Определение 5.3.** $L_p$-расстояние между двумя точками $q_1$ и $q_2$ в евклидовом пространстве $E^d$ с координатами $x_1, x_2, \ldots, x_d$ для произвольного действительного числа $1 \leqslant p \leqslant \infty$ определяется следующей формулой:

$$d_p(q_1, q_2) = \left( \sum_{j=1}^{d} | x_j(q_1) - x_j(q_1) |^p \right)^{1/p}. \qquad (5.4)$$

Таким образом, традиционная евклидова метрика совпадает с $L_2$-метрикой. Все задачи, обсуждавшиеся ранее, можно рассматривать и в случае $L_p$-метрики для произвольного $p$, и такое исследование было проведено в числе прочих Ли и Вонгом [Lee, Wong (1980)] и Хвангом [Hwang (1979)]. Помимо $L_2$-расстояния особый интерес представляют $L_1$-расстояние (известное также под названием *манхеттенское расстояние*) — длина кратчайшего пути, каждый участок которого является отрезком, параллельным одной из осей координат, и $L_\infty$-расстояние, определяемое максимальной разностью координат. Эти метрики являются естественными для целого ряда приложений, таких как моделирование движений руки и проектирование топологии интегральных схем.

Естественно ожидать, что сужение некоторой задачи на ограниченный класс данных позволит получить более эффективный алгоритм ее решения. Примером такого ограничения служит ситуация, когда все $N$ точек множества $S$ являются вершинами выпуклой оболочки. В этом случае использование свойства выпуклости позволяет решить задачу ВСЕ БЛИЖАЙШИЕ СОСЕДИ в оптимальное время $\theta(N)$ [Lee, Preparata (1978)]. Около десяти лет вопрос о существовании алгоритма построения диаграммы Вороного выпуклого многоугольника с временем обработки $\theta(N)$ оставался одним из животрепещущих вопросов вычислительной геометрии. Лишь недавно был получен положительный ответ на этот вопрос [Aggarwal, Guibas, Saxe, Shor (1987)]. Предложенный ими метод основывается на рассмотренном в разд. 6.3.1.2 соответствии между диаграммами Вороного множества $S$ из $N$ точек, лежащих в плоскости $z = 1$, и разбиением пространства, определяемого плоскостями, касательными к параболоиду $z = x^2 + y^2 + 1$ в точках $\{(x_i, y_i, x_i^2 + y_i^2 + 1): (x_i, y_i) \in S\}$. В частности, проекция на плоскость $z = 1$ пересечения соответствующих полупространств, определяемых этими плоскостями, дает диаграмму Вороного ближней точки множества $S$. Если $S$ — это множество вершин выпуклого многоугольника $P$, лежащего в плоскости $z = 1$, то этот метод дает диаграмму Вороного многоугольника $P$. Ключевым моментом является тот факт, что знание порядка следования вершин при обходе границы многоугольника $P$ позволяет разработать хитроумный алгоритм вычисления требуемого пересечения полупространств за линейное время.



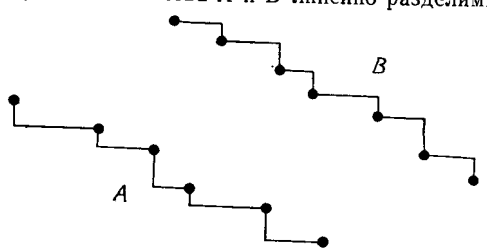
В этой главе триангуляция Делоне была введена как некоторый попутный результат, получаемый при построении диаграммы Вороного. Однако можно определить триангуляцию Делоне как единственную триангуляцию, обладающую тем свойством, что окружность, описанная вокруг любого треугольника, не содержит внутри ни одной другой точки. Исходя из такого определения, можно непосредственно построить триангуляцию Делоне, использовав для этого рекурсивную процедуру, аналогичную приведенной процедуре построения диаграммы Вороного. Как показали Ли и Шехтер [Lee, Schachter, (1980)], время работы такой процедуры соответствует оптимальной оценке. В действительности этот подход был предложен для случая $L_1$-метрики, так как для этой метрики диаграмма Вороного уже не является единственной и вследствие этого нарушается связь между диаграммами Вороного и триангуляцией Делоне.

Другой интересный класс задач касается расстояния между множествами точек. Обычно расстояние между двумя множествами точек $A$ и $B$ определяется как минимальное расстояние между точкой множества $A$ и точкой множества $B$, а именно

$d(A, B) = \min_a \min_b d(a, b)$, где $a \in A$, $b \in B$ и $d$ — евклидово расстояние. При таком определении $d(A, B) = d(B, A)$. Другая мера, представляющая интерес и не являющаяся симметричной, это *хаусдорфово* расстояние [Grünbaum (1967)]. *Хаусдорфово* расстояние от $A$ до $B$ равно $\max_a \min_b d(a, b)$, а хаусдорфово расстояние между двумя множествами $A$ и $B$ равно $\max\{d(A, B), d(B, A)\}$. В случае конечных множеств $A$ и $B$ задача БЛИЖАЙШАЯ ПАРА МЕЖДУ МНОЖЕСТВАМИ может быть решена с использованием диаграммы Вороного в оптимальное время $\theta(N \log N)$. Можно ожидать, что в случае, когда точки множества являются вершинами выпуклого или простого многоугольника, существует более быстрый алгоритм решения. Аталла [Atallah (1983)] рассматривает хаусдорфово расстояние между двумя выпуклыми многоугольниками и дает алгоритм его вычисления, имеющий сложность $O(N)$. Шварц [Shwartz (1981)] рассмотрел задачу поиска ближайшей пары точек двух выпуклых многоугольников (двух множеств, не являющихся конечными) и предложил алгоритм со сложностью $O(\log^2 N)$. Впоследствие этот результат был улучшен до $O(\log N)$ [Edelsbrunner (1982); Chin, Wang (1983)]. Для поиска ближайшей пары вершин двух выпуклых многоугольников необходимо и достаточно времени $O(N)$ [Chin, Wang (1984); Toussaint (1983a)].

## 5.8. Упражнения

1. *Ли.* На плоскости заданы два множества $A$ и $B$, содержащие по $N$ точек каждое. Найти две ближайшие точки, одна из которых принадлежит $A$, а другая $B$. Показать, что для этого требуется $\Omega(N \log N)$ операций. Что изменится, если множества $A$ и $B$ линейно разделимы?




2. *Ли.* На плоскости заданы два множества точек $A$ и $B$, каждое из которых образует «лестницу» (т. е. каждое множество совпадает с множеством своих максимумов в отношении доминирования; см. разд. 4.1.3).

(a) Найти пару точек $(p_i, p_j)$, где $p_i \in A$ и $p_j \in B$, ближайшую по $L_1$-метрике.

(b) Можно ли это сделать за линейное время?

3. *Обращение диаграммы Вороного.* Задана планарная карта валентности 3 с $N$ вершинами (каждая вершина имеет степень 3). Разработать эффективный алгоритм, проверяющий, является ли эта карта диаграммой Вороного некоторого конечного множества точек $S$. В случае положительного ответа алгоритм должен в явном виде построить множество $S$.



4. *Диаграмма Вороного на сфере.* На трехмерной сфере $S^3$ задано множество $S$ из $N$ точек. Рассмотреть диаграмму Вороного $S^3$-Vor($S$) множества $S$, построенную на сфере $S^3$ с использованием евклидовой метрики.

(a) Показать, что каждое ребро диаграммы $S^3$-Vor($S$) является дугой большой окружности сферы $S^3$.

(b) Предположим, что все точки множества $S$ находятся на полусфере сферы $S^3$ (все точки находятся по одну сторону от некоторой плоскости, проходящей через центр сферы $S^3$).

Показать, как можно использовать алгоритм построения диаграммы Вороного на плоскости (см. разд. 5.5) для построения $S^3$-Vor($S$).

5. (a) Охарактеризовать диаграмму Вороного множества из $N$ точек на плоскости в случае использования $L_1$-метрики.

(b) Тот же самый вопрос для $L_\infty$-метрики.

(c) Какова взаимосвязь диаграмм Вороного, построенных для метрик $L_1$ и $L_\infty$?

6. В случае $L_2$-метрики точки множества $S$, многоугольники Вороного которых не ограничены, образуют выпуклую оболочку множества $S$ (теорема 5.10). Имеет ли место то же самое для $L_1$-метрики?

7. Сформулировать и доказать теорему, аналогичную теореме 5.8, для $L_1$-метрики.

8. *Ли.* Рассмотреть следующее определение триангуляции Делоне множества из $N$ точек, никакие четыре из которых не лежат на одной окружности. Две точки $p_i$ и $p_j$ определяют ребро триангуляции Делоне тогда и только тогда, когда существует окружность, проходящая через эти две точки и не содержащая внутри никаких других точек множества. Показать, что это определение дает триангуляцию, удовлетворяющую условию: окружность, описанная вокруг любого треугольника, не содержит внутри никаких других точек множества. Следовательно, это то же самое, что и граф двойственный диаграмме Вороного этого множества точек.

9. *Ли.* Показать, что приведенному выше определению триангуляции Делоне соответствует единственная триангуляция (в предположении, что никакие четыре точки этого множества не лежат на одной окружности).

10. На плоскости заданы два множества точек $A$ и $B$. Использовать диаграмму Вороного для вычисления

$$\min_{a \in A} \min_{b \in B} \operatorname{dist}(a, b)$$

за время $O(N \log N)$, где $N = |A| + |B|$ (используется евклидова метрика).

ГЛАВА 6

# Близость: варианты и обобщения

Основной вывод предыдущей главы, состоит в том, что диаграмма Вороного, с одной стороны, является чрезвычайно универсальным средством для решения ряда фундаментальных задач о близости точек, с другой стороны, она сама представляет необычайно привлекательный математический объект. И именно эти две стороны — инструментальная и эстетическая — явились источником вдохновения для многочисленных исследований в этой области. В этой главе мы продемонстрируем результаты таких исследований. Начнем с обсуждения двух очень важных приложений: с упоминавшейся уже задачи о евклидовом минимальном остовном дереве (и некоторых ее вариантах) и задачи о триангуляции плоскости в общей постановке. Затем будут показаны различные обобщения понятия локуса. Завершается глава анализом класса задач, касающихся покрытий множеств, что позволит оценить возможности различных моделей вычислений.

## 6.1. Евклидовы минимальные остовные деревья

В предыдущей главе была рассмотрена задача о евклидовом минимальном остовном дереве (ЕМОД, задача Б.5, разд. 5.1) и было показано, что задача СОРТИРОВКА сводима к ней за линейное время. Это дает нижнюю оценку $\Omega(N \log N)$ для времени построения ЕМОД множества из $N$ точек. В этом разделе будет показано, что эта оценка алгоритмически достижима.

Алгоритм, который будет описан, как и любой известный алгоритм поиска минимального остовного дерева в произвольном графе, основывается на следующей довольно очевидной лемме:



**Лемма 6.1** [Prim (1957)] *Пусть $G=(V,E)$ — граф со взвешенными ребрами, а $\{V_1, V_2\}$ — разбиение множества $V$. В $G$ имеется минимальное остовное дерево, содержащее кратчайшее из ребер, одна вершина которого принадлежит $V_1$, а другая $V_2$.*

В нашем случае множество вершин $V$ — это множество точек $S$, а длина ребра равна евклидову расстоянию между его конечными точками. На каждом шаге построения ЕМОД алгоритм будет обрабатывать лес, содержащий несколько деревьев (которые впоследствии станут поддеревьями получающегося в результате ЕМОД). Первоначально лес представляет совокупность вершин (т. е. каждая точка представляет отдельное дерево без ребер). Основной шаг алгоритма выполняется следующим образом (здесь $d(u,v)$ обозначает длину отрезка $\overline{uv}$):

(1) Выбрать из леса некоторое дерево $T$.

(2) Найти ребро $(u', v')$ такое, что $d(u', v') = \min\{d(u,v): u \in T, v \in S-T\}$.

(3) Если $T'$ — дерево, содержащее $v'$, то объединить $T$ и $T'$, соединив их ребром $(u', v')$. Выполнение алгоритма завершается, когда лес содержит единственное дерево, являющееся ЕМОД исходного множества точек.

Хотя реализация операций (1) и (3) требует большой тщательности, основные трудности рассматриваемого метода связаны с выполнением операции (2).

Действительно, возникает интуитивное опасение, что объем работы, необходимой для ее выполнения, пропорционален $|T| \times (N - |T|)$, что соответствует перебору всех пар вершин. одна из которых принадлежит $T$, а вторая берется из оставшейся части. К счастью, благодаря следующей лемме, здесь нам на помощь приходит диаграмма Вороного.

**Лемма 6.2.** *Пусть $S$ — множество точек на плоскости, а $\Delta(p)$ обозначает множество точек, смежных с $p \in S$ в триангуляции Делоне множества $S$. При любом разбиении $\{S_1, S_2\}$ множества $S$, если $\overline{qp}$ — кратчайший отрезок, соединяющий точки из $S_1$ и $S_2$, то $q$ принадлежит $\Delta(p)$.*

*Доказательство.* Пусть на отрезке $\overline{pq}$ достигается минимальное расстояние между точками $S_1$ и $S_2$ и при этом $p \in S_1$, а $q \in S_2$ (рис. 6.1). Утверждается, что $q \in \Delta(p)$. Если $q \notin \Delta(p)$, то перпендикуляр к отрезку $\overline{pq}$, делящий его пополам, не содержит отрезка границы $V(p)$ — многоугольника Вороного точки $p$ (теорема 5.9). Отсюда следует, что $V(p)$ пересекает $\overline{pq}$ в некоторой точке $u$, расположенной между $p$ и точкой $M$ — серединой отрезка $\overline{pq}$. Ребро $l$ диаграммы Вороного, содержащее точку $u$, перпендикулярно отрезку $\overline{pp'}$, $p' \in S$ и делит его

пополам. При любом возможном выборе $u$ и $l$ положение точки $p'$ ограничивается внутренностью круга $C$ с центром в точке $M$ и диаметром $\overline{qp}$. Возможны два случая: (1) $p' \in S_1$, в этом случае $d(q, p') < d(q, p)$, так как ближе к $q$ находится $p'$, а не $p$ (противоречие); (2) $p' \in S_2$, в этом случае $d(p, p') < d(q, p)$, так как $p'$ ближе к $p$, чем $q$ (снова противоречие).

Другими словами, согласно этой лемме, необходимо просматривать лишь ребра триангуляции Делоне, т. е. необходимо искать минимальное остовное дерево планарного графа [Shamos (1978)].

Вернемся вновь к реализации основного шага алгоритма построения ЕМОД. Черитон и Тарьян [Cheriton, Tarjan (1976)] предложили в числе других методов следующую простую стратегию. Для выбора дерева $T$ (которое будет объединено с другим деревом леса) они указали однородное правило выбора, перед использованием которого в очередь помещаются все деревья, содержащие по одной вершине.

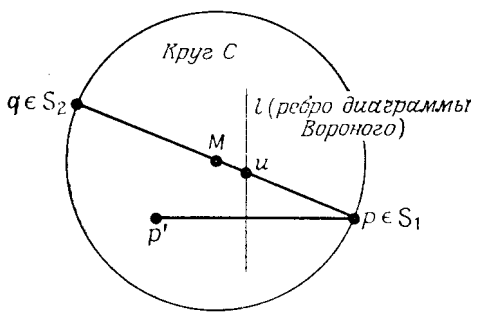



Рис. 6.1. Иллюстрация к доказательству условия $q \in \Delta(p)$.

1. Выбрать дерево из начала очереди.

2. Если дерево $T''$ получено в результате объединения дерева $T$ с некоторым другим деревом $T'$, то удалить $T$ и $T'$ из очереди и добавить $T''$ в конец очереди.

Так как каждый раз производится объединение двух деревьев и исходно в очереди имеются $N$ деревьев, то основной шаг алгоритма выполняется в точности $(N-1)$ раз.

Определим для каждого дерева $T$, находящегося в очереди, целое число, называемое *этапом* и вычисляемое следующим образом: $этап(T) = 0$, если $|T| = 1$, и $этап(T) = \min(этап(T'), этап(T'')) + 1$, если $T$ является результатом объединения деревьев $T'$ и $T''$. Очередь деревьев обладает интересным свойством: в любой момент работы алгоритма совокупность чисел $этап(T)$ в порядке от начала к концу очереди образует неубывающую последовательность. Будем говорить, что *этап* $j$ завершен, если из очереди удалено дерево $T$, для которого $этап(T) = j$, и ни для одного другого дерева $T'$ в очереди значение $этап(T')$ не равно $j$. Заметим, что после завершения *этапа* $j$ очередь содержит не более $N/2^{j+1}$ элементов, а всего



имеется не более $\lceil \log_2 N \rceil$ этапов[1]). Последующая обработка происходит следующим образом. Выбрав из очереди первый элемент $T$ (при этом $этап(T) = j$), среди ребер, соединяющих вершины дерева $T$ с вершинами вне его, ищем ребро наименьшей длины. При такой организации обработки на каждом этапе каждое ребро просматривается не более двух раз (за исключением ребер, уже вошедших в некоторое дерево $T$). Таким образом, каждый этап обработки может быть выполнен за время, пропорциональное числу ребер, которое в силу планарности графа равно $O(N)$. А так как число этапов не превосходит $\lceil \log_2 N \rceil$, то для времени работы алгоритма получаем оценку $O(N \log N)$. В этом месте читатель может заметить, что мы могли бы удовлетвориться полученным результатом, так как время, затрачиваемое на построение триангуляции Делоне, равно $O(N \log N)$, и во всяком случае мы имеем оптимальный алгоритм. Однако интересно найти метод, позволяющий получить ЕМОД из диаграммы Вороного за *линейное время*. Этого можно достичь с помощью операции «очистки», предложенной Черитоном и Тарьяном.

Применение *операции очистки* имеет целью сжать исходный граф $G$ (в нашем случае это граф триангуляции Делоне), преобразовав его в некоторый граф $G^*$, который в любой момент работы алгоритма содержит *лишь необходимую информацию*. Это значит, что каждое дерево $T$, входящее в лес $F$, сжимается в единственную вершину графа $G$ (т. е. удаляются все неотобранные ребра, соединяющие вершины дерева $T$ (хорды)) и, кроме того, удаляются все ребра, за исключением самого короткого из неотобранных, соединяющие деревья $T'$ и $T''$. Важно отметить, что если $G$ — планарный граф, то $G^*$ тоже будет планарным.

Очистка графа производится сразу же после завершения этапа (см. выше). Не составляет труда предложить такую реализацию, при которой время обработки пропорционально числу ребер графа, удаляемых при очистке.

В заключение приведем формальное описание процедуры построения ЕМОД. Предполагается, что с каждым деревом $T$ связан список неотобранных ребер, инцидентных вершинам дерева $T$.

```
    procedure ЕМОД
1.  begin F := ∅;
2.      for i := 1 to N do
3.          begin этап(p_i) := 0;
                  F ⇐ p_i
```

[1]) То есть $этап(T)$ может принимать не более $\lceil \log_2 N \rceil$ различных значений. — *Прим. перев.*

```
        end; (* очередь инициализирована *)
4.      j := 1; (* номеру этапа присваивается следующее
        значение *)
5.      while (F содержит более одного элемента) do
6.          begin T ⇐ F;
7.              if (этап(T) = j) then begin очистить граф;
8.                                         j := j + 1
                                       end;
9.              (u, v) := кратчайшее из неотобранных ребер,
                инцидентных T (u принадлежит T);
10.             T' := дерево в F, содержащее v;
11.             T'' := объединить(T, T');
12.             удалить T' из F;
13.             этап(T'') := min(этап(T), этап(T')) + 1;
14.             F ⇐ T'' (* T'' добавляется в конец очереди *)
        end
    end.
```

Теперь у нас есть все необходимое для анализа алгоритма. По завершении этапа $(j-1)$ в очереди имеется не более $N/2^j$ деревьев, и, следовательно, соответствующий граф $G^*$ имеет не более $N/2^j$ вершин и менее $3N/2^{j-1}$ ребер, так как он является планарным. Этап $j$ завершается за время, пропорциональное числу ребер (каждое ребро просматривается на этапе не более двух раз (строка 9)), и то же самое справедливо для времени, затрачиваемого на очистку графа, которая производится по завершении этапа $j$ (строки 7, 8 выполняются непосредственно перед переходом к этапу $(j+1)$). Таким образом, однократное выполнение тела цикла (строки 6—14) происходит за время $O(N/2^j)$. И если вспомнить, что число этапов не превосходит $\lceil \log_2 N \rceil$, получаем следующую верхнюю оценку для полного времени выполнения цикла 6—14:

$$\sum_{j=1}^{\lceil \log_2 N \rceil} K_1 \frac{N}{2^{j-1}} < 2K_1 N,$$

где $K_1$ — некоторая константа. Тем самым доказана следующая теорема.

**Теорема 6.1.** *ЕМОД множества $S$ из $N$ точек на плоскости может быть построено, исходя из триангуляции Делоне множества $S$, за оптимальное время $\theta(N)$.*

Объединяя этот результат с уже известным нам фактом о том, что триангуляция Делоне может быть построена за время $\theta(N \log N)$, получаем

**Следствие 6.1.** *ЕМОД множества $S$ из $N$ точек на плоскости может быть построено за оптимальное время $\theta(N \log N)$.*



На рис. 6.2 показаны множество точек, его диаграмма Вороного и ЕМОД этого множества.

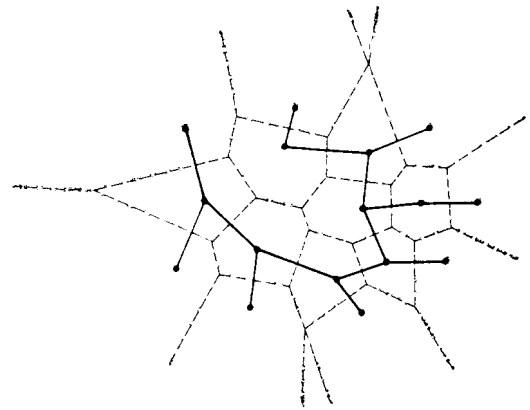


Рис. 6.2. Множество точек и соответствующие ему диаграмма Вороного и ЕМОД.

Далее будет рассмотрено одно интересное приложение ЕМОД конечного множества точек.

### 6.1.1. Евклидова задача о коммивояжере

**Задача Б.9** (ЕВКЛИДОВА ЗАДАЧА О КОММИВОЯЖЕРЕ). Найти кратчайший замкнутый путь, проходящий через $N$ заданных точек на плоскости.

Пример такого кратчайшего маршрута показан на рис. 6.3. (Читатель, представляющий сложность этой задачи, может удивиться тому, как удалось получить решение для приведенного примера, содержащего 16 точек. Это было сделано с помощью эвристического приема, предложенного Кристофидесом, который будет описан далее.)

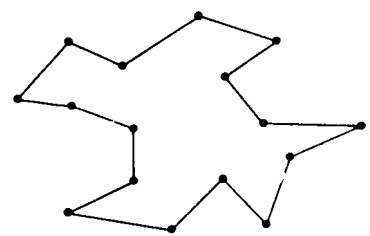


Рис. 6.3. Маршрут коммивояжера.

Отличие этой задачи от задачи о коммивояжере в общей постановке имеет ту же природу, что и отличие задачи о евклидовом остовном дереве от задачи о минимальном остовном дереве в произвольном графе. А именно расстояния между точками не являются произвольными, а определяются евклидовой

метрикой. Как известно, задача о коммивояжере в общей постановке (ЗК) является *NP*-трудной [1]).

Можно было бы предположить, что свойства евклидовой метрики могут быть использованы для разработки полиномиального алгоритма решения этой задачи в случае множества точек, заданного на плоскости. Однако Гэри, Грэхем и Джонсон [Garey, Graham, Johnson (1976)], Пападимитриу и Стайглиц [Papadimitriou, Steiglitz (1976)] и Пападимитриу [Papadimitriou (1976)] наряду с другими результатами о *NP*-полноте некоторых геометрических задач доказали, что евклидова задача о коммивояжере (ЕЗК) является *NP*-трудной. Учитывая это, мы не будем обсуждать вопрос разработки эффективного

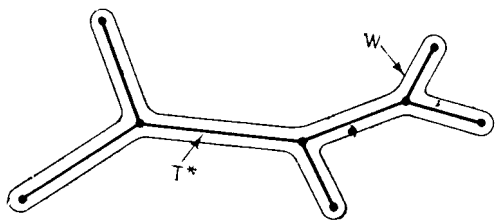



Рис. 6.4. Продублировав все ребра ЕМОД, получаем граф с эйлеровым циклом.

в худшем случае алгоритма для ЕЗК, а рассмотрим взаимосвязь ЕЗК с другими задачами о близости точек с точки зрения разработки хороших приближенных методов.

Начнем с рассмотрения более простого (и менее эффективного) из двух известных методов, основанного на следующей теореме:

**Теорема 6.2** [Rosenkrantz, Stearns, Lewis (1974)]. *Используя минимальное остовное дерево, можно получить приближенное решение* ЗК, *дающее путь, длина которого меньше удвоенной длины кратчайшего пути.*

*Доказательство.* Пусть $T^*$ обозначает евклидово минимальное остовное дерево заданного множества точек $S$, а $\Phi$ — оптимальный маршрут коммивояжера. Начнем с того, что заменим каждое ребро дерева $T^*$ парой ребер. В результате получим граф $W$, каждая вершина которого имеет четную степень (рис. 6.4). Граф $W$ является связным эйлеровым графом. Это значит, что его ребра можно занумеровать таким образом, что получившаяся последовательность образует эйлеров цикл,

[1]) Сведения, касающиеся *NP*-полных задач, а также доказательство того, что ЗК является *NP*-трудной, можно найти в [Karp (1972), Garey, Johnson (1979)].



или, иначе говоря, маршрут, проходящий через все вершины (при этом через некоторые вершины он проходит не один раз). Следовательно, длина$(W) = 2$ длина$(T^*)$. Заметим, что если удалить из $\Phi$ некоторое ребро $e$, то получим путь $\Phi_{\text{путь}}$, который также является остовным деревом множества $S$. Очевидно, что длина$(T^*) \leqslant$ длина$(\Phi_{\text{путь}})$ (по определению $T^*$) и длина$(\Phi_{\text{путь}}) <$ длина$(\Phi)$. Отсюда получаем длина$(W) < 2$ длина$(\Phi)$.

Заметим, что приближенный маршрут, показанный на рис. 6.4, можно улучшить, сделав его более коротким, удалив все ненужные остановки, т. е. при обходе точек ни одна из уже пройденных точек не проходится вновь. Более конкретно это

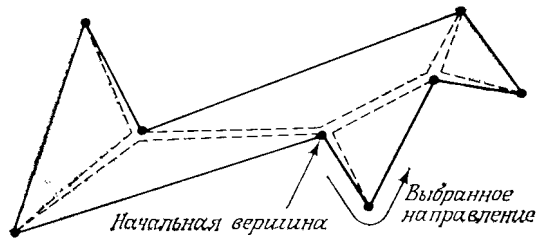



Рис. 6.5. Введение «хорд» в эйлеровом цикле обеспечивает однократное посещение каждой вершины.

делается так. На эйлеровом цикле $W$ выбираются произвольным образом направление его обхода и начальная вершина (которую мы оставим непомеченной). Каждая пройденная вершина помечается, а следующей вершиной окончательного маршрута является первая после нее непомеченная вершина, встретившаяся при обходе $W$ в установленном направлении. Применение такой процедуры удаления повторений вершин к примеру на рис. 6.4 дает маршрут, показанный на рис. 6.5. (Заметим, что эта процедура удаления не может увеличить длину получаемого маршрута, так как расстояния между точками удовлетворяют неравенству треугольника.)

Следующий результат, касающийся приближенных методов, использует понятие *минимального взвешенного паросочетания.*

**Задача Б.10** (МИНИМАЛЬНОЕ ЕВКЛИДОВО ПАРОСОЧЕТАНИЕ). На плоскости заданы $2N$ точек. Объединить их в пары, соединив отрезками так, чтобы сумма длин отрезков была минимальной.

Пример такого паросочетания показан на рис. 6.6.

Едмондс [Edmonds (1965)] показал, что минимальное взвешенное паросочетание в произвольном графе может быть най-

дено за полиномиальное время, а Габоу [Gabow (1972)] разработал алгоритм со сложностью $O(N^3)$. Следующий результат, полученный Кристофидесом [Christofides (1976)], связывает минимальные остовные деревья, паросочетания и задачу о коммивояжере.

**Теорема 6.3.** *Если расстояния между точками удовлетворяют неравенству треугольника, то за время $O(N^3)$ можно найти приближенное решение задачи о коммивояжере, представляющее*

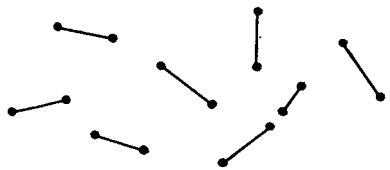


Рис. 6.6. Минимальное евклидово паросочетание.

*маршрут, длина которого не превосходит 3/2 длины минимального маршрута.*

*Доказательство.* Пусть задано множество точек $S$. Следующий алгоритм соответствует условиям теоремы и дает необходимый результат.

1. Найти минимальное остовное дерево $T^*$ множества $S$.
2. Найти минимальное евклидово паросочетание $M^*$ для множества $X \subseteq S$ вершин, имеющих *нечетную* степень в $T^*$. (Множество $X$ всегда содержит четное число элементов для любого графа.)
3. Граф $T^* \cup M^*$ является эйлеровым, так как все его вершины имеют четную степень. Обозначим через $\Phi_e$ эйлеров цикл этого графа.
4. Продвигаясь по $\Phi_e$ от ребра к ребру таким образом, чтобы не проходить вновь уже пройденные вершины, построить приближенный маршрут $\Phi_{\text{прибл}}$.

Пусть, как обычно, $\Phi$ — минимальный маршрут. Заметим, что: (1) длина $(T^*) <$ длина $(\Phi)$; (2) длина $(M^*) \leqslant \frac{1}{2}$ длина $(\Phi)$. Действительно, выбрав из $\Phi$ каждое второе ребро, получаем два паросочетания на множестве $S$ (к первому относятся отобранные ребра, а ко второму оставшиеся). Длина кратчайшего из двух паросочетаний не превосходит 1/2 длина $(\Phi)$ и со всей очевидностью не меньше длина $(M^*)$, так как $M^*$ — минимальное паросочетание. Ранее было показано, что длина $(T^*) <$ $<$ длина $(\Phi)$. И так как расстояния между точками удовлетворяют неравенству треугольника, длина $(\Phi_{\text{прибл}}) \leqslant$ длина $(\Phi_e)$.



Объединяя эти результаты, получаем

$$\begin{aligned} \text{длина } (\Phi_{\text{прибл.}}) &\leqslant \text{длина } (\Phi_e) \\ &= \text{длина } (T^*) + \text{длина } (M^*) \\ &< \text{длина } (\Phi) + \tfrac{1}{2} \text{ длина } (\Phi) \\ &= \tfrac{3}{2} \text{ длина } (\Phi). \end{aligned}$$

Справедливость временно́й оценки следует из результата, полученного Габоу.

**Замечание.** Воспользовавшись минимальным евклидовым паросочетанием, можно получить лучшее приближенное решение, отличающееся от минимального не в 2, а в 3/2 раза. При этом сложность вычислений возрастает с $O(N \log N)$ до $O(N^3)$. За несколько лет, прошедших с момента, когда Кристофидес получил свой результат, никому не удалось ни найти способ поиска приближенного решения, отличающегося от минимального менее чем в 3/2 раза, ни уменьшить время поиска приближенного решения. Само по себе примечательно, что метод, используемый для получения минимального евклидова паросочетания, не использует геометрические свойства. Действительно, единственный известный метод заключается в сведении (преобразовании) данной задачи к задаче поиска максимального взвешенного паросочетания (сведение осуществляется путем замены длины $l$ каждого ребра величиной $M - l$, где $M = \max_{\text{по всем ребрам}} l$) и применении к полученной в результате сведения задаче метода, разработанного для произвольных графов.

## 6.2. Планарные триангуляции

В разд. 5.1 уже отмечалась важность триангуляций для многочисленных приложений, связанных с интерполяцией поверхностей, как в задачах графического отображения данных, так и в задачах численного анализа. В дополнение к этим очень важным приложениям возможность выполнять планарные триангуляции сама по себе является полезным средством, используемым для решения других задач вычислительной геометрии. Достаточно упомянуть два примера: (1) в методе геометрического поиска Киркпатрика (см. разд. 2.2.2.3) предполагается, что планарное разбиение является триангуляцией; (2) алгоритм построения пересечения многогранников (см. разд. 7.3.1) требует выполнить предварительную триангуляцию поверхности многогранников.

В предыдущей главе уже было показано, что триангуляция — а именно триангуляция Делоне — множества точек мо-

жет быть найдена за оптимальное время. Однако это не решает проблемы, так как, хотя триангуляция Делоне является замечательным объектом, обладающим очень привлекательными свойствами, могут встретиться приложения, для которых триангуляция Делоне не подходит. Например, может потребоваться минимизировать суммарную длину ребер триангуляции (минимальная взвешенная триангуляция). Было сделано предположение, что триангуляция Делоне также является минимальной взвешенной триангуляцией. Еще в одном предположении утверждалось, что метод триангуляции, который будет описан в разд. 6.2.1 (метод «жадной триангуляции»), также дает минимальную взвешенную триангуляцию. Оба этих утверждения были опровергнуты Ллойдом [Lloyd (1977)], и на сегодня вопрос о сложности задачи построения минимальной взвешенной триангуляции (т. е. является ли она $NP$-трудной или существует полиномиальный алгоритм для ее решения) остается открытым [1]).

Другие критерии не связаны с длиной ребер, а касаются величины внутренних углов треугольников. Во многих приложениях требуется, чтобы треугольники были по возможности «однородными», т. е. были «более или менее равносторонними». С этой точки зрения триангуляция Делоне является очень привлекательной, так как для нее минимальное значение угла по всем треугольникам является максимальным среди всех триангуляций. (Действительно, это эквивалентно тому, что окружность, описанная вокруг треугольника в триангуляции Делоне, не содержит внутри других точек заданного множества [Lawson (1977); Lee (1978)].)

Отсылаем читателя к разд. 5.6, где подробно рассмотрены свойства триангуляции Делоне. Далее будут рассмотрены некоторые другие методы триангуляции применительно либо к множествам точек, либо к смешанным множествам, содержащим точки и отрезки. Здесь стоит напомнить, что для построения триангуляции множества из $N$ точек на плоскости любым из алгоритмов требуется $\Omega(N \log N)$ операций (см. разд. 5.3).

### 6.2.1. Жадная триангуляция

Как хорошо известно, «жадный» метод — это такой метод, при котором никогда не отменяется то, что уже было сделано ранее. Таким образом, жадный метод триангуляции последовательно порождает ребра триангуляции (по одному за раз) и завершает этот процесс после того, как порождено необходимое число ребер, которое полностью определяется размером множества точек и его выпуклой оболочкой. Если цель состоит в минимизации суммарной длины ребер, то все, что можно сделать, используя жадный метод, это применить локальный критерий, добавляя на каждом этапе кратчайшее из возможных ребер, совместимое с ранее порожденными ребрами, т. е. не пересекающее ни одно из них.

[1]) Однако известно, что минимальная взвешенная триангуляция индивидуального $N$-угольника может быть вычислена за время $O(N^3)$ [Gilbert (1979)].



В этом суть метода. Непосредственная реализация этой идеи сводится к следующей последовательности действий (как обычно, $S$ обозначает множество точек, а $N$ — его размер). Сначала порождается множество, содержащее все $\binom{N}{2}$ ребер, соединяющих точки множества $S$, которые затем упорядочиваются по возрастанию длин, образуя пул ребер. Вначале множество ребер триангуляции является пустым. Основной шаг процедуры триангуляции включает следующие действия. В пуле выбирается и исключается из него ребро наименьшей длины. Если это ребро не пересекает ни одно из ребер, уже вошедших в триангуляцию, то оно также включается в число ребер триангуляции. Иначе ребро просто исключается из дальнейшего рассмотрения. Процесс завершится, либо когда будет построена триангуляция (что можно определить, подсчитывая число добавляемых ребер), либо когда пул ребер станет пустым [Düppe, Gottschalk (1970)].

Этот метод, рассмотренный здесь лишь в общих чертах, прост не только для реализации, но и для анализа. Для сортировки ребер по длине требуется $O(N^2 \log N)$ операций. Затем $\binom{N}{2}$ раз производится выборка ребра из пула и каждое выбранное ребро проходит проверку на пересечение с ребрами, уже вошедшими в триангуляцию, на что требуется некоторое время $\varphi(N)$. Вид $\varphi$ зависит от метода, реализующего эту проверку. В результате получаем для времени, необходимого для этого метода, следующую оценку: $O(N^2 \log N + N^2 \varphi(N))$.

Простейший способ выполнить упомянутую проверку — это проверить, пересекает ли выбранное ребро каждое из $k$ ребер, уже вошедших в триангуляцию. Так как каждая из этих элементарных проверок выполняется за постоянное время, то для $\varphi$ имеем $\varphi(k) = C_1 k$ (где $C_1$ — некоторая константа). Следовательно, полное время обработки равно $O(N^3)$.

Более эффективный подход был предложен Джильбертом [Gilbert (1979)]. Цель, ставившаяся при разработке этого подхода, заключалась в том, чтобы сбалансировать работу по принятию решений (принадлежит ли выбранное ребро триангуляции) и работу по выбору ребер (просмотр ребер в порядке

увеличения длин). Ясно, что затраты на последнюю из работ составляют $O(N^2 \log N)$ независимо от того, используется предварительная сортировка или подход на основе пирамиды (пирамидальная сортировка) [Aho, Hopcroft, Ullman (1974)]. Поэтому нужно добиться, чтобы на обработку ребра требовалось не более $O(\log N)$ операций. А это можно сделать следующим образом. Предположим, что выбрано ребро $\overline{p_1p_i}$ (рис. 6.7(a)).

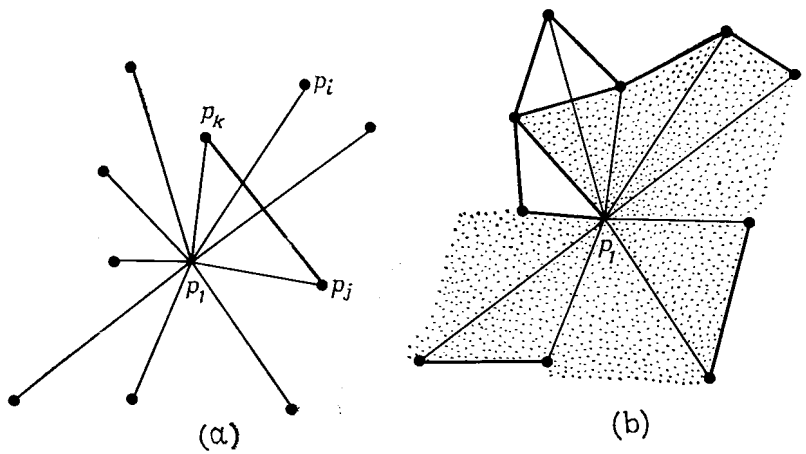



Рис. 6.7. «Звезда из спиц» в точке $p_1$. Ребра триангуляции показаны жирными отрезками.

Рассмотрим множество ребер, соединяющих $p_1$ со всеми другими точками множества $S$, и обозначим его ЗВЕЗДА($p_1$). Это множество образует «звезду из спиц» в точке $p_1$. Эти спицы упорядочены в порядке обхода вокруг точки $p_1$, например против часовой стрелки, и делят полный угол в $2\pi$ радиан на $(N-1)$ секторов, или угловых интервалов. Если некоторое ребро $\overline{p_kp_j}$ уже включено в триангуляцию, то оно проходит через несколько *последовательных* секторов звезды в $p_1$, и то же самое имеет место для любой точки $p_l$ ($l \neq k, j$). Заметим также, что никакие два ребра, уже включенные в триангуляцию, не пересекаются (по определению), так что ребра, попадающие в каждый угловой интервал, определяемый ЗВЕЗДА($p_l$), где $l = 1, \ldots, N$, *упорядочены*[1]). Предположим, точка $p_i$ попала в сектор $p_jp_1p_k$, определяемый ЗВЕЗДА($p_1$). Чтобы решить, следует ли включать ребро $\overline{p_1p_i}$ в триангуляцию, необходимо проверить, пересекает ли оно ребра триангуляции, проходящие через этот сектор, и, в частности, *ближайшее* из них к $p_1$ ребро. Таким образом, проверка пересечения сводится к частному случаю задачи определения положения точки на плоскости, когда в каждом секторе, определяемом ЗВЕЗДА($p_l$) (для всех $l$), необходимо хранить одно ребро триангуляции (называемое опорным ребром). Типичная ситуация показана на рис. 6.7(b), где затемнена область, допустимая для ребер триангуляции.

[1]) Имеется в виду упорядоченность ребер внутри сектора по их расстоянию от точки $p_e$. — *Прим. перев.*



Структура данных, используемая при проверке приемлемости ребер, является динамической, так как она должна изменяться (перестраиваться) каждый раз, как некоторое ребро включается в триангуляцию. Учитывая, однако, что в каждой звезде изменения затрагивают *множество последовательных секторов*, вполне естественно использовать деревья отрезков (см. разд. 1.2.3.1). В частности, секторы, порождаемые ЗВЕЗДА($p_l$), представлены $(N-1)$ листьями дерева отрезков $T_l$. С каждым узлом $v$ дерева $T_l$ связано ребро $e(v)$, являющееся ближайшим ребром к $p_l$ среди всех ребер, которые связывались с узлом $v$ в результате вставки в дерево отрезков. (Очевидно, что $e(v)$ может быть пустым ребром.) При проверке ребра $\overline{p_ip_j}$ выполняется поиск в дереве $T_l$, осуществляемый с помощью прохождения пути, определяемого $p_j$. В каждом узле $v$, лежащем на этом пути, производится проверка взаимного положения точки $p_j$ и ребра $e(v)$. Ребро считается допустимым тогда и только тогда, когда оно не пересекает ни одно из ребер, встретившихся при прохождении пути. Если выбранное ребро оказывается непригодным, то поиск прекращается. Иначе, когда ребро $\overline{p_ip_j}$ является допустимым, необходимо произвести изменения: перестроить деревья, соответствующие каждому из множеств ЗВЕЗДА($p_l$), где $l \neq i, j$. Это реализуется просто путем вставки ребра $p_ip_j$ в дерево для ЗВЕЗДА($p_l$) и надлежащего изменения каждого узла $v$, лежащего на путях, проходимых при вставке. (Отсылаем читателя к разд. 1.2.3, где подробно рассмотрена работа с деревьями отрезков.) Очевидно, что изменения в каждом из деревьев, соответствующих ЗВЕЗДА($p_l$), $l = 1, \ldots, N$, можно сделать за время $O(\log N)$. Так как, если некоторое ребро оказывается допустимым, необходимо произвести изменения в $(N-2)$ деревьях и эта операция выполняется $O(N)$ раз, то полное время, необходимое на выполнение изменений в деревьях при триангуляции, составляет $O(N^2 \log N)$.

Учитывая, что на управление выбором ребер, проверку допустимости ребер и внесение изменений в структуры данных требуется время $O(N^2 \log N)$ (на каждую в отдельности), получаем следующий результат:

**Теорема 6.4.** *Триангуляция множества из $N$ точек жадным методом может быть выполнена за время $O(N^2 \log N)$ с использованием памяти объемом $O(N^2)$.*

### 6.2.2. Триангуляция с ограничениями

Во многих случаях природа решаемой задачи такова, что при триангуляции необходимо учитывать разного рода ограничения, т. е. в исходной постановке задачи на множество ребер триангуляции накладываются некоторые ограничения. Типичный пример такой ситуации дает задача триангуляции внутренности простого многоугольника.

Из всех описанных ранее методов для решения таких задач с ограничениями можно было бы использовать жадный метод, но он имеет один недостаток: эффективность получения решения далека от оптимальной. С другой стороны, не видно способа приспособить для решения таких задач метод триангуляции Делоне. (См. разд. 5.6 и 6.5.) Следовательно, необходимо искать новый метод.

Как обычно, на плоскости задано множество $S$ из $N$ точек. Кроме того, на $S$ задано множество $E$ из $M$ непересекающихся ребер (ограничения), которые должны войти в число ребер триангуляции. Мы немного и несущественно изменим предыдущее соглашение, условившись, что область, подвергаемая триангуляции, представляет наименьший прямоугольник $\mathrm{rect}(S)$, стороны которого параллельны координатным осям, охватывающий все точки множества $S$. (Заметим, что в $\mathrm{rect}(S)$ существуют по крайней мере две вершины, имеющие наибольшее значение $y$-координаты, и по крайней мере две вершины с наименьшей $y$-координатой.)

Цель состоит в том, чтобы эффективно выполнить разбиение $\mathrm{rect}(S)$ на многоугольники меньшего размера, триангуляция которых не вызывает затруднений. Следующее определение описывает класс таких многоугольников [Garey, Johnson, Preparata, Tarjan (1978)].

**Определение 6.1.** Многоугольник $P$ называется *монотонным* относительно прямой $l$, если он простой и его граница является объединением двух цепей, монотонных относительно $l$ (см. разд. 2.2.2.2.)[1]).

Будем предполагать далее, что прямая $l$ — это $y = 0$ системы координат и что никакие две точки множества $S$ не имеют одинаковую ординату (как обычно, это предположение

[1]) Было показано, что проверка монотонности простого многоугольника может быть выполнена за линейное время [Preparata, Supowit (1981)].



не принципиально, но упрощает рассмотрение). Далее будет представлен простой алгоритм триангуляции монотонного многоугольника, а сейчас покажем, как можно выполнить разбиение прямоугольника $\mathrm{rect}(S)$.

Предлагаемый метод принадлежит классу методов, основанных на заметании плоскости, и, по существу, совпадает с процедурой «регуляризации» планарного графа, описанной в разд. 2.2.2.2. Действительно, множества $S$ и $E$ определяют планарный граф, уложенный на плоскости. Процедура регуляризации (описание которой здесь не приводится, чтобы не делать ненужных повторений) за время $O(N \log N)$ добавляет ребра так, что выполняются следующие условия:

(1) никакие два ребра не пересекаются (за исключением, возможно, в вершинах);

(2) каждая вершина (за исключением вершин с наибольшей $y$-координатой) непосредственно соединена по крайней мере с одной вершиной, имеющей бóльшую $y$-координату;

(3) каждая вершина (за исключением вершин с наименьшей $y$-координатой) непосредственно соединена по крайней мере с одной вершиной, имеющей меньшую $y$-координату.

Утверждается, что каждая область планарного графа, полученная в результате применения процедуры регуляризации, является монотонным многоугольником. Доказательство этого утверждения основывается на понятии внутреннего пика. Вершина $v$ простого многоугольника называется *внутренним пиком*, если внутренний угол в вершине $v$ превышает $\pi$ и обе смежные с ней вершины имеют $y$-координату, не превосходящую $y$-координату вершины $v$. Из условий (2) и (3) следует, что ни одна вершина регуляризованного графа не может быть внутренним пиком. Ключ к доказательству сделанного утверждения дает следующая лемма:

**Лемма 6.3.** *Если $P$ — простой многоугольник без внутренних пиков, то $P$ — монотонный относительно $y$-оси многоугольник.*

*Доказательство.* Пусть $v_1, v_2, \ldots, v_m$ — последовательность вершин многоугольника $P$ в порядке обхода по часовой стрелке, а $v_1$ и $v_s$ — вершины с наибольшей и наименьшей $y$-координатами соответственно (рис. 6.8). Если многоугольник $P$ не монотонный, то по крайней мере одна из двух цепей из $v_1$ в $v_s$, образованных ребрами многоугольника $P$, не является строго убывающей по $y$-координате. Рассмотрим случай, когда это — цепь, проходящая через вершину $v_2$ (другой случай полностью симметричен данному). Выберем в качестве первой вершины на этом пути $v_i$, $1 < i < s$, такую, что $y$-координата вершины $v_{i+1}$ превосходит $y$-координату вершины $v_i$.

Для начала заметим, что последовательность из трех вершин $(v_{i-1}v_iv_{i+1})$ должна образовывать правый поворот, так как иначе $v_i$ была бы внутренним пиком многоугольника $P$ (рис. 6.8(a)). Рассмотрим теперь прямую, проходящую через вершины $v_1$ и $v_s$ (рис. 6.8(b)). Пусть $r \neq v_i$ — первая точка на границе многоугольника $P$, встретившаяся при движении из $v_i$ в $v_s$ по прямой, проходящей через $v_i$ и $v_s$ ($r$ может совпадать

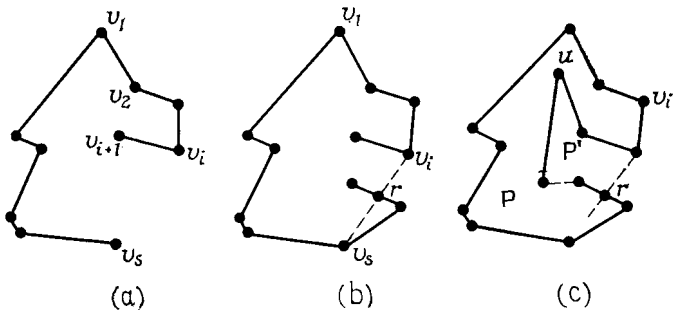



Рис. 6.8. (a) — $v_i$ — первая вершина в последовательности $v_1, v_2, \ldots$, для которой $y(v_{i+1}) > y(v_i)$; (b) — $r$ — первая точка пересечения прямой, идущей из $v_i$ в $v_s$, с границей многоугольника $P$; (c) — вершина $u$, имеющая в многоугольнике $P$ наибольшую $y$-координату, является внутренним пиком многоугольника $P$.

с $v_s$). Отрезок, соединяющий $v_i$ и $r$, делит область вне многоугольника $P$ на две части, одна из которых представляет ограниченный многоугольник $P'$ (рис. 6.8(c)). Все вершины многоугольника $P'$, за исключением $r$, являются вершинами многоугольника $P$. Пусть $u$ — вершина многоугольника $P'$, имеющая наибольшую $y$-координату. Тогда $u$ является также и вершиной многоугольника $P$, и, кроме того, является внутренним пиком $P$ (противоречие).

Лемма 6.3 и тот факт, что ни один из многоугольников, полученных в результате регуляризации, не имеет внутренних пиков, доказывают утверждение о монотонности каждого из многоугольников. Теперь можно обратиться к вопросу триангуляции монотонного многоугольника [Garey, Johnson, Preparata, Tarjan (1978)].

### 6.2.2.1. Триангуляция монотонного многоугольника

Так как многоугольник $P$ монотонный относительно $y$-оси, то путем слияния двух монотонных цепей, образующих границу $P$, можно за время $O(N)$ упорядочить вершины $P$ в порядке уменьшения $y$-координаты. Пусть $u_1, u_2, \ldots, u_N$ — последовательность вершин, полученная в результате упорядоче-



ния. Из монотонности следует, что для каждой вершины $u_i$ $(1 \leqslant i \leqslant N-1)$ имеется вершина $u_j$, где $i < j$, такая, что $\overline{u_iu_j}$ является ребром многоугольника $P$.

Алгоритм триангуляции обрабатывает по одной вершине за раз в порядке уменьшения $y$-координаты. В ходе этой обработки порождаются *диагонали* многоугольника $P$. Каждая диагональ ограничивает треугольник, оставляя для последующей триангуляции многоугольник, имеющий на одну сторону меньше. Алгоритм использует стек, содержащий пройденные вершины, но которые еще не отсечены диагоналями. Алгоритму доступны как самый верхний (верх), так и самый нижний (низ) элементы стека (нижний только для контроля).

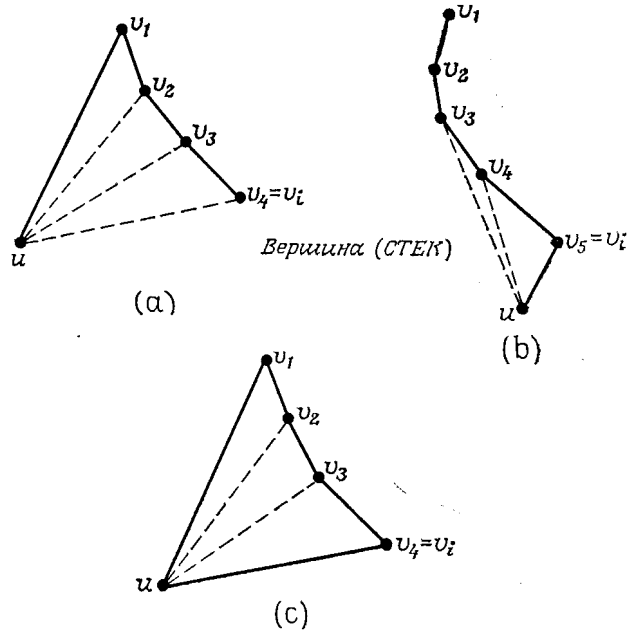



Рис. 6.9. Три случая, возникающие на основном шаге. Штриховыми линиями обозначены добавленные диагонали.

Вершины $v_1$ (нижний элемент стека), $v_2, \ldots, v_i$, находящиеся в стеке СТЕК, образует цепь, являющуюся границей многоугольника $P$, и при этом $y(v_1) > y(v_2) > \ldots > y(v_i)$. Если $i \geqslant 3$, то угол $(v_jv_{j+1}v_{j+2}) \geqslant 180°$ для $j = 1, \ldots, i-2$ (рис. 6.9).

Алгоритм работает следующим образом:

**Начальный шаг.** В стек помещаются вершины $u_1$ и $u_2$.

**Основной шаг.** Пусть $u$ — текущая вершина.

(1) Если $u$ смежна с $v_1$, но не смежна с $v_i$ (верх(СТЕК)), то добавить диагонали $uv_2$, $uv_3$, ..., $uv_i$. Заменить содержимое стека на $v_iu$ (рис. 6.9(а)).

(2) Если $u$ смежна с $v_i$, но не смежна с $v_1$, то, пока $i > 1$ и угол $(uv_iv_{i-1}) < 180°$, добавлять диагональ $uv_{i-1}$ и выталкивать $v_i$ из стека. Если это условие, состоящее из двух частей, больше не выполняется (или если оно не выполнялось с самого начала), то добавить $u$ в стек (рис. 6.9(b)).

(3) Иначе, когда $u$ смежна как с $v_1$, так и с $v_i$, добавить диагонали $uv_2$, $uv_3$, ..., $uv_{i-1}$. (В этом случае обработка заканчивается (рис. 6.9(c)).)

Корректность алгоритма зависит от того, лежат ли добавляемые диагонали целиком внутри многоугольника. Рассмотрим, например, диагональ $(uv_2)$, добавляемую в случае (1). Ни одна из вершин $v_3$, ..., $v_i$ не может находиться внутри или на границе треугольника $(u, v_1, v_2)$, так как внутренние углы в вершинах $v_2$, $v_3$, ..., $v_{i-1}$ равны не менее чем 180° и это приводит к тому, что $u$ и $v_3$, $v_4$, ..., $v_i$ находятся по разные стороны от прямой, содержащей отрезок $v_1v_2$. Никакая другая вершина многоугольника не может находиться внутри или на границе этого треугольника, так как все такие вершины имеют $y$-координату, меньшую чем вершина $u$. Ни одна точка внутри треугольника не может быть внешней для многоугольника, так как в этом случае ломаная прошла бы через внутренность треугольника и по крайней мере одна из его вершин оказалась внутри треугольника. Таким образом, диагональ $uv_2$ целиком лежит внутри многоугольника. Доказательства для случаев (2) и (3) проводятся аналогично, и, кроме того, довольно просто можно проверить, что свойства стека являются инвариантом алгоритма.

Проанализируем работу алгоритма. Слияние цепей, производимое вначале, может быть выполнено за время $O(N)$. В процессе триангуляции каждая вершина заносится в стек и просматривается в точности один раз, за исключением, когда в случае (2) не выполняется условие «пока $i > 1$ и угол $(uv_iv_{i-1}) < 180°$». Если отнести соответствующую работу, так же как и затраты, связанные с доступом к верхнему и нижнему элементам стека, на счет обработки текущей вершины $u$, то очевидно, что эта процедура может быть выполнена также за время $O(N)$. В результате получаем следующую теорему:

**Теорема 6.5.** *Триангуляция монотонного многоугольника с $N$ сторонами может быть выполнена за время $O(N)$, что является оптимальным.*



Объединяя этот результат с полученным ранее, согласно которому прямоугольник rect($S$) можно разбить на монотонные многоугольники за время $O(N \log N)$, и припомнив нижнюю оценку для задачи триангуляции, получаем следующую теорему:

**Теорема 6.6.** *Триангуляцию с ограничением на множестве из $N$ точек можно выполнить за время $\theta(N \log N)$, что является оптимальным.*

Отметим, однако, что теорема 6.6 не применима в случае триангуляции простого многоугольника, так как в этом случае исходно заданные ребра триангуляции уже не являются произвольными. Мы еще вернемся к этому вопросу в разд. 6.5.

## 6.3. Обобщения диаграммы Вороного

Напомним определение диаграммы Вороного, данное в предыдущей главе: Диаграмма Вороного конечного *множества $S$ точек на плоскости* представляет такое разбиение плоскости, при котором каждая область этого разбиения образует множество точек, более близких к *одному из элементов* множества $S$, чем к любому другому элементу этого множества. В этом определении курсивом выделены три элемента (множество точек, плоскость, один элемент), которые могут служить основой для обобщения. И такие обобщения были сделаны, правда, с различным успехом в каждом из этих трех направлений.

Прежде всего, оставаясь, как и прежде, в плоскости (т. е. в двумерном пространстве), можно расширить заданное множество, добавив помимо точек другие геометрические объекты, такие как отрезки, окружности и т. п. Однако в этой книге это направление подробно рассматриваться не будет (см. разд. 6.5).

Далее в поисках структуры данных, обеспечивающей эффективное решение задачи поиска $k$-ближайших соседей (см. задачу Б.6 в разд. 5.1), может возникнуть желание определить области, содержащие точки, более близкие к некоторому заданному подмножеству из $k$ элементов множества $S$, чем к любому другому подмножеству с таким же числом элементов. Любопытно, что если положить $k$ равным $N - 1$, то получится диаграмма Вороного дальней точки.

И наконец, хотя определение обычной диаграммы Вороного на случай пространства произвольной размерности очевидно, ее построение связано со значительными алгоритмическими проблемами. Действительно, можно легко показать, что при заданном числе точек $N$ количество элементов, необходимых для описания диаграммы Вороного, растет экспоненциально в за-

висимости от размерности пространства. Это — явление, с которым мы уже сталкивались, решая задачу о выпуклой оболочке. Впрочем, связь между диаграммами Вороного и выпуклыми оболочками далеко не исчерпывается фактом экспоненциального роста соответствующих им описаний.

### 6.3.1. Диаграмма Вороного высших порядков на плоскости

Диаграмма Вороного, являясь очень мощным средством, не дает ничего, когда речь идет о наиболее удаленных точках, $k$ ближайших точках и других метрических отношениях.

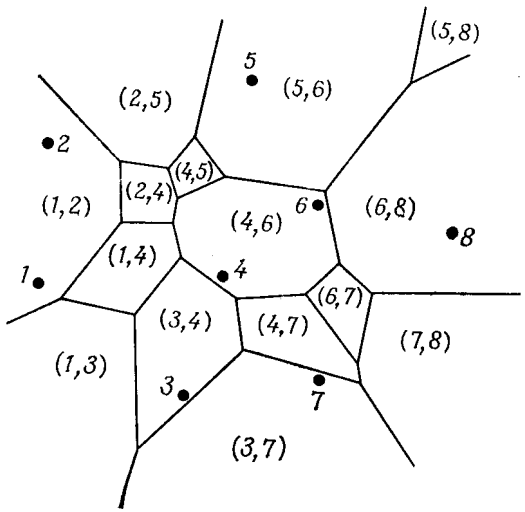



Рис. 6.10. Диаграмма Вороного второго порядка. Точки помечены своими номерами. Некоторые многоугольники Вороного в диаграмме второго порядка могут быть пустыми. Например, отсутствует многоугольник (5, 7).

Как таковая, диаграмма Вороного не может быть применена для решения поставленных задач. Затруднение вызвано тем, что мы рассматривали многоугольник Вороного, связанный с одной точкой. Но такое ограничение вовсе не является необходимым, и будет полезно поговорить об *обобщенном многоугольнике Вороного* $V(T)$ подмножества точек $T$ [Shamos, Hoey (1975)], определяемом так:

$$V(T) = \{p:\ \forall v \in T \forall w \in S - T,\ d(p, v) < d(p, w)\}.$$

То есть $V(T)$ — это множество точек на плоскости, таких что каждая точка из $T$ ближе к некоторой точке $p \in V(T)$, чем



любая точка, не принадлежащая $T$. Дадим эквивалентное определение:

$$V(T) = \bigcap H(p_i, p_j),\quad p_i \in T,\quad p_j \in S - T,$$

где $H(p_i, p_j)$, как обычно, представляет полуплоскость, содержащую $p_i$ и определяемую прямой, перпендикулярной отрезку $\overline{p_i p_j}$ и делящей его пополам. Это определение показывает, что обобщенный многоугольник Вороного по-прежнему является выпуклым. Конечно, может оказаться, что $V(T)$ пусто. В примере на рис. 6.10 отсутствуют точки, для которых их двумя ближайшими соседями являются точки $p_5$ и $p_7$. Множество $S$, содержащее $N$ точек, имеет $2^N$ подмножеств. Для скольких из них многоугольники Вороного непустые? Если соответствующее число невелико, то есть надежда организовать поиск $k$ ближайших соседей, не используя для этого чрезмерный объем памяти.

Определим *диаграмму Вороного порядка* $k$, обозначив ее $\mathrm{Vor}_k(S)$ — как совокупность всех обобщенных многоугольников Вороного $k$-элементных подмножеств множества $S$. Таким образом [1]),

$$\mathrm{Vor}_k(S) = \bigcup V(T),\quad T \subset S,\quad |T| = k.$$

При такой записи обычная диаграмма Вороного это просто $\mathrm{Vor}_1(S)$. Название «диаграмма» для $\mathrm{Vor}_k(S)$ вполне уместно, так как ее многоугольники разбивают плоскость на части. Если задана диаграмма $\mathrm{Vor}_k(S)$, то $k$ ближайших соседей некоторой новой точки $q$ можно найти, определив многоугольник, в котором находится $q$. На рис. 6.10 показана диаграмма Вороного второго порядка, содержащая области, определяющие пары ближайших точек.

Прежде чем приступать к обсуждению вычислительных задач, связанных с построением обобщенных диаграмм Вороного, необходимо изучить их структуру. Этому вопросу посвящен следующий раздел.

#### 6.3.1.1. Элементы инверсной геометрии

Начнем с небольшого отступления в область инверсной геометрии — средства, которое особенно подходит для наших целей.

---

[1]) Правильнее записать так:

$$\mathrm{Vor}_k(S) = \bigcup \{V(T)\},\ T \subset S,\ |T| = k,$$

— *Прим. перев.*

**Определение 6.2.** *Инверсия* в пространстве $E^d$ — это преобразование, переводящее точку, определяемую вектором $\mathbf{v}$, в точку, определяемую вектором $\mathbf{v}' = \mathbf{v} \cdot 1/|\mathbf{v}|^2$ [1]).

Заметим, что инверсия является инволюцией, так как скалярное произведение $\mathbf{v}$ и ее образа $\mathbf{v}'$ равно единице:

$$\mathbf{v} \cdot \mathbf{v}'^T = \mathbf{v} \cdot \mathbf{v}^T \frac{1}{|\mathbf{v}|^2} = 1,$$

и, значит, $\mathbf{v}$ и $\mathbf{v}'$ являются образами друг друга при преобразовании инверсии. Рассмотрим одно характерное свойство инверсии, выражаемое следующей теоремой:

**Теорема 6.7.** *Инверсия в пространстве $E^d$ отображает гиперсферы в гиперсферы.*

*Доказательство.* Гиперсфера $\sigma$ с центром в $\mathbf{c}$ и радиусом $r$ — это множество точек $\mathbf{v}$, удовлетворяющих уравнению $|\mathbf{v} - \mathbf{c}|^2 = r^2$. Отсюда сразу получаем $2\mathbf{v} \cdot \mathbf{c}^T = |\mathbf{v}|^2 + |\mathbf{c}|^2 - r^2$. Теперь рассмотрим величину $|\mathbf{v}' - \mathbf{c}/(|\mathbf{c}|^2 - r^2)|^2$, где $\mathbf{v}$ — образ точки $\mathbf{v}$ при инверсии, т. е. $\mathbf{v}' = 1/|\mathbf{v}|^2 \cdot \mathbf{v}$. Получаем

$$\left|\mathbf{v}' - \frac{\mathbf{c}}{|\mathbf{c}|^2 - r^2}\right|^2 = \frac{1}{|\mathbf{v}|^2} + \frac{|\mathbf{c}|^2}{(|\mathbf{c}|^2 - r^2)^2} - \frac{2\mathbf{v} \cdot \mathbf{c}^T}{|\mathbf{v}|^2 \cdot (|\mathbf{c}|^2 - r^2)}$$

$$= \frac{(|\mathbf{c}|^2 - r^2)^2 + |\mathbf{v}|^2 |\mathbf{c}|^2 - |\mathbf{v}|^2 (|\mathbf{c}|^2 - r^2) - (|\mathbf{c}|^2 - r^2)^2}{|\mathbf{v}|^2 \cdot (|\mathbf{c}|^2 - r^2)^2}$$

$$= \frac{r^2}{(|\mathbf{c}|^2 - r^2)^2}.$$

Так как эта величина является константой (она зависит от констант $|\mathbf{c}|^2$ и $r$), то отсюда следует, что образом гиперсферы $\sigma$ при преобразовании инверсии является гиперсфера с центром в $\mathbf{c}/(|\mathbf{c}|^2 - r^2)$ и радиусом $r/(|\mathbf{c}|^2 - r^2)$.

Преобразование инверсии имеет особенность в начале координат пространства $E^d$. Образом начала координат является бесконечно удаленная гиперплоскость. Следствием этого является тот факт, что при преобразовании инверсии внутренность гиперсферы $\sigma$ отображается на внутренность ее образа $\sigma'$ тогда и только тогда, когда $\sigma$ (а значит, и $\sigma'$) не содержит внутри себя начало координат. В противном случае внутренность $\sigma$ отображается на внешнюю область $\sigma'$.

Для нас, в частности, представляет интерес случай, когда гиперсферы проходят через начало координат, являющийся промежуточным между двумя случаями, упомянутыми выше. Гиперсфера, проходящая через начало координат, отображается в гиперплоскость, а внутренность гиперсферы отображается на полупространство (определяемое гиперплоскостью-образом), *не содержащее* начало координат. Иначе говоря, семейство гиперплоскостей в $E^d$ отображается в семейство гиперсфер, проходящих через начало координат.

[1]) Между точками и векторами в $E^d$ имеется взаимно однозначное соответствие. Поэтому далее вместо «точка, определяемая вектором **V**» будем использовать краткую форму «точка **V**». — *Прим. перев.*



Для большей наглядности последующее изложение основывается на рассмотрении двумерного и трехмерного случаев. Кроме того, при обсуждении диаграмм Вороного высших порядков на плоскости мы также воспользуемся трехмерным пространством; однако для простоты на рисунках будет изображаться двумерная ситуация. Таким образом, из предыдущего

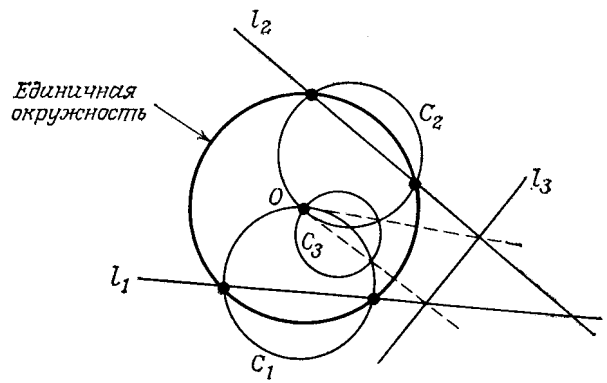



Рис. 6.11. Пример преобразования инверсии на плоскости для случая прямых и окружностей, проходящих через начало координат.

обсуждения следует, что при инверсии в случае плоскости окружности переходят в окружности, а в трехмерном пространстве сферы отображаются в сферы.

В пространстве $E^3$ особую роль играет единичная сфера (сфера единичного радиуса с центром в начале координат), так как точки этой сферы остаются неподвижными при инверсии. На рис. 6.11 показан пример преобразования инверсии на плоскости. На этом рисунке прямая $l_i$ $(i = 1, 2, 3)$ и окружность $C_i$ при инверсии преобразуются друг в друга.

### 6.3.1.2. Структура диаграмм Вороного высших порядков

Теперь мы готовы к обсуждению основного вопроса. Пусть $S$ — множество из $N$ точек на плоскости. Поместим эту плоскость в пространство $E^3$ с координатами $x$, $y$ и $z$, совместив ее с плоскостью $z = 1$ (заметим, однако, что можно было выбрать любую другую плоскость). Построим образ этой плоско-

сти при инверсии, т. е. сферу $C$ радиусом 1/2 с центром в точке $(0, 0, 1/2)$. (См. рис. 6.12, отражающий аналогичную ситуацию в пространстве меньшей размерности.) Отобразим каждую точку множества $S$ в точку на $C$ путем проектирования вдоль прямой, проходящей через начало координат в $E^3$ [1]). В результате этого на сфере $C$ получим множество $S'$ из $N$ точек.

Рассмотрим теперь плоскость $\pi$, определяемую тремя точками $p_i'$, $p_j'$ и $p_l'$ множества $S'$ (эти точки являются образами точек $p_i$, $p_j$ и $p_l$ множества $S$ при преобразовании инверсии).

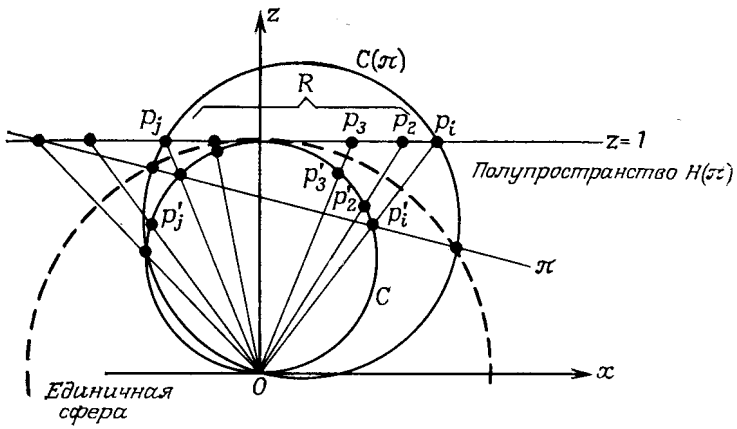



Рис. 6.12. Преобразование инверсии отображает множество точек, лежащих в плоскости $z = 1$, на сферу $C$.

(См. рис. 6.12.) Эта плоскость определяет на сфере $C$ сегмент, не содержащий начало координат. Обозначим через $S'(\pi) \subseteq S'$ подмножество точек множества $S'$, попавших внутрь этого сегмента сферы. Заметим, что $S'(\pi)$ содержит точки, лежащие внутри полупространства $H(\pi)$, определяемого плоскостью $\pi$ и не содержащего начала координат. Рассмотрим образ плоскости $\pi$ при инверсии, обозначив его $C(\pi)$. Как мы уже знаем, $C(\pi)$ — это сфера, проходящая через начало координат и точки $p_i$, $p_j$ и $p_l$ (сфера однозначно определяется этими четырьмя точками). Так как внутренние точки полупространства $H(\pi)$ отображаются внутрь сферы $C(\pi)$, то образы точек множества $S'(\pi)$ при инверсии попадают внутрь $C(\pi)$. В частности, они попадают внутрь окружности $C(p_i, p_j, p_l)$, проходящей через точки $p_i$, $p_j$ и $p_l$, так как эта окружность является пересечением сферы $C(\pi)$ с плоскостью $z = 1$. Обозначим буквой $v$ центр окружности $C(p_i, p_j, p_l)$, а $R \subset S$ — подмножество точек множества $S$, лежащих внутри $C(p_i, p_j, p_l)$ (очевидно, что $R$ является образом $S'(\pi)$). Утверждается, что $v$ является вершиной в некоторой диаграмме Вороного высшего порядка. Действительно, положим $|R| = k$ и предположим на время, что $0 < k < N - 3$. Очевидно, что $R$ содержит точки, находящиеся от $v$ на меньшем расстоянии, чем любая из точек $\{p_i, p_j, p_l\}$. Точка $v$ будет вершиной диаграммы Вороного порядка $(k+1)$, являясь общей точкой многоугольников $V(R \cup \{p_i\})$, $V(R \cup \{p_j\})$ и $V(R \cup \{p_l\})$, и вершиной диаграммы порядка $(k+2)$, являясь общей точкой многоугольников $V(R \cup \{p_i, p_j\})$, $V(R \cup \{p_i, p_l\})$ и $V(R \cup \{p_i, p_l\})$. Если $R = \varnothing$ и, значит, $k = 0$, то $v$ является вершиной лишь $\mathrm{Vor}_1(S)$, а если $|R| = N - 3$, то $v$ является вершиной лишь $\mathrm{Vor}_{N-1}(S)$. Теперь обратим внимание на то, что плоскость $\pi$ может быть выбрана $\binom{N}{3}$ способами и каждая такая плоскость определяет вершину диаграммы Вороного (большинство из них являются вершинами двух диаграмм Вороного высших порядков). Так как каждая диаграмма Вороного является планарным графом и каждая вершина диаграммы имеет степень не менее трех (на самом деле в точности три, за исключением вырожденных вершин), то в каждой диаграмме $\mathrm{Vor}_k(S)$, $k = 1, \ldots, N-1$, число многоугольников пропорционально числу вершин. Объединяя эти факты, получаем следующий интересный результат [Brown (1979a), (1979b)].

[1]) Такая проекция называется *стереографической*.



**Теорема 6.8.** *Число многоугольников Вороного в диаграммах всех порядков равно* $O(N^3)$.

Чтобы добиться более глубокого понимания структуры диаграмм Вороного, воспользуемся подходом, аналогичным подходу, недавно предложенному Эдельсбруннером и Зейделем [Edelsbrunner, Seidel (1986)]. Рассмотрим стереографическую проекцию, представляющую сужение инверсии в пространстве $E^3$ на плоскость $z = 1$ (или, что эквивалентно, на сферу $C$, являющуюся образом этой плоскости). Выпишем явно уравнение сферы $C$:

$$x^2 + y^2 + \left(z - \frac{1}{2}\right)^2 = \frac{1}{4}. \tag{6.1}$$

Предположим теперь, что мы применяем к $E^3$ некоторое подходящее проективное преобразование $\varphi$. Обозначим через $\xi$, $\eta$ и $\zeta$ координаты в преобразованном пространстве. В частности, нас интересует преобразование, отображающее плоскость $z = 0$ в бесконечно удаленную плоскость. Мы осуществим это преобразование следующим образом: сначала введем в $E^3$ однородные координаты $x_1$, $x_2$, $x_3$ и $x_4$ (см. разд. 1.3.2, где обсуждается связь между однородными и неоднородными координатами), за-

тем рассмотрим их как координаты в $E^4$ и, наконец, применим к $E^4$ преобразование поворота, выражаемое следующим соотношением (здесь $\xi_1$, $\xi_2$, $\xi_3$ и $\xi_4$ однородные координаты в преобразованном пространстве):

$$(\xi_1, \xi_2, \xi_3, \xi_4)^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot (x_1, x_2, x_3, x_4)^T. \tag{6.2}$$

Переписав уравнение (6.1) в однородных координатах и применив преобразование (6.2), получим уравнение для $\varphi(C)$ в однородных координатах:

$$\xi_1^2 + \xi_2^2 + \xi_4^2 = \xi_3\xi_4. \tag{6.3}$$

Переходя вновь к неоднородным координатам $\xi = \xi_1/\xi_4$, $\eta = \xi_2/\xi_4$ и $\zeta = \xi_3/\xi_4$, получаем уравнение для $\varphi(C)$ в неоднородных координатах:

$$\zeta = \xi^2 + \eta^2 + 1, \tag{6.4}$$

которое описывает параболоид вращения $\mathscr{P}$.

В дополнение к тому, что нам требовалось, преобразование $\varphi$ отображает плоскость $z = 1$ в плоскость $\zeta = 1$, единичную

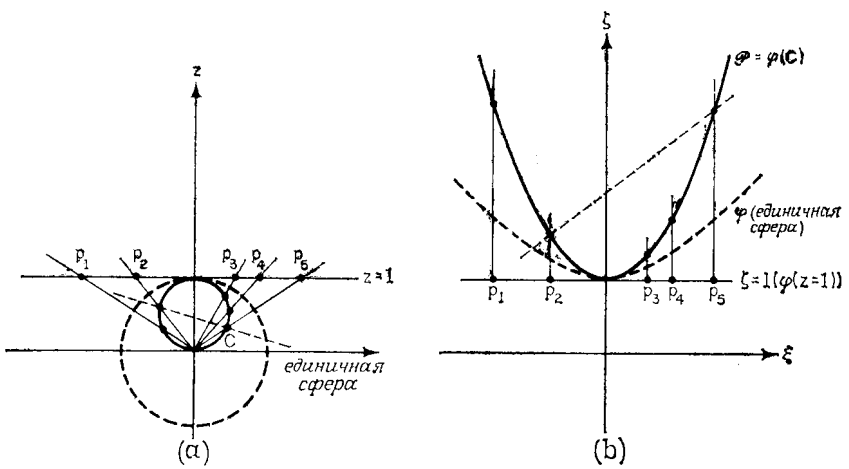



Рис. 6.13. Преобразование $\varphi$ отображает диаграмму (a) в диаграмму (b).

сферу в гиперболоид с уравнением $\xi^2 + \eta^2 - \zeta^2 + 1 = 0$ и точку начала координат в бесконечно удаленную точку по $\zeta$-оси. В итоге диаграмма, приведенная на рис. 6.12 и повторенная на рис. 6.13(a), отображается при применении преобразования $\varphi$ в диаграмму, представленную на рис. 6.13(b). Рассмотрим теперь образы при преобразовании $\varphi$ пар точек, лежащих соответственно на $C$ и на плоскости $z = 1$, являющихся образами друг друга при преобразовании инверсии. Эти новые точки, лежащие соответственно на $\mathscr{P}$ и на плоскости $\zeta = 1$, теперь получаются простым проецированием параллельно $\zeta$-оси (по вертикали). Обозначим эту вертикальную проекцию $\gamma$: $(\zeta = 1) \to \mathscr{P}$.



Пусть снова $S$ — это конечное множество точек, лежащих в плоскости $\zeta = 1$, а $S' = \{\gamma(p): p \in S\}$. Пусть $p_1$ и $p_2$ — две произвольные точки из множества $S$. Рассмотрим плоскости $\pi(p_1)$ и $\pi(p_2)$, касательные к $\mathscr{P}$ в точках $\gamma(p_1)$ и $\gamma(p_2)$ соответственно. Следующая лемма указывает на одно очень важное свойство этих двух плоскостей. Доказательство леммы оставляем в качестве упражнения читателю.

**Лемма 6.4.** *Проекция линии пересечения плоскостей $\pi(p_1)$ и $\pi(p_2)$ на плоскость $\zeta = 1$ перпендикулярна отрезку $p_1p_2$ и делит его пополам.*

Предположим теперь, что уже построено множество плоскостей $\{\pi(p): p \in S\}$. Эти плоскости разбивают пространство на ячейки $\{D_1, D_2, \ldots, D_M\}$. Установим, какая связь имеется между этими ячейками и многоугольниками Вороного (некоторой диаграммы $\mathrm{Vor}_k(S)$) на плоскости $\zeta = 1$. Для каждой точки $p \in S$ определим полупространство $\mathrm{HS}(p)$, задаваемое плоскостью $\pi(p)$ и лежащее вне параболоида $\mathscr{P}$ (здесь снова $\pi(p)$ — это плоскость, касательная к $\mathscr{P}$ в точке $\gamma(p)$). Рассмотрим некоторую ячейку $D$ разбиения пространства, задаваемого множеством плоскостей $\{\pi(p): p \in S\}$. С ячейкой $D$ связано (единственное) множество $T(D) \subseteq S$, определяемое как единственное подмножество множества $S$ такое, что

$$D \subseteq HS(p) \text{ для каждой точки } p \in T(D).$$

(См. рис. 6.14, где приведен двумерный аналог.) Пусть $q'$ — некоторая типичная точка в $D$, а $q$ — ее вертикальная проекция на плоскость $\zeta = 1$. Кроме того, пусть $p_i \in T(D)$ и $p_j \in S - T(D)$ (рис. 6.14(b)). По определению множества $T(D)$, $q' \in HS(p_i)$ и $q' \notin HS(p_j)$. Вертикальная плоскость $\tau$, проходящая по линии пересечения плоскостей $\pi(p_i)$ и $\pi(p_j)$, определяет два полупространства. Из условия $q' \in \mathrm{HS}(p_i)$ и $q' \notin \mathrm{HS}(p_j)$ следует, что $q'$ лежит в полупространстве, содержащем точку $p_i$. С учетом леммы 6.4 получаем, что $q$ принадлежит полуплоскости $H(p_i, p_j)$, и, так как точки $p_i \in T(D)$ и $p_j \in S - T(D)$ выбирались произвольно, приходим к следующему результату:

**Теорема 6.9.** *Проекция на плоскость $\zeta = 1$ ячейки $D$ разбиения пространства, определяемого множеством плоскостей*

*$\{\pi(p)\colon p\in S\}$, является многоугольником Вороного $V(T)$, где $T$ — наибольшее подмножество такое, что $D\subseteq \mathrm{HS}(p)$ для каждой точки $p\in T$.*

Эта теорема показывает, что определенное выше разбиение пространства дает полную информацию о семействе диаграмм

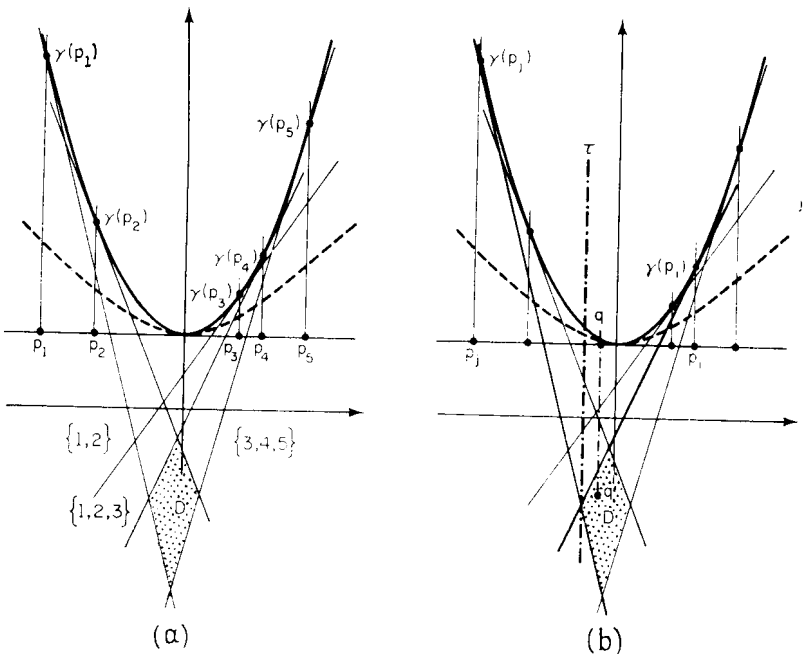



Рис. 6.14. (a) — Двумерный аналог разбиения пространства, определяемого касательными плоскостями; (b) — иллюстрация к доказательству теоремы 6.9.

Вороного высших порядков. Вопрос об алгоритмах построения диаграмм Вороного будет рассмотрен в следующем разделе.

### 6.3.1.3. Построение диаграмм Вороного высших порядков

Для разработки необходимого алгоритма исследуем взаимосвязь между $\mathrm{Vor}_k(S)$ и $\mathrm{Vor}_{k+1}(S)$. (Уместно отметить, что $\mathrm{Vor}_{N-1}(S)$ называется также диаграммой Вороного дальней точки, что вполне оправданно, так как каждый ее многоугольник представляет множество точек плоскости, более близких к одной из точек множества $S-\{p_i\}$ (для некоторого $i$), чем к точке $p_i$, т. е. множество точек, для которых $p_i$ является самой дальней точкой множества $S$.)



Начнем обсуждение со случая, когда множество $S$ содержит три точки $p_1$, $p_2$ и $p_3$ (рис. 6.15). В этом случае имеется единственная вершина диаграммы Вороного, являющаяся общей точкой трех полупрямых (лучей), перпендикулярных отрезкам, соединяющим пары точек, и делящих их пополам. Обычная диаграмма Вороного показана на рис. 6.15 сплошными линиями. Если продолжить эти полупрямые за вершину диаграммы (на рисунке это показано штриховыми линиями), то получившиеся три луча разбивают плоскость на три области так, что

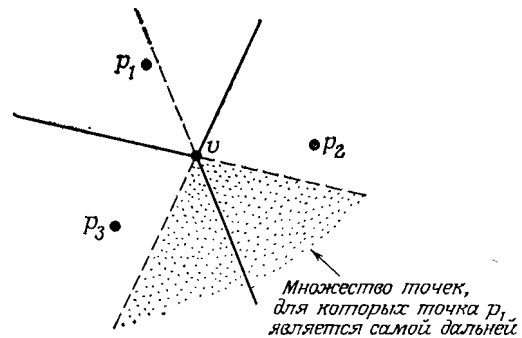



Рис. 6.15. Диаграммы Вороного ближайшей и дальней точек для множества из трех точек.

каждая точка в любой из этих областей находится ближе к одной из двух заданных точек, чем к третьей точке. Таким образом, мы получили диаграмму Вороного дальней точки множества $S=\{p_1, p_2, p_3\}$.

Рассмотрим теперь многоугольник $V(T)$ диаграммы $\mathrm{Vor}_k(S)$, где $T=\{p_1, \ldots, p_k\}$. Известно, что $V(T)$ является выпуклым многоугольником. Пусть $v$ — одна из его вершин. Вершина $v$ является общей для трех многоугольников диаграммы $\mathrm{Vor}_k(S)$: $V(T)$, $V(T_1)$ и $V(T_2)$. Различают два случая[1]):

(1) $|T\oplus T_1\oplus T_2|=k+2 \quad (k\neq N-1)$,

(2) $|T\oplus T_1\oplus T_2|=k-2 \quad (k\neq 1)$.

В случае (1) вершина $v$ называется вершиной *ближнего типа*, а в случае (2) — вершиной *дальнего типа*. Чтобы извлечь некоторую пользу из введенной классификации, отметим, например, что в случае (1) имеем

(a) $T=R_1\cup\{p_k\}$, $T_1=R_1\cup\{p_{k+1}\}$, $T_2=R_1\cup\{p_{k+2}\}$, где $R_1=\{p_1, \ldots, p_{k-1}\}$. Так что, если удалить $R_1$ из $S$, то вершина $v$

[1]) Символ $\oplus$ обозначает симметрическую разность множеств.

и инцидентные ей ребра (которые превратились теперь в бесконечные лучи) образуют диаграмму Вороного ближней точки множества $\{p_k, p_{k+1}, p_{k+2}\}$. В противоположность этому в случае (2) обычно имеет место

(b) $T = R_2 \cup \{p_{k-1}, p_k\}$, $T_1 = R_2 \cup \{p_{k-1}, p_{k+1}\}$,
$T_2 = R_2 \cup \{p_k, p_{k+1}\}$,

где $R_2 = \{p_1, \ldots, p_{k-2}\}$. Если снова удалить $R_2$ из $S$, то вершина $v$ и инцидентные ей ребра образуют диаграмму Вороного *дальней* точки множества $\{p_{k-1}, p_k, p_{k+1}\}$ [1]). Интересен следующий факт: окружность Делоне с центром в точке $v$ содержит внутри $k-1$ точек множества $S$, если $v$ вершина ближнего типа, и $k-2$ точек в противном случае.

Кроме того, предположим, что $v$ — вершина ближнего типа диаграммы $\mathrm{Vor}_k(S)$. Пусть $v$ является общей вершиной многоугольников $V(T)$, $V(T_1)$ и $V(T_2)$, как и в случае (a) выше. Если в некоторой подходящей окрестности вершины $v$ заменить каждое из трех инцидентных ей ребер на его продолжение за вершину $v$, то вершина $v$ станет общей вершиной трех новых многоугольников, которыми, как нетрудно видеть, будут $V(T \cup T_1)$, $V(T \cup T_2)$ и $V(T_1 \cup T_2)$. Но $|T \cup T_1| = |T \cup T_2| = |T_1 \cup T_2| = k+1$. Таким образом, в результате замены ребер, инцидентных вершине ближнего типа, на их продолжение в диаграмме $\mathrm{Vor}_k(S)$, эта вершина становится вершиной дальнего типа диаграммы $\mathrm{Vor}_{k+1}(S)$! Короче говоря, в этом заключается основная идея для метода построения последовательности диаграмм $\mathrm{Vor}_2(S), \mathrm{Vor}_3(S), \ldots, \mathrm{Vor}_{N-1}(S)$.

Продолжая анализ механизма алгоритма построения диаграммы Вороного, предположим, что вершина ближнего типа $v$ диаграммы $\mathrm{Vor}_k(S)$ смежна с некоторой вершиной дальнего типа $v_1$ (рис. 6.16). Это значит, что вершина $v$ была порождена в $\mathrm{Vor}_k(S)$ в результате описанной выше операции продолжения ребер, инцидентных вершинам ближнего типа диаграммы $\mathrm{Vor}_{k-1}(S)$ (одной из таких вершин является вершина $v_1$). В свою очередь при построении $\mathrm{Vor}_{k+1}(S)$ будут продолжаться ребра, инцидентные вершине $v$. При этом в ходе этого процесса будет удалено ребро $\overline{v_1 v}$ и, таким образом, вершина $v_1$ исчезает из диаграммы $\mathrm{Vor}_{k+1}(S)$. Образно говоря, это обсуждение иллюстрирует жизненный цикл вершины диаграммы Вороного: вершина появляется в двух диаграммах Вороного, имеющих последовательные порядки, при этом сначала это вершина ближнего типа, а затем дальнего типа.

[1]) Заметим, что $\mathrm{Vor}_1(S)$ содержит лишь вершины ближнего типа, а $\mathrm{Vor}_{N-1}(S)$ — лишь вершины дальнего типа.



Построение диаграммы $\mathrm{Vor}_{k+1}(S)$, по существу, сводится к разбиению каждого многоугольника диаграммы $\mathrm{Vor}_k(S)$ на части многоугольников диаграммы $\mathrm{Vor}_{k+1}(S)$. Это эквивалентно определению пересечения каждого многоугольника $V(T)$ диаграммы $\mathrm{Vor}_k(S)$ с обычной диаграммой Вороного $\mathrm{Vor}_1(S-T)$. Однако для этого вовсе не обязательно вычислять $\mathrm{Vor}_1(S-T)$ полностью, а достаточно найти лишь ту ее часть, которая является внутренней для $V(T)$. В частности, пусть $e$ — ребро многоугольника $V(T)$, инцидентное некоторой вершине ближнего типа $v$. Тогда $e$ принадлежит границе двух многоугольников $V(T)$ и $V(T')$, где $|T \oplus T'| = 2$, и находится внутри $V(T \cup T')$,

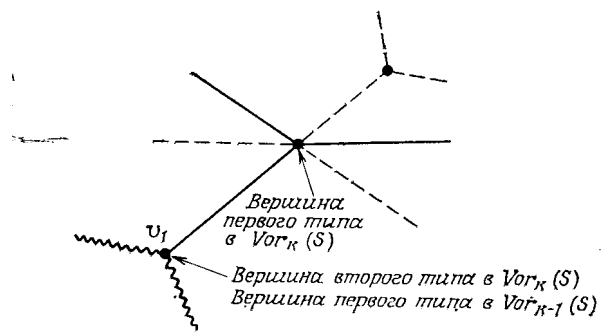



Рис. 6.16. Иллюстрация «жизненного цикла» вершины диаграммы Вороного.

являющегося многоугольником диаграммы $\mathrm{Vor}_{k+1}(S)$. Отсюда следует, что если число вершин ближнего типа на границе многоугольника $V(T)$ равно $s$, то $V(T)$ будет разбит на $s$ частей, если он ограничен, и на $(s+1)$ частей, если он неограничен. Так как вершины ближнего типа определяют подмножество множества $S-T$, влияющее на разбиение многоугольника $V(T)$, то последнее можно получить либо путем построения за время $O(s \log s)$ обычной диаграммы Вороного, либо воспользовавшись специальным методом, разработанным Ли [Lee, (1976, 1982)], который также требует время $O(s \log s)$. В любом случае, если число вершин ближнего типа в диаграмме $\mathrm{Vor}_k(S)$ равно $V_k^{(1)}$, то диаграмма $\mathrm{Vor}_{k+1}(S)$ может быть получена за время, не превосходящее $O(V_k^{(1)} \log V_k^{(1)})$.

Ли [Lee (1982)] провел подробный анализ различных параметров диаграммы $\mathrm{Vor}_k(S)$. В частности, приведем без доказательства следующую лемму:

**Лемма 6.5.** *Число вершин ближнего типа* $V_k^{(1)}$ *диаграммы* $\mathrm{Vor}_k(S)$ *ограничено сверху величиной*

$$2k(N-1) - k(k-1) - \sum_{i=1}^{k} \nu_i,$$

*где* $\nu_i$ *— число неограниченных областей в диаграмме* $\mathrm{Vor}_k(S)$.

Очевидно, что $V_k^{(1)} = O(kN)$. Отсюда следует, что $\mathrm{Vor}_{k+1}(S)$ может быть получена из $\mathrm{Vor}_k(S)$ за время $O(kN \log N)$, а суммарное время построения $\mathrm{Vor}_{k+1}(S)$, начиная с $S$, равно

$$\sum_{i=1}^{k-1} O(iN \log N) = O(k^2 N \log N).$$

Следующая теорема подводит итог сказанному:

**Теорема 6.10.** *Диаграмма Вороного порядка* $k$ *множества из* $N$ *точек может быть построена за время* $O(k^2 N \log N)$.

Такой подход можно было бы применить для построения *всех* диаграмм $\mathrm{Vor}_k(S)$, $k = 1, \ldots, N-1$, на что потребовалось бы время $O(N^3 \log N)$. Однако, используя недавно разработанный подход [Edelsbunner, Seidel (1986)], основанный на теории, в общих чертах представленной в разд. 6.3.1.2, можно построить разбиение пространства, определяемое семейством $\{\pi(p)\colon p \in S\}$ плоскостей, касательных к параболоиду $\mathscr{P}$, затратив на это время, пропорциональное числу пересечений, образуемых тремя плоскостями, т. е. за $O(N^3)$. Таким образом, имеет место теорема 6.11:

**Теорема 6.11.** *Совокупность всех диаграмм Вороного высших порядков множества из* $N$ *точек можно построить за время* $O(N^3)$.

Этот раздел мы завершаем двумя замечаниями. Первое — задача Б.6 о поиске $k$-ближайших соседей сводится к задаче определения положения точки в диаграмме $\mathrm{Vor}_k(S)$. Время решения этой задачи включает время поиска — определение положения в $\mathrm{Vor}_k(S)$ — и время выдачи ответа. Очевидно, что последнее равно $O(k)$, в то время как первое пропорционально $\log V_k$, где $V_k$ — число вершин диаграммы $\mathrm{Vor}_k(S)$. Вспоминая, что $V_k = V_k^{(1)} + V_{k-1}^{(1)}$ и что $V_k^{(1)} = O(kN)$, получаем, что поиск $k$ ближайших соседей выполняется за время $O(\log kN + k) = O(\log N + k)$. Подводя итог сказанному, получаем следующую теорему (см. в разд. 2.2 соответствующий результат для поиска):

**Теорема 6.12.** *Если на плоскости задана некоторая точка, то* $k$ *ее ближайших соседей из числа* $N$ *точек могут быть найдены за время* $O(\log N + k)$ *с использованием предварительной обработки с временем* $O(k^2 N \log N)$.

Второе замечание состоит в том, что понятие обобщенной диаграммы Вороного объединяет задачи о ближайшей и дальней точках, так как множество точек, $k$ ближайших соседей которых составляют множество $T$, является также множеством точек, для которых $N-k$ наиболее удаленных от них точек составляют множество $S-T$. Таким образом, диаграмма ближней точки порядка $k$ в точности совпадает с диаграммой дальней точки порядка $(N-k)$. Давайте рассмотрим одну из них более



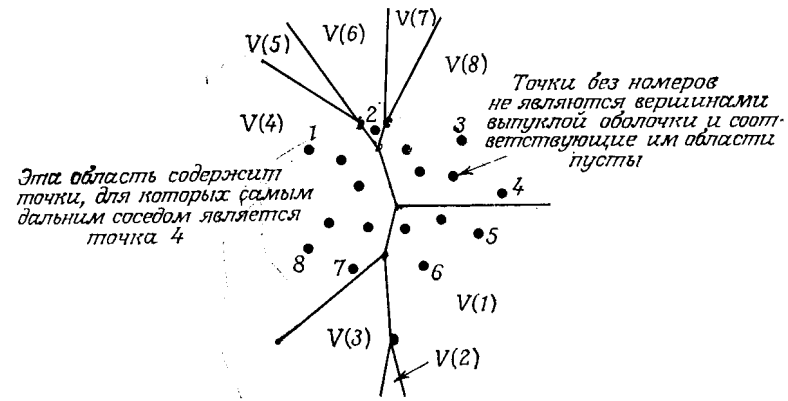



Рис. 6.17. Диаграмма Вороного дальней точки.

подробно, а именно диаграмму ближней точки порядка $(N-1)$ или, что эквивалентно, диаграмму дальней точки порядка 1 (рис. 6.17).

С каждой точкой $p_i$ связана выпуклая многоугольная область $V_{N-1}(p_i)$ такая, что $p_i$ наиболее удалена от каждой точки в этой области. Эта диаграмма определяется только точками выпуклой оболочки, так что в ней отсутствуют ограниченные области. Конечно, $\mathrm{Vor}_{N-1}(S)$ можно построить, применив непосредственно описанный общий метод. Однако в этом случае время решения задачи составило бы $O(N^3)$ (по теореме 6.12). Необходимо отметить, что существует прямой метод, основанный на подходе «разделяй и властвуй», аналогичный алгоритму построения диаграммы ближней точки, позволяющий получить результат за оптимальное время $\theta(N \log N)$ [Shamos (1978); Lee (1980b)]. Если уже найдены диаграммы левой и правой половин множества, то разделяющая ломаная линия $\sigma$ обладает теми же свойствами, что и в случае диаграммы ближней точки. Однако к этому моменту должны быть удалены все части ребер диаграммы $\mathrm{Vor}_{N-1}(S_1)$, лежащие слева от $\sigma$, а также части ребер диаграммы $\mathrm{Vor}_{N-1}(S_2)$, лежащие справа от $\sigma$. В следующем разделе у нас еще будет возможность обсудить очень тесную связь, существующую между $\mathrm{Vor}_1(S)$ и $\mathrm{Vor}_{N-1}(S)$.

### 6.3.2. Диаграммы Вороного ближней и дальней точек в многомерных пространствах

Как мы уже видели ранее, обобщенные диаграммы Вороного (всех порядков) на плоскости имеют топологию планарных графов. Из этого следует, что число вершин, число граней и число ребер диаграммы имеют один и тот же порядок. Это позволяет разрабатывать оптимальные по времени

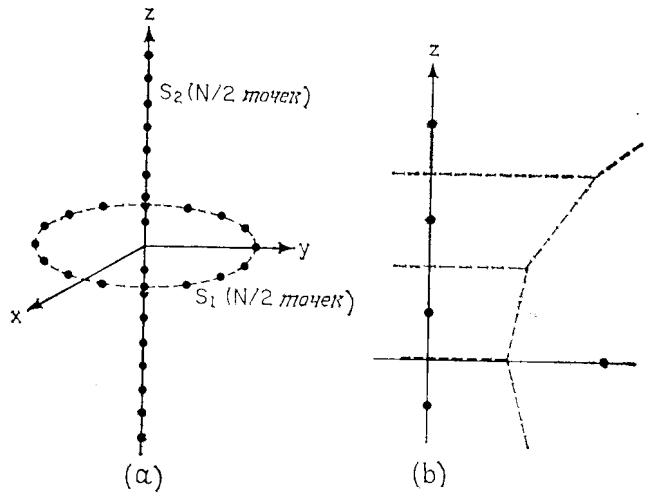



Рис. 6.18. Пример множества из $N$ точек в трехмерном пространстве, диаграмма Вороного которого имеет $\theta(N^2)$ ребер.

алгоритмы построения диаграмм Вороного ближней и дальней точек.

Можно ли получить такой же результат для пространств размерности, большей двух? Все надежды на это тут же рушатся, если учесть, что диаграмма Вороного ближней точки множества из $N$ точек в трехмерном пространстве может иметь $O(N^2)$ ребер [Preparata (1977)]. Легко построить пример, обладающий этим свойством (рис. 6.18). Пусть $S_1$ — множество из $N/2$ точек, равномерно расположенных на окружности, лежащей в плоскости $(x, y)$, центр которой находится в начале координат. Кроме того, пусть $S_2$ — множество из $N/2$ точек, равномерно расположенных на $z$-оси, по $N/4$ точек выше и ниже начала координат. Положим $S = S_1 \cup S_2$. Утверждается, что любой отрезок, соединяющий некоторую точку множества $S_1$ с некоторой точкой множества $S_2$, является ребром графа Делоне, т. е. оно является двойственным некоторому многоугольнику диаграммы Вороного (грани некоторого политопа диаграммы Вороного). Это хорошо видно на сечении диаграммы



Вороного плоскостью, содержащей $z$-ось и некоторую точку множества $S_1$ (рис. 6.18(b)). (Читатель может легко воспроизвести опущенные здесь детали.) Поэтому, хотя диаграмма Вороного содержит $O(N)$ областей, она может иметь $O(N^2)$ вершин и ребер. Позже Кли [Klee (1980)] развил эту идею для пространств произвольной размерности $d$. Полученный результат сформулирован в следующей теореме:

**Теорема 6.13.** *$M(d, N)$ — максимальное число вершин диаграммы Вороного множества из $N$ точек в $d$-мерном пространстве удовлетворяет условию*

$$\lceil d/2 \rceil! \, n^{\lceil d/2 \rceil} \leqslant M(d, N) \leqslant 2(\lceil d/2 \rceil! \, n^{\lceil d/2 \rceil})$$

*для четных $d$ и условию*

$$\frac{(\lceil d/2 \rceil - 1)!}{e} \cdot n^{\lceil d/2 \rceil} \leqslant M(d, N) < \lceil d/2 \rceil! \, n^{\lceil d/2 \rceil}$$

*для нечетных $d$.*

Экспоненциальный рост сложности диаграммы Вороного раскрывает любопытное сходство с поведением выпуклых оболочек в пространствах большой размерности. Это сходство, как впервые заметил Браун [Brown (1979b)], имеет глубокие корни, которые со всей очевидностью проявляются в контексте преобразования инверсии, описанного в разд. 6.3.1.1.

Не теряя общности, рассмотрим диаграммы Вороного на плоскости. Пусть $S = \{p_1, \ldots, p_N\}$ — множество из $N$ точек, лежащих в плоскости $z = 1$ пространства $E^3$, $C$ — образ этой плоскости при преобразовании инверсии в $E^3$, а $S' = \{p'_1, \ldots, p'_N\}$ — образ множества $S$ при инверсии (точки множества $S'$ лежат на $C$, $p_i$ и $p'_i$ являются образами друг друга). Рассмотрим выпуклую оболочку $\mathrm{CH}(S')$ множества $S'$ и разобьем ее на две части, которые будем называть *дальней стороной* и *ближней стороной* в зависимости от того, «видны» или нет их точки из начала координат (на рис. 6.19 показан двумерный аналог ситуации).

Пусть $\pi$ — плоскость, содержащая грань $F$ оболочки $\mathrm{CH}(S')$, определяемую тремя точками: $p'_1$, $p'_2$ и $p'_3$ (предполагается, что многогранник $\mathrm{CH}(S')$ является симплициальным, т. е. каждая его грань является треугольником). Сфера $C(\pi)$, являющаяся образом плоскости $\pi$ при инверсии, проходит через точки $O$, $p_1$, $p_2$ и $p_3$ и пересекает плоскость $z = 1$ по окружности, определяемой точками $p_1$, $p_2$ и $p_3$. В зависимости от того, принадлежит $F$ ближней или дальней стороне оболочки $\mathrm{CH}(S')$, будем различать два случая:

(1) *F принадлежит ближней стороне оболочки* $\mathrm{CH}(S')$. В этом случае полупространство, определяемое плоскостью $\pi$ и не содержащее начала координат, не содержит внутри ни одной точки множества $S'$. Так как это полупространство отображается внутрь сферы $C(\pi)$, то окружность, проходящая через точки $p_1$, $p_2$ и $p_3$, является окружностью Делоне, центр которой является вершиной диаграммы $\mathrm{Vor}_1(S)$. Таким образом, каждая грань ближней стороны оболочки $\mathrm{CH}(S')$ определяет треугольник в

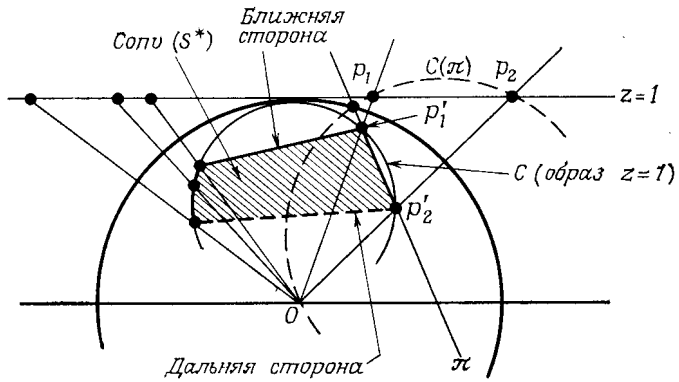



Рис. 6.19. Пример, иллюстрирующий связь выпуклых оболочек с диаграммами Вороного. Ближняя сторона выпуклой оболочки $\mathrm{conv}(S^*)$ показана сплошным отрезком, а дальняя — штриховыми.

триангуляции Делоне множества $S$ (и с учетом двойственности вершину диаграммы $\mathrm{Vor}_1(S)$).

(2) *F принадлежит дальней стороне оболочки* $\mathrm{CH}(S')$. Снова полупространство, определяемое плоскостью $\pi$ и не содержащее начала координат, отображается внутрь сферы $C(\pi)$. Однако в этом случае полупространство содержит внутри $N-3$ точек множества $S''$, так что окружность, проходящая через точки $p_1$, $p_2$ и $p_3$, содержит внутри $N-3$ точек множества $S$. Отсюда следует, что центр этой окружности является вершиной диаграммы Вороного дальней точки множества $S$ [1]).

Это наблюдение не только устанавливает тесную связь между диграммами $\mathrm{Vor}_1(S)$ и $\mathrm{Vor}_{N-1}(S)$, но также связывает диаграмму Вороного ближней точки множества из $N$ точек в $d$-мерном пространстве с выпуклой оболочкой $N$ точек, лежащих на поверхности сферы в $(d+1)$-мерном пространстве. Следовательно, можно воспользоваться существующими методами построения выпуклых оболочек в многомерных пространствах (см. разд. 3.3) для получения многомерных диаграмм Вороного. Соответствующая работа была выполнена Эйвисом и Бхатачарья [Avis, Bhattacharya (1983)] и Зейделем [Seidel (1982)].

[1]) В предположении, что никакие четыре точки множества $S$ не лежат на одной окружности, окружность, являющаяся пересечением сферы $C(\pi)$ и плоскости $z=1$, проходит в точности через три точки множества $S$. Соответственно грань $F$ определяется образами этих трех точек и никакими другими точками множества $S'$, а это значит, что $F$ является треугольником (а $\mathrm{CH}(S)$ — симплициальным политопом).



## 6.4. Промежутки и покрытия

Рассмотрим теперь очень важный класс задач о близости, который мы коротко назвали «промежутки и покрытия». Промежутки — это шары (в случае плоскости это круги), не содержащие точек заданного множества, а *покрытия* — это множества шаров, объединение которых содержит все точки заданного множества. Начнем со следующей задачи:

**Задача Б.11** (НАИМЕНЬШАЯ ОХВАТЫВАЮЩАЯ ОКРУЖНОСТЬ)[1]). На плоскости заданы $N$ точек. Найти наименьшую окружность, охватывающую все заданные точки.

Это классическая задача, которой посвящена обширная литература. Поиск эффективного алгоритма решения этой задачи начался, по-видимому, в 1860 г. [Sylvester (1860)]. Наименьшая охватывающая окружность единственна, и, кроме того, она либо является описанной окружностью для некоторой тройки точек заданного множества, либо некоторая пара точек заданного множества служит диаметром этой окружности [Rademacher, Toeplitz (1957), с. 16]. Таким образом, существует прямой алгоритм решения этой задачи, который перебирает все пары и тройки точек множества, строит определяемые ими окружности и выбирает среди них наименьшую, охватывающую при этом все исходное множество точек. Непосредственная реализация этой процедуры дала бы алгоритм со сложностью $O(N^4)$. Елзинга и Хирн [Elzinga, Hearn (1972a, 1972b)] улучшили этот метод, получив алгоритм со сложностью $O(N^2)$.

В исследовании операций задача об охватывающей окружности известна под названием «минимаксная задача о *размещении центра обслуживания*». В этой задаче требуется найти точку $p_0=(x_0, y_0)$ (центр окружности), для которой наибольшее из расстояний до точек заданного множества минимально. Точка $p_0$ определяется критерием

$$\min_{p_0} \max_{i} \{(x_i - x_0)^2 + (y_i - y_0)^2\}. \tag{6.5}$$

[1]) Часто для этой задачи используется название МИНИМАЛЬНЫЙ ПОКРЫВАЮЩИЙ КРУГ.

Такой минимаксный критерий используется при определении местоположения пунктов экстренной помощи, таких как полицейские участки и станции скорой помощи, с целью минимизировать время оказания помощи в худшем случае [Toregas *et al.* (1971)]. Он также используется при оптимальном размещении радиопередатчика, обслуживающего $N$ принимающих устройств, чтобы минимизировать необходимую мощность радиопередатчика [Nair, Chandrasekaran (1971)]. По-видимому, критерий (6.5) ввел в заблуждение некоторых авторов, рассматривавших задачу о наименьшей охватывающей окружности как задачу непрерывной оптимизации, так как, на их взгляд, в постановке задачи отсутствуют какие-либо ограничения на положение точки $p_0$. Это привело к появлению ряда итеративных алгоритмов решения этой задачи, хотя известно, что она является дискретной [Lawson, Zhukhovitsky, Avdeyeva (1966)]. Этот пример иллюстрирует важный момент: *тот факт, что задачу $\mathscr{A}$ можно сформулировать как частный случай задачи $\mathscr{B}$, не дает основания полагать, что общий метод, используемый для решения задачи $\mathscr{B}$, является эффективным и для решения задачи $\mathscr{A}$.* Мы уже имели возможность убедиться в справедливости этого принципа в связи с задачей о евклидовом минимальном остовном дереве (разд. 6.1): хотя ее можно рассматривать в постановке для полного графа, определенного на заданном множестве точек, усилия, затраченные на поиск метода, использующего особенности задачи, были сполна вознаграждены разработкой оптимального алгоритма.

Теперь обратимся к следующей задаче:

**Задача Б.12** (НАИБОЛЬШАЯ ПУСТАЯ ОКРУЖНОСТЬ). На плоскости заданы $N$ точек. Найти наибольшую окружность, не содержащую внутри ни одной точки этого множества, центр которой лежит внутри выпуклой оболочки множества точек (рис. 6.20).

Ограничение на положение центра окружности необходимо, так как при его отсутствии задача не имела бы ограниченного решения. Эта задача является двойственной к задаче Б.11 в том смысле, что она является *максиминной*. Иначе говоря, требуется найти точку $p_0$, определяемую критерием

$$\max_{p_0 \in \text{Hull}(s)} \min_i \{(x_i - x_0)^2 + (y_i - y_0)^2\}. \tag{6.6}$$

Очевидно, что могут существовать несколько точек $p_0$, удовлетворяющих критерию (6.6). Задача о наибольшей пустой окружности является примером другой задачи о размещении какой-либо службы или предприятия. Но в этом случае необходимо выбрать положение объекта так, чтобы он был максимально удален от других $N$ заданных объектов. Размещаемый



объект может быть источником загрязнения, поэтому его следует располагать таким образом, чтобы минимизировать эффект от такого соседства, либо это может быть новое производство и при этом желательно избежать конкуренции за окружающее пространство. Эти задачи возникают довольно часто в промышленности [Francis, White (1974)]. Один из ранних алгоритмов решения данной задачи давал решение за время $O(N^3)$ в худшем случае [Dasarathy, White (1975)].

Тот факт, что алгоритм, решающий задачу Б.12, имеет сложность $O(N^3)$, в то время когда, как нам кажется, можно получить лучший результат ($O(N^2)$) для «двойственной» задачи, приводит в полное недоумение. Имеется ли между этими двумя задачами глубокое различие? Сейчас мы увидим, что отличие между $O(N^3)$ и $O(N^2)$ отражает недостаточную глубину анализа задач, так как с помощью диаграммы Вороного обе эти задачи могут быть решены за время $O(N \log N)$. Тогда как время $\theta(N \log N)$ является оптимальным для задачи Б.12, для задачи Б.11, как мы увидим, достижима оптимальная оценка получения решения, равная $\theta(N)$.

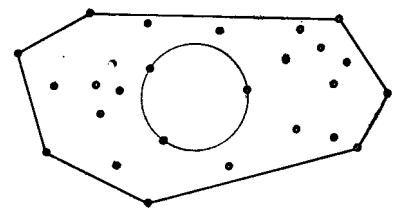


Рис. 6.20. Наибольшая пустая окружность с центром внутри выпуклой оболочки.

Начнем с того, как, используя диаграмму Вороного, можно решить задачу Б.11. Мы знаем, что наименьшая охватывающая окружность $C$ полностью определяется либо диаметром заданного множества $S$, либо тремя его точками. Напомним, что окружность с центром в вершине диаграммы Вороного дальней точки множества $S$ ($\text{Vor}_{N-1}(S)$) и проходящая через три точки множества $S$, определяемые этой вершиной, охватывает все точки множества $S$. Кроме того, окружность с центром в произвольной точке, лежащей на ребре $e$ диаграммы $\text{Vor}_{N-1}(S)$, и проходящая через две точки такие, что ребро $e$ перпендикулярно отрезку, соединяющему эти точки, и делит его пополам, также охватывает все точки множества $S$. Таким образом, если окружность $C$ определяется тремя точками множества $S$, то ее центр находится в некоторой вершине диаграммы $\text{Vor}_{N-1}(S)$. В противном случае, если $C$ определяется двумя точками, то ее центр находится на некотором ребре диаграммы $\text{Vor}_{N-1}(S)$. Возможен следующий подход к решению задачи [Shamos (1978)]. Сначала за время $O(N \log N)$ определяется диаметр множества $S$ (например, методом, рассмотренным в разд. 4.2.3) и проверяется, охватывает или нет окружность, построенная на этом диаметре, заданное множество точек. Если окружность

охватывает множество, то она и является искомой. Иначе центр $c$ окружности $C$ совпадает с одной из вершин диаграммы $\mathrm{Vor}_{N-1}(S)$. Эта диаграмма содержит лишь $O(N)$ точек, а радиусы окружностей, связанных с каждой из ее вершин, равны расстоянию от этой вершины до любой из трех точек, многоугольники которых пересекаются в вершине. Минимальное по всем вершинам расстояние дает радиус окружности $C$. Ясно, что эту процедуру можно выполнить за время $O(N \log N)$, так как и для определения диаметра множества $S$, и для построения диаграммы $\mathrm{Vor}_{N-1}(S)$ требуется время $O(N \log N)$. Кроме того, дополнительно требуется время $O(N)$ для просмотра вершин диаграммы $\mathrm{Vor}_{N-1}(S)$[1]).

В теореме 6.14 резюмируется вышесказанное:

**Теорема 6.14.** *Наименьшую охватывающую окружность множества из $N$ точек можно определить за время $O(N \log N)$.*

Оптимален ли этот результат? Для определения минимальной охватывающей окружности в действительности совсем не требуется полной информации о диаграмме Вороного дальней точки. Достаточно иметь лишь ту ее часть, которая попадает в некоторую окрестность центра окружности. Именно эта идея позволила Меджиддо разработать оптимальный алгоритм с оценкой $\theta(N)$ [Megiddo (1983)]. Основываясь на исходной формулировке (6.5), Меджиддо рассматривал ее как задачу выпуклого программирования. Этот новый изящный метод будет описан в разд. 7.2.5.

Обратимся теперь к задаче, двойственной к только что рассмотренной, т. е. к задаче о наибольшей пустой окружности. Мы покажем, что на плоскости и в пространствах более высокой размерности эта задача может быть решена с помощью диаграммы Вороного. Для простоты и большей конкретности изложения мы будем рассматривать в основном двумерный случай, хотя обобщение на многомерный случай не составляет труда. Имеет место следующая теорема:

**Теорема 6.15.** *Наибольшая пустая окружность для множества из $N$ точек на плоскости может быть построена за время $O(N \log N)$.*

*Доказательство.* Пусть на плоскости задано множество $S$ из $N$ точек. Рассмотрим функцию $f(x, y)$, значение которой равно расстоянию между точкой $p = (x, y)$ и ближайшей к ней точкой множества $S$. Так как условием задачи положение центра наибольшей пустой окружности ограничено выпуклой оболочкой

[1]) Альтернативный подход, дающий ту же самую оценку сложности, обсуждается в работе [Preparata (1977)].



множества $S$, то мы будем рассматривать пересечение диаграммы $\mathrm{Vor}(S)$ с выпуклой оболочкой $\mathrm{conv}(S)$. Это пересечение представляет совокупность выпуклых многоугольников, каждый из которых получается в результате пересечения многоугольника диаграммы Вороного с $\mathrm{conv}(S)$. Внутри многоугольника диаграммы функция $f(x, y)$ является выпуклой вниз как по $x$, так и по $y$, и это справедливо для каждого многоугольника рассматриваемого разбиения выпуклой оболочки. Таким образом, $f(x, y)$ достигает своего максимума в некоторой вершине одного из таких многоугольников[1]). Эта вершина является либо вершиной диаграммы Вороного (и в этом случае наибольшая пустая окружность является окружностью Делоне), либо точкой пересечения ребра диаграммы Вороного с ребром выпуклой оболочки. Все вершины диаграммы Вороного можно найти за время $O(N \log N)$, и, следовательно, остается лишь показать, что можно достаточно быстро найти пересечение выпуклой оболочки и диаграммы Вороного $\mathrm{Vor}(S)$[2]). Но сначала обратим внимание на следующие два факта.

**Свойство 1.** *Ребро диаграммы Вороного пересекает не более двух ребер оболочки* $\mathrm{CH}(S)$. *Действительно, учитывая выпуклость* $\mathrm{conv}(S)$, *пересечение любой прямой с* $\mathrm{conv}(S)$ *состоит из единственного отрезка (возможно, пустого).*

**Свойство 2.** *Каждое ребро оболочки* $\mathrm{CH}(S)$ *пересекает по крайней мере одно ребро диаграммы Вороного. Действительно, каждое ребро оболочки* $\mathrm{CH}(S)$ *соединяет две различные точки множества $S$, принадлежащие двум различным многоугольникам Вороного.*

«Закроем» условно каждый неограниченный многоугольник Вороного отрезком на бесконечно удаленной прямой, лежащей в плоскости (рис. 6.21).

Пусть $e_1, e_2, \ldots, e_n$ — последовательность ребер выпуклой оболочки $\mathrm{CH}(S)$, перечисленных в порядке обхода против часовой стрелки, а $u$ — точка пересечения произвольно выбранного ребра $e_j$ с ребром $r$ диаграммы Вороного (согласно свойству 2, точка $u$ всегда существует). Считая, что ребро $r$ ориентировано в направлении из $u$ во внешнюю для $\mathrm{conv}(S)$ область, обозначим через $V(i)$ многоугольник диаграммы, расположенный слева от $r$. Если двигаться из точки $u$ по границе многоугольника $V(i)$ (вне выпуклой оболочки $\mathrm{conv}(S)$), обходя ее против часовой стрелки, то вновь попадем в некоторую точку $w$ пересечения ребра $\mathrm{CH}(S)$ с ребром диаграммы. Учитывая вы-

[1]) Имеется в виду максимум на множестве $\mathrm{conv}(S)$. — *Прим. перев.*

[2]) Речь идет о пересечении ребер выпуклой оболочки с ребрами диаграммы Вороного. — *Прим. перев.*

пуклость многоугольника $V(i)$ и свойство 2, точка $w$ принадлежит либо ребру $e_j$, либо ребру $e_{j+1}$. Таким образом, каждое ребро многоугольника $V(i)$ проверяется на пересечение с двумя ребрами оболочки $\mathrm{CH}(S)$, что делается за постоянное время. После того как найдена точка $w$, процедура обхода повторяется вновь, только теперь вместо точки $u$ он начинается из $w$, и так до тех пор, пока вновь не будет достигнута точка $u$. Действуя таким образом, можно найти все точки пересечения $\mathrm{Vor}(S)$ с $\mathrm{CH}(S)$.

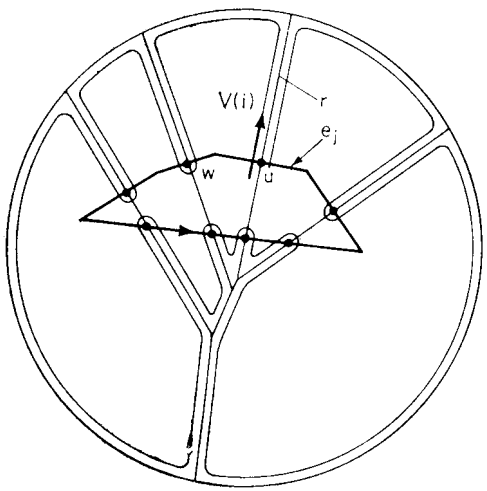



Рис. 6.21. Пример обхода диаграммы $\mathrm{Vor}(S)$ при определении пересечений $\mathrm{Vor}(S)$ с $\mathrm{conv}(S)$.

При таком обходе каждый посещаемый отрезок диаграммы $\mathrm{Vor}(S)$ проходится не более двух раз. Так как, согласно свойству 2, каждое ребро имеет не более двух таких отрезков, то каждое ребро проверяется самое большее четыре раза, и, следовательно, пересечение диаграммы $\mathrm{Vor}(S)$ с оболочкой $\mathrm{CH}(S)$ вычисляется за время $O(N)$[1]).

Здесь может появиться желание узнать, насколько оптимален полученный результат. И, как обычно, для ответа на этот вопрос обратимся к одномерному случаю. В этом случае задача сводится к поиску на прямой пары последовательных точек, максимально удаленных друг от друга (МАКСИМАЛЬНЫЙ ПРОМЕЖУТОК), так как в одномерном случае окружность вырождается в отрезок. Нижняя оценка $\Omega(N \log N)$ времени решения этой задачи, полученная Мэнбером и Томпа [Manber, Tompa (1982)] для модели линейных деревьев решений, недавно была расширена Ли и Ву [Lee, Wu (1986)] на более общую математическую модель вычислений. (Их результат основывается на сведении задачи МАКСИМАЛЬНЫЙ ПРОМЕЖУТОК к задаче ОДНОРОДНЫЙ ПРОМЕЖУТОК, рассматриваемой как задача-прототип.) Это контрастирует со следующим самым неожиданным результатом, полученным Гонзалесом [Gonzales (1975)]: если изменить модель вычислений,

[1]) Описанный подход связан с аналогичным результатом, полученным Туссэном [Toussaint (1983b)].



то задачу можно решить даже за линейное время. Изменение заключается в добавлении к стандартному набору функций функции вычисления целой части «$\lfloor\ \rfloor$» (которая не является аналитической). Здесь мы приводим этот выдающийся алгоритм Гонзалеса:

**procedure** МАКСИМАЛЬНЫЙ-ПРОМЕЖУТОК
Входные данные: $N$ действительных чисел $X[1:N]$ (числа не упорядочены)
Выходные данные: MAXGAP — длина наибольшего промежутка между последовательной парой чисел упорядоченного множества

```
begin MIN := min X[i];
      MAX := max X[i]; [1])
      (* разбить интервал между точками MIN и MAX на
      N — 1 равных отрезков. Массивы HIGH и LOW
      используются для хранения соответственно
      наибольшего и наименьшего значений на отрезках *)
      for i := 1 until N — 1 do
          begin COUNT[i] := 0;
                LOM[i] := HIGH[i] := Λ
          end; (* установлены начальные значения
                параметров отрезков *)
      (* распределение точек по отрезкам *)
      for i := 1 until N — 1 do
          begin BUCKET := 1 + ⌊(N — 1) × (X[i] —
                MIN)/(MAX — MIN)⌋;
                COUNT[BUCKET] := COUNT [BUCKET] + 1;
                LOW[BUCKET] := min(V[i],
                LOW[BUCKET]); [2])
                HIGH[BUCKET] := max (X[i],
                HIGH[BUCKET]); [2])
          end;
      (* Заметим, что N — 2 точек распределены по N — 1
      отрезкам, и, следовательно, один из отрезков должен
      быть пустым. Это значит, что наибольший промежуток
      не может помещаться между двумя точками, попавшими в один и тот же отрезок. Теперь достаточно
      выполнить однократный просмотр отрезков *)
      MAXGAP := 0;
      LEFT := HIGH[1];
```

[1]) MIN и MAX присваиваются значения соответственно минимального и максимального элементов массива $X$. — *Прим. перев.*

[2]) Здесь считается, что $\min(x, \Lambda) = \max(x, \Lambda) = x$.

```
for i := 2 until N − 1 do
    if (COUNT[i] ≠ 0) then
        begin THISGAP := LOW[i] − LEFT;
            MAXGAP := max(THISGAP, MAXGAP);
            LEFT := HIGH[i]
        end
end.
```

Этот алгоритм несколько проливает свет на вычислительную мощность функции выделения целой части. Учитывая кажущееся сходство задач МАКСИМАЛЬНЫЙ ПРОМЕЖУТОК и БЛИЖАЙШАЯ ПАРА в одномерном случае, существование алгоритма с линейной сложностью представляет выдающийся результат. К сожалению, по-видимому, невозможно обобщить этот результат на случай двумерного пространства.

## 6.5. Замечания и комментарии

Хотя евклидово минимальное остовное дерево множества точек на плоскости может быть построено за оптимальное время, построение ЕМОД в пространствах более высокой размерности, вообще говоря, остается открытой задачей. Используя геометрическую природу задачи, Яо [Yao (1982)] разработал алгоритм построения ЕМОД множества из $N$ точек в $d$-мерном пространстве за время $T(N, d) = O(N^{2-a(d)} \cdot (\log N)^{1-a(d)})$, где $a(d) = 2^{-(d+1)}$, имеющий оценку $O(N(\log N)^{1,8})$ для $d = 3$ (этот метод тесно связан с алгоритмом Яо для определения диаметра множества точек (см. разд. 4.3)).

С диаграммой Вороного тесно связана задача построения скелета многоугольника, являющегося предметом довольно большого числа исследований, относящихся к распознаванию образов, так как он довольно хорошо отражает «форму» многоугольника. *Скелет* простого многоугольника $P$ — это множество $\sigma$ его внутренних точек таких, что каждая точка $p \in \sigma$ равноудалена по крайней мере от двух точек на границе $P$. В связи с этим скелет многоугольника также называют *срединными осями* [Duda, Hart (1973)]. Образование скелета многоугольника можно наглядно представить, наблюдая процесс «выгорания» многоугольника. Представьте, что по всей границе многоугольника одновременно загорается огонь. Если предположить, что огонь распространяется с постоянной скоростью, то точки, в которых происходит встреча фронтов распространения огня, образуют скелет многоугольника (этот процесс ассоциируется с пожаром в прерии). В другой интерпретации скелет — это подграф планарной карты, образованной областями близости



сторон многоугольника $P$. Это приводит к расширению понятия диаграммы Вороного на некоторое множество отрезков (являющихся сторонами простого многоугольника).

Более сильное обобщение диаграммы Вороного, о котором уже упоминалось в разд. 6.3, заключается в расширении этого понятия на конечное множество точек и открытых отрезков, расположенных произвольным образом. В этом случае ребра диаграммы Вороного являются не только отрезками прямой, но и дугами параболы, так как они отделяют области близости пар объектов следующих типов: (точка, точка), (линия, линия), (точка, линия). Пары последнего типа и порождают дуги парабол. Первые алгоритмы решения этой задачи, близкие к оптимальным, были предложены Драйсдейлом и Ли [Drysdale (1979); Lee (1978); Lee, Drysdale (1981)] (последний из этих алгоритмов имеет сложность $O(N \log^2 N)$ для смешанного множества из $N$ объектов). Позднее Киркпатрик [Kirkpatrick (1979)] нашел оптимальное решение задачи со сложностью $\theta(N \log N)$. Киркпатрик первым решил важную задачу объединения (слияния) за линейное время диаграмм Вороного двух множеств точек, не разделенных прямой. Затем он использовал этот прием и алгоритм с линейной сложностью для получения минимального остовного дерева планарного графа (см. разд. 6.1) для решения задачи в общей постановке. Заметим, что этот метод неявно решает задачу о срединных осях.

Применение диаграммы Вороного дает изящный метод получения довольно эффективного решения задачи кругового регионального поиска. Задача заключается в следующем: на плоскости заданы множество $S$, содержащее $N$ точек, и окружность $C$ с центром в точке $q$ (запрос); требуется перечислить точки множества $S$, попавшие внутрь $C$. Как обычно для задач регионального поиска, цель заключается в достижении оценки $O(f(N) + k)$, где $k$ — размер множества точек, соответствующих запросу. Если эта цель достижима, то алгоритм характеризуется парой $(M(N), f(N))$, где $M(N)$ — требуемый объем памяти. В главе 2 мы уже указывали на неадекватность подходов на основе регионального поиска к данной задаче. Исходную идею использовать диаграммы Вороного следует отнести на счет Бентли и Маурера [Bentley, Maurer (1979)]. Они предложили строить семейство диаграмм Вороного $\{\mathrm{Vor}_{2^i}(S)\colon i = 0, 1, \ldots, \lfloor \log_2 N \rfloor\}$ с последующим просмотром этой совокупности в порядке $i = 0, 1, 2, \ldots$. При просмотре диаграммы определяется область в $\mathrm{Vor}_{2^i}(S)$, содержащая центр окружности $q$, и проверяется соответствующий список соседей. Просмотр оканчивается, как только обнаруживается, что текущий проверяемый список содержит точку, расстояние до которой от $q$ больше $r$. Непосредственный анализ показывает, что этот алгоритм

характеризуется парой $(N^3, \log N \cdot \log\log N)$. Недавно эта идея была несколько улучшена [Chazelle, Cole, Preparata, Yap (1986)]. Применение общего подхода «фильтрующего поиска», обсуждавшегося в разд. 2.4, позволило получить алгоритм, характеризуемый парой $(N(\log N \cdot \log\log N)^2, \log N)$.

В разд. 6.1 отмечалось, что минимальное остовное дерево (МОД) множества $S$ из $N$ точек на плоскости является подграфом триангуляции Делоне (ТД). Имеются другие интересные геометрические структуры, аналогичным образом связанные с ТД и нашедшие приложения в распознавании образов и кластерном анализе. Это — граф Габриэля и граф относительного соседства. *Граф Габриэля* (ГГ) множества $S$ определяется следующим образом [Gabriel, Sokal (1969)]: пусть круг $(p_i, p_j)$ — круг с диаметром $\overline{p_ip_j}$; точки $p_i$ и $p_j$ множества $S$ соединяются ребром в ГГ лишь в случае, когда круг $(p_i, p_j)$ не содержит внутри точек множества $S$. Эффективный алгоритм построения ГГ за время $O(N \log N)$ основан на удалении из ТД всех ребер, не пересекающих двойственных им ребер диаграммы Вороного [Matula, Sokal (1980)]. Граф *относительного соседства* (ГОС) определяется так [Toussaint (1980b)]: точки $p_i$ и $p_j$ множества $S$ соединяются ребром тогда и только тогда, когда

$$\operatorname{dist}(p_i, p_j) \leqslant \min_{k \neq i, j} (\operatorname{dist}(p_i, p_k), \operatorname{dist}(p_j, p_k)).$$

Это определение означает, что граф содержит ребро $(p_i, p_j)$ лишь в том случае, когда пересечение двух кругов радиусом длина $(\overline{p_ip_j})$ с центрами в точках $p_i$ и $p_j$ не содержит внутри других точек множества $S$. Построение ГОС значительно более сложная задача, чем построение ГГ. Суповит [Supowit (1983)] разработал алгоритм со сложностью $O(N \log N)$. Между четырьмя упомянутыми графами имеется интересная взаимосвязь:

$$МОД \subseteq ГОС \subseteq ГГ \subseteq ТД. \tag{6.7}$$

В случае плоскости построение этих графов понято довольно хорошо, но для пространств более высокой размерности известно очень мало. В частности, одна из известных открытых задач заключается в разработке и анализе худшего случая алгоритма построения МОД в пространствах размерности три и выше.

Наконец, упомянем класс задач о декомпозиции объектов, заключающихся в разбиении заданного геометрического объекта на совокупность более простых «примитивных» геометрических объектов. Часто такими примитивными объектами являются выпуклые многоугольники, звездные многоугольники и т. д.

Среди таких задач большой интерес представляет задача разбиения внутренности простого многоугольника на треугольники, т. е. триангуляция простого многоугольника. Очевидно, что для решения этой задачи можно применить общий метод триангуляции с ограничениями, обсуждавшийся в разд. 6.2. Следуя таким путем, мы получим алгоритм, имеющий сложность $O(N \log N)$, где $N$ — число вершин многоугольника. Однако, как уже упоминалось ранее, нижняя оценка для задачи триангуляции с ограничениями не применима в данном случае. На протяжении почти десятилетия вопрос о том, можно ли, используя свойство простоты многоугольника, получить алгоритм триангуляции многоугольника, имеющий временну́ю сложность $o(N \log N)$, оставался без ответа, пока Тарьян и ван Вик [Tarjan, van Wyk (1987)] не разработали алгоритм решения этой задачи, имеющий временну́ю сложность $O(N \log\log N)$. Однако их результат не исчерпывает вопроса, делая еще более привлекательной цель, заключающуюся в разработке алгоритма с временно́й сложностью $O(N)$.



По существу, имеются два типа задач, связанных с декомпозицией: задачи разбиения объекта на непересекающиеся части и задачи покрытия, когда получающиеся части могут перекрываться. Иногда в целях декомпозиции объекта на минимальное число частей допускается введение дополнительных вершин, называемых *точками Штейнера*. Подробное обсуждение задач минимальной декомпозиции содержится в недавнем обзоре Кейла и Сака [Keil, Sack (1985)].

## 6.6. Упражнения

1. Детально разработать алгоритм построения минимального остовного дерева множества $S$ из $N$ точек на плоскости, использовав для этого пирамиду содержащую ребра триангуляции Делоне множества $S$. Ребро, извлекаемое из пирамиды, отклоняется или принимается в зависимости от того, образует оно цикл вместе с ребрами, принятыми ранее, или нет. Показать, что время работы этого алгоритма равно $\theta(N \log N)$.

2. Привести контрпример к утверждению, согласно которому триангуляция Делоне является минимально взвешенной триангуляцией.

3. Привести контрпример к утверждению, согласно которому жадная триангуляция является минимально взвешенной триангуляцией.

4. *Ли.* Показать, что в минимальном евклидовом паросочетании (задача Б.10) отрезки, определяемые парами точек паросочетания, не пересекаются.

5. *Зайдель.* Пусть $S = \{x_1, \ldots, x_N\} \subset \mathbb{R}$. Показать, что за линейное время можно построить множество $S' = \{p_1, \ldots, p_N\} \subset \mathbb{R}^2$ вместе с триангуляцией множества $S'$, где $x(p_i) = x_i$, $1 \leqslant i \leqslant N$.

6. Диаграмма $k$-го ближайшего соседа множества $S$ из $N$ точек представляет разбиение плоскости на области (не обязательно внутренне связанные) такие, что каждая область $R_j$ представляет множество точек плоскости, для которых заданная точка $p_j \in S$ является $k$-м соседом (т. е. для любой точки $q \in R_j$ длина отрезка $\overline{qp_j}$ является $k$-м элементом в упорядоченной в порядке возрастания последовательности длин отрезков $\overline{qp_i}$, $i = 1, \ldots, N$).

Показать связь диаграммы $k$-го ближайшего соседа с диаграммами $\mathrm{Vor}_{k-1}(S)$ и $\mathrm{Vor}_k(S)$, $k > 1$.

7. Показать, что в случае $L_1$-метрики диаграмма Вороного дальней точки множества $S$ из $N$ точек на плоскости ($N \geqslant 4$) состоит не более чем из *четырех* областей.

8. Рассмотреть преобразование инверсии на плоскости, отображающее точку $\mathbf{v}$ в $\mathbf{v}' = \mathbf{v}(1/|\mathbf{v}|^2)$.

(a) Получить уравнение для образа прямой, определяемой уравнением $ax + by + c = 0$ (образом прямой является окружность, проходящая через начало координат).

(b) Доказать, что если окружность $C$ содержит внутри начало координат, то внутренность окружности $C$ отображается на внешность ее образа при преобразовании инверсии.

9. Доказать лемму 6.4.

10. На плоскости задано множество $S$ из $N$ точек. Пусть $l$ — прямая, перпендикулярная отрезку $\overline{p_ip_j}$, $p_i$, $p_j \in S$, и делящая его пополам. Предположим, что $l$ содержит последовательность $v_1, v_2, \ldots, v_s$ вершин диаграмм Вороного, которые перечислены в порядке следования от одного конца прямой к другому. (Каждая вершина $v_i$ является центром окружности, описанной вокруг трех точек множества $S$.) Покажите, что если луч, заканчивающийся в вершине $v_1$, является ребром диаграммы $\mathrm{Vor}_h(S)$, то луч, заканчивающийся в вершине $v_s$, является ребром диаграммы $\mathrm{Vor}_{N-h}(S)$.

11. Доказать лемму 6.5.

12. *Зайдель.* Пусть $S = \{(x_i, y_i)\colon 1 \leqslant i \leqslant N\} \subset \mathbf{R}^2$ — множество из $N$ точек на плоскости. Обозначим через $S^1$ множество, получающееся в результате проецирования вдоль $z$-оси множества $S$ на параболоид вращения $z = x^2 + y^2$, т. е. $S^1 = \{(x_i, y_i, x_i^2 + y_i^2)\colon 1 \leqslant i \leqslant N\} \subset \mathbb{R}^3$. Рассмотрим выпуклую оболочку $P$ множества $S^1$.

(a) Показать по возможности наиболее прямым способом, что ортогональная проекция на плоскость $(x, y)$ граней оболочки $P$, находящихся в «нижней» части $P$, совпадает с триангуляцией Делоне множества $S$.

(b) Обобщить этот результат на случай пространств более высокой размерности.

13. *Зайдель.* Рассмотрим следующее обобщение диаграмм Вороного (известное под названиями: комплексы ячеек Дирихле, диаграмма Пауэра или диаграмма Лагерра). Пусть $S$ — множество из $N$ точек в $E^2$. С каждой точкой $p \in S$ связано действительное число $r_p$. Для каждой точки $p \in S$ определим множество

$$GV_p(S) = \{x \in \mathbb{R}^2\colon d(p, x)^2 + r_p \leqslant d(q, x)^2 + r_q,\ \forall q \in S\},$$

называемое обобщенной областью Вороного точки $p$ (относительно множества $S$). Назовем множество $\mathrm{GVor}(S) = \{GV_p(S)\colon p \in S\}$ обобщенной диаграммой Вороного множества $S$.

(a) Показать, что для каждой точки $p \in S$ область $GV_p(S)$ является выпуклой и имеет прямолинейные ребра.

(b) Показать, что в случае $r_p = 0$ для всех $p \in S$ обобщенная диаграмма Вороного превращается в обычную диаграмму Вороного множества $S$.

(c) Показать, что при надлежащем выборе значений чисел $r_p'$ можно добиться того, что для некоторых $p \in S$ области $GV_p(S)$ будут пустыми.

(d) Разработать алгоритм построения обобщенной диаграммы Вороного множества $S$, имеющий сложность $\theta(N \log N)$. (Указание: воспользуйтесь идеей, представленной в разд. 6.3.1 (диаграммы Вороного высших порядков).)



14. В задаче о НАИМЕНЬШЕЙ БОМБЕ требуется определить наименьшую окружность, охватывающую не менее $k$ из $N$ заданных точек на плоскости. Разработать полиномиальный алгоритм, решающий эту задачу.

15. Доказать, что граф Габриэля множества $S$ из $N$ точек (определение графа Габриэля дано в разд. 6.5) получается из триангуляции Делоне множества $S$ в результате удаления из нее каждого ребра, которое не пересекает двойственное ему ребро диаграммы Вороного.

16. Доказать отношение (6.7) в разд. 6.5.

17. *Ли.* Показать, что в случае $L_1$-метрики граф относительного соседства $N$ точек на плоскости по-прежнему является подграфом соответствующей триангуляции Делоне. Разработайте алгоритм построения этого графа.

18. Разработать алгоритм со сложностью $O(N \log N)$ построения срединных осей выпуклого $N$-угольника (определение срединных осей дано в разд. 6.5).

ГЛАВА 7

# Пересечения

Основной причиной изучения задач о пересечениях явился тот простой факт, что два объекта не могут одновременно находиться в одном и том же месте. Программа, предназначенная для архитектурного проектирования, должна внимательно позаботиться о том, чтобы двери не оказались там, где их невозможно открыть, а коридоры не проходили через шахту лифта. В машинной графике один объект, подлежащий выводу, заслоняет другой, если их проекции на картинную плоскость пересекаются. Выкройки можно вырезать из одного куска материи только тогда, когда их удается разложить так, что они не накрывают одна другую. Важность разработки эффективных алгоритмов определения пересечений станет ясной, если учесть, что в промышленных приложениях происходит постоянный рост сложности задач: сложное графическое изображение может состоять из сотни тысяч векторов, архитектурная база данных зачастую содержит свыше миллиона элементов, а одна интегральная схема может содержать миллионы компонентов. В подобных случаях даже квадратичные алгоритмы неприемлемы.

Другая причина для исследований сложности алгоритмов пересечения заключается в том, что они проливают свет на глубинную структуру геометрических задач и позволяют поставить ряд фундаментальных вопросов. Например, какова сложность решения вопроса о простоте многоугольника? Исследования подобных вопросов были бы оправданны, даже если бы они не имели никаких практических приложений, тем более что будет показано то, как широко используются алгоритмы данной главы. Поскольку две фигуры пересекаются только тогда, когда одна из них содержит хотя бы одну точку другой[1]), то естественно, что в алгоритмах пересечения не обойтись без проверки принадлежности точки. Поэтому можно считать задачи о пересечениях естественным расширением задач о принадлежности, рассмотренных в контексте геометрического поиска в гл. 2.

[1]) Это зависит от того, как определено пересечение границ.



Для того чтобы лучше представить себе данную область, рассмотрим теперь более детально ряд характерных приложений.

## 7.1. Примеры из приложений

### 7.1.1. Задачи удаления невидимых линий и поверхностей

Одной из важных задач машинной графики, привлекшей к себе внимание многих исследователей [Desens, 1969); Freeman, Loutrel (1967); Galimberti, Montanari (1969); Loutrel (1970); Matsushita (1969); Newman, Sproull (1973); Sutherland (1966); Warnock (1969); Watkins (1970)], является *задача удаления невидимых линий* и *поверхностей*. Двумерным образом

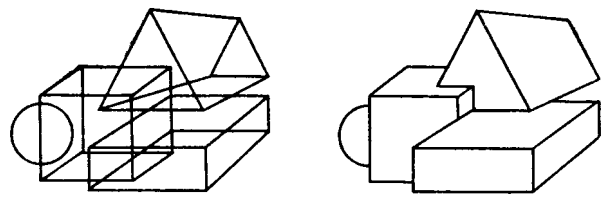


Рис. 7.1. Удаление невидимых линий.

трехмерной сцены всегда является какая-нибудь ее проекция. Однако нельзя просто спроецировать каждый из объектов на картинную плоскость, поскольку некоторые объекты могут быть частично или полностью скрыты от наблюдателя. Чтобы создать правдоподобный чертеж, необходимо удалить те линии, которые не смог бы увидеть реальный зритель. На рис. 7.1 показана сцена до и после удаления невидимых линий.

Один объект заслоняет другой, если их проекции пересекаются, поэтому определение пересечений и их построение является сердцевиной задачи удаления невидимых линий. В разработку технического обеспечения для решения этой задачи были сделаны значительные инвестиции, поскольку она особенно трудна на практике ввиду предъявляемых к графическим дисплейным системам требований работы в реальном времени и так как объекты обычно находятся в движении.

С точки зрения усилий, затраченных на развитие графической аппаратуры, вызывает удивление то, что вопросу о сложности решения задачи удаления невидимых линий уделялось

так мало внимания, поскольку именно здесь можно получить наибольший выигрыш. Конструирование черного ящика, снабженного микропрограммой даже при использовании параллельных вычислений, не способно, как правило, дать эффект, сравнимый с тем, что дает использование новейших алгоритмических методов (причем разрыв растет при увеличении размеров задачи).

Во многих случаях, особенно для векторных дисплеев, компонентами сцены являются многоугольники. Если проекциями двух объектов являются многоугольники $P_1$ и $P_2$, а $P_1$ ближе к зрителю, чем $P_2$, то необходимо изобразить $P_1$ и $P_2 \cap \bar{P}_1$ (понятно, что $P_2 \cap \bar{P}_1$ — это пересечение $P_2$ и дополнения к $P_1$).

Таким образом, основная вычислительная задача при удалении невидимых линий состоит *в формировании пересечения двух многоугольников*. На практике нужно сделать нечто большее, ибо изображение будет складываться из множества разных многоугольников, каждый из которых необходимо вывести, и тем не менее можно ожидать, что в основе алгоритма останется определение попарных пересечений. Не обладая оптимальным алгоритмом пересечения многоугольников, нельзя рассчитывать на эффективную реализацию удаления невидимых линий. В данной главе будут получены точные оценки алгоритмов пересечения для выпуклых, звездных и произвольных многоугольников. Задача с многоугольниками является примером первого типа из числа рассматриваемых нами задач о пересечениях. Для сохранения желаемого уровня общности рассуждений будем указывать род геометрических объектов, которые могут быть многоугольниками, отрезками, полиэдрами и т. п., только тогда, когда это потребуется в конкретном приложении. Итак, имеется:

**Задача типа С.1** [1]) (ПОСТРОЕНИЕ ПЕРЕСЕЧЕНИЯ). Даны два геометрических объекта. Надо построить их пересечение.

### 7.1.2. Разпознавание образов

Одним из основных методов распознавания образов является классификация путем *контролируемого обучения* [2]). Даны $N$ точек, каждая из которых идентифицируется по принадлежности к одному из $m$ классов; нужно их предварительно обработать таким образом, чтобы любую новую точку (*пробную точку*) можно было корректно классифицировать. На рис. 7.2 показан двумерный пример, на котором оси соответствуют весу и росту людей, принадлежащих к некой группе сверстников. Мужчины обозначены буквой «М», женщины — буквой «Ж». Буквой «Ч» обозначен человек, чьи вес и рост известны. Можно ли классифицировать Ч, основываясь только на значениях этих параметров? Каким решающим правилом надлежит воспользоваться?

[1]) С — аббревиатура слова «сечение». — *Прим. перев.*

[2]) Детали многих геометрических задач, возникающих при распознавании образов, можно найти в работах [Andrews (1972), Duda, Hart (1973), Meisel (1972)].



Желательно получить, если это возможно, *линейный классификатор*, т. е. такую линейную функцию $f$, чтобы для определения класса, к которому относится Ч, было достаточно

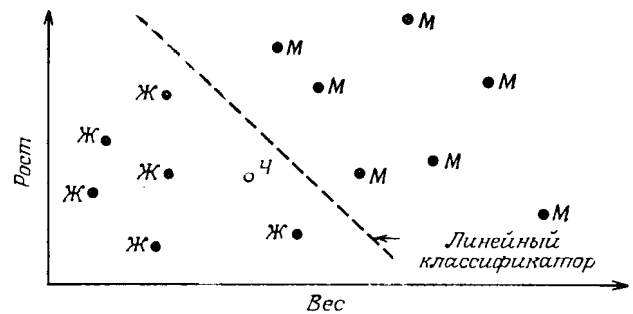



Рис. 7.2. Задача классификации с двумя переменными.

одного вычисления (собственно вычисления линейной функции и сравнения):

**if** $f(x_ч, y_ч) > T$ **then** Ч $\in$ М **else** Ч $\in$ Ж.

В последнем выражении через $T$ обозначено пороговое значение. В пространстве $k$ измерений $f(x_1, x_2, \ldots, x_k) = T$ является гиперплоскостью; при двух измерениях это прямая. Линейный классификатор считается хорошим, если он *разделяет* классы друг от друга так, что все точки типа М лежат по одну его сторону, а все точки Ж — по другую.

**Определение 7.1.** Два множества называются *линейно разделимыми* тогда и только тогда, когда существует такая гиперплоскость $H$, которая их разделяет.

Тем самым определение факта существования линейного классификатора является средством решения вопроса о разделимости обучающих выборок. Разделимость — это классическая задача комбинаторной геометрии. Основной критерий линейной разделимости дает следующая теорема:

**Теорема 7.1** [Stoer, Witzgall (1970), теорема 3.3.9]. *Два множества точек линейно разделимы тогда и только тогда, когда их выпуклые оболочки не пересекаются.*

Эта теорема проиллюстрирована для плоского случая на рис. 7.3. Поскольку известно, что выпуклой оболочкой конечного множества точек является выпуклый политоп, то линейная разделимость определяется проверкой пересечения двух выпуклых политопов. Последняя задача является примером задач следующего типа.

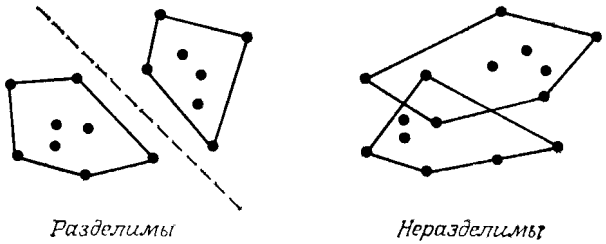



Рис. 7.3. Два множества разделимы тогда и только тогда, когда их выпуклые оболочки не пересекаются.

**Задача типа С.2** (ПРОВЕРКА ПЕРЕСЕЧЕНИЯ). Даны два геометрических объекта. Пересекаются ли они?

Позднее будет показано, что проверка пересечения зачастую проще, чем соответствующая ей задача построения.

### 7.1.3. Трассировка и размещение

Поскольку микроминиатюризация в электронике достигла фантастических темпов, то количество компонент в чипах, проводников на платах и проводов в схемах выросло настолько, что подобное оборудование невозможно спроектировать без помощи ЭВМ. Число элементов в одной интегральной схеме вполне может превысить миллион, и каждый из них должен быть размещен конструктором с учетом множества физических и геометрических ограничений. Используемые при этом программы основаны на эвристиках и часто дают некорректные решения, например наложения компонент схемы или пересечения проводников (см. [Akers (1972); Hanan, Kurzberg (1972); Hanan (1975)]). Эвристические методы используются, поскольку некоторые из задач, связанных с размещением компонент, являются *NP*-полными [Garey, Johnson, Stockmeyer (1976)]. Следовательно, решения необходимо подвергать тщательной проверке, состоящей из попарных сравнений всех элементов чипа, что дорого и требует много времени. Это приводит к следующему теоретически важному классу задач.

**Задача типа С.3** (ПОПАРНЫЕ ПЕРЕСЕЧЕНИЯ). Даны $N$ геометрических объектов. Надо определить все их попарные пересечения.



Разумеется, мы будем искать такой алгоритм, при котором не надо сравнивать каждый элемент со всеми остальными. Решение задачи этого типа, которое будет дано в разд. 7.2.3, имеет много практических и теоретических приложений (см. также гл. 8).

### 7.1.4. Линейное программирование и общее пересечение полупространств

Линейное программирование можно отнести к четвертому типу задачи пересечения. Областью допустимых решений задачи линейного программирования является пересечение полупространств, определенных множеством ее ограничений. Целевая функция достигает максимума в одной из вершин выпуклого полиэдра, способ описания которого отличается от того, что рассматривался в гл. 3. Здесь у нас нет множества точек, среди которых следует найти вершины оболочки, вместо этого имеется набор полуплоскостей, ограничивающих эту оболочку, а нам нужно найти вершины. Очевидно, что после построения общей области для данных $N$ объектов мы сможем получить решение задачи линейного программирования. Однако нам нужно найти лишь ту вершину, на которой целевая функция экстремальна (максимальна или минимальна) и нет оснований полагать, что даже для случая малой размерности необходимо построение *всей* полиэдральной области.

Одномерная задача линейного программирования тривиальна. Ее можно сформулировать следующим образом:

$$ax + b \to \max \quad \text{при} \quad a_i x + b_i \leqslant 0, \quad i = 1, \ldots, N. \tag{7.1}$$

Допустимая область тут пуста, является отрезком, или лучом, поскольку возможно такое пересечение лучей, которое продолжается в $-\infty$ или $+\infty$.

Пусть $L$ — самая левая из точек положительно ориентированных лучей, а $R$ — самая правая из точек отрицательных лучей. Если $L > R$, то допустимая область пуста. Если $L \leqslant R$, то ею является отрезок $[L, R]$. Очевидно, что $L$ и $R$ можно вычислить за линейное время, поэтому оценка по времени для одномерной задачи линейного программирования пропорциональна $O(N)$. При большем числе измерений задачу линейного программирования можно решить путем построения общей области (пересечения) полупространств. Однако эти две задачи не эквивалентны, как будет показано более точно в разд. 7.2.5.

Теперь обсудим несколько основных алгоритмов, предназначенных для решения вышеописанных задач. Большинство этих алгоритмов создано для задач типа ПОСТРОЕНИЕ ПЕРЕСЕЧЕНИЯ (задачи типа С.1), и тем не менее всюду, где это

уместно, будет проводиться сопоставление с соответствующими задачами типа «проверки». Мы начнем с плоских приложений, а затем перейдем к пространственным примерам. Нужно ли повторять, что уровень знаний в этой области при росте числа измерений отражает картину, знакомую сегодня во всей вычислительной геометрии.

## 7.2. Плоские приложения

### 7.2.1. Пересечение выпуклых многоугольников

В данном разделе под «многоугольником» будем понимать его границу и внутренность; цикл из ребер многоугольника будет явно называться его границей. Сформулируем задачу:

**Задача С.1.1** (ПОСТРОЕНИЕ ПЕРЕСЕЧЕНИЯ ВЫПУКЛЫХ МНОГОУГОЛЬНИКОВ). Даны два выпуклых многоугольника: $P$ с $L$ вершинами и $Q$ с $M$ вершинами. Надо построить их пересечение.

Без потери общности положим, что $L \leqslant M$.

**Теорема 7.2.** *Пересечением выпуклых $L$-угольника и $M$-угольника является выпуклый многоугольник, имеющий не более $L+M$ вершин.*

*Доказательство.* Пересечение $P$ и $Q$ образовано пересечением $L+M$ внутренних полуплоскостей, определяемых ребрами этих двух многоугольников. Что и требовалось доказать.

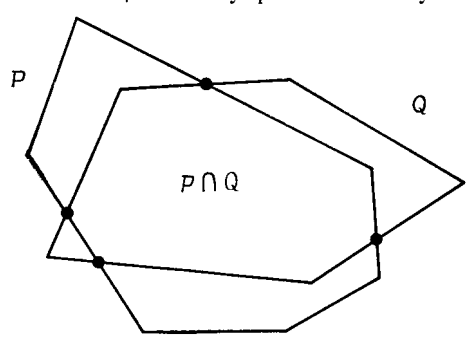


Рис. 7.4. Пересечение выпуклых многоугольников.

Граница $P \cap Q$ состоит из чередующихся цепей вершин этих двух многоугольников, разделенных точками пересечения их границ (рис. 7.4).

Очевидный метод формирования пересечения состоит в обходе границы $P$ ребро за ребром, в процессе которого просто и безыскусно определяются все точки пересечения с границей $Q$ (не более двух на каждое ребро) и попутно ведется список точек пересечения и вершин. Это потребует затраты времени $O(LM)$, поскольку будет проверено пересечение каждого ребра $P$ с каждым ребром $Q$. Естественно для сокращения вычислительной работы попытаться использовать факт выпуклости многоугольников.



Один из подходов заключается в разбиении плоскости на области, в каждой из которых пересечение этих двух многоугольников можно вычислить легко. Построение новых геометрических объектов для решения поставленной задачи не редкость в вычислительной геометрии; читатель может вспомнить исключительное значение этого приема во многих методах геометрического поиска, показанных в гл. 2. В данном случае будет адекватным простейшее разбиение плоскости — индуцированное пучком лучей [Shamos, Hoey (1976)]. Выберем произвольную точку $O$ на плоскости. (Эту точку $O$ можно взять на бесконечности; в действительности исходный метод Шеймоса — Хоуи основан на выборе $O$ в бесконечности на оси $y$.) Из $O$ проводим лучи через все вершины многоугольников $P$ и $Q$. Эти лучи разбивают плоскость на секторы. Мы хотим упорядочить этот набор лучей по углу вокруг точки $O$. К счастью, это легко сделать. На самом деле, если $O$ лежит внутри многоугольника, скажем, $P$, то лучи к вершинам $P$, очевидно, упорядочиваются относительно $O$ так же, как и вершины, порядок которых задан. Если же $O$ лежит вне $P$, то вершины $P$ образуют две цепи (обе они заключены между теми двумя вершинами $P$, которых касаются опорные прямые из $O$), а для каждой цепи соответствующие ей лучи из $O$ отсортированы по углу. Поэтому в любом случае эти секторы, определяемые вершинами $P$, упорядочиваются по углу за время, линейно зависящее от числа вершин. Если эти две последовательности (одна для $P$ и одна для $Q$) известны, то их можно слить за линейное время.

Ключевым является наблюдение, что пересечение каждого сектора с выпуклым многоугольником образует *четырехугольник* (*трапецию*, если $O$ — бесконечно удаленная точка). Поэтому внутри каждого сектора пересечением $P$ и $Q$ будет пересечение двух четырехугольников, которое можно найти за константное время. Результирующие куски можно собрать воедино за один обход секторов, на который понадобится линейное время. А затем потребуется очищающий просмотр для удаления ложных вершин, которые возникают на границах между секторами.

**Теорема 7.3.** *Пересечение выпуклых $L$-угольника и $M$-угольника можно построить за время $\theta(L+M)$.*

На рис. 7.5 проиллюстрирован метод, где точка $O$ выбрана так, как предложили Шеймос и Хоуи. В данном случае секторы называются полосами, а полярный угол относительно $O$ заменен значением абсциссы $x$.

Другой подход является, как это ни удивительно, элегантным усовершенствованием [O'Rourke, Chien, Olson, Naddor (1982)] того грубого метода, который был описан в начале данного

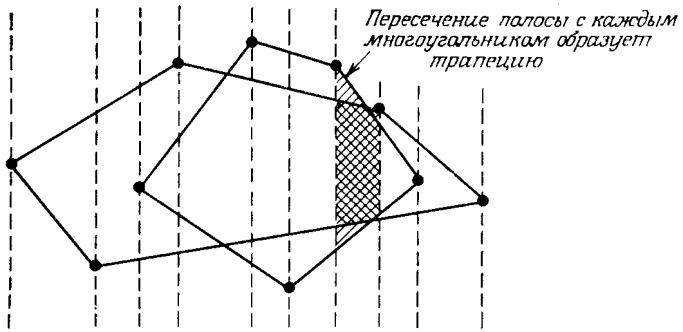



Рис. 7.5. Полосы, определенные вершинами двух выпуклых многоугольников.

раздела. Его основная идея неформально следующая. Предположим, что $P \cap Q \neq \varnothing$, и рассмотрим многоугольник $P^* \triangleq P \cap Q$. Границей $P^*$ является чередующаяся последовательность участков границ $P$ и $Q$. Если одним из таких участков является кусок границы, скажем, $P$, то некий участок границы $Q$ окружает ее во внешней области $P^*$ (рис. 7.6). Благодаря выпуклости обоих многоугольников в каждой паре таких смежных

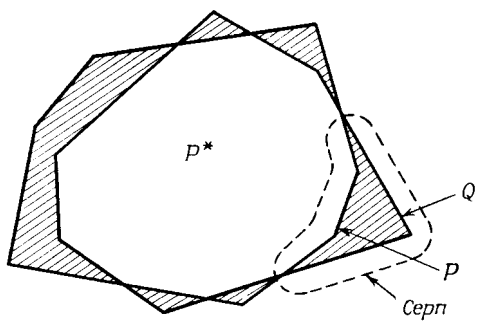



Рис. 7.6. Два пересекающихся многоугольника. «Серпы» заштрихованы.

участков естественно выделяются *внешняя* и *внутренняя цепи*, которые можно назвать в совокупности «серпом» для удобства последующего описания. Серп ограничен парой точек пересечения, именуемых *начальной* и *конечной* в соответствии с единой ориентацией границ $P$ и $Q$. Говорят, что вершина (из $P$ или $Q$) принадлежит серпу, если (1) она лежит между начальной и конечной точками пересечения (на любой цепи) или (2) является концом ребра внутренней цепи, содержащего конечную точку серпа (рис. 7.7). Заметим, что существуют вершины, принадлежащие двум серпам.



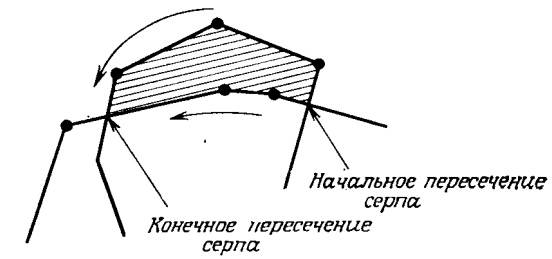



Рис. 7.7. Названия элементов серпа. Выделенные вершины принадлежат заштрихованному серпу.

Для создания механизма продвижения положим, что $(p_1, p_2, \ldots, p_L)$ и $(q_1, q_2, \ldots, q_M)$ — это списки вершин $P$ и $Q$ соответственно, перечисленных при обходе их против часовой стрелки (так что многоугольник считается расположенным «слева» от своей границы). Предположим, что продвижение осуществляется одновременно по обеим границам и что $p_i$ и $q_j$ — текущие вершины заданных многоугольников, а, кроме того, *текущие ребра* оканчиваются в $p_i$ и $q_j$; возможны четыре разные ситуации, которые показаны на рис. 7.8. (Остальные ситуации сводятся к данным четырем путем замены ролей $P$ и $Q$.) Пусть $h(p_i)$ обозначает полуплоскость, определенную $\overline{p_{i-1}p_i}$ и содержащую $P$ (аналогично определяется и $h(q_j)$). Очевидно, что $P^*$ содержится внутри пересечения $h(p_i)$ и $h(q_j)$, т. е. в заштрихованных областях на рис. 7.8.

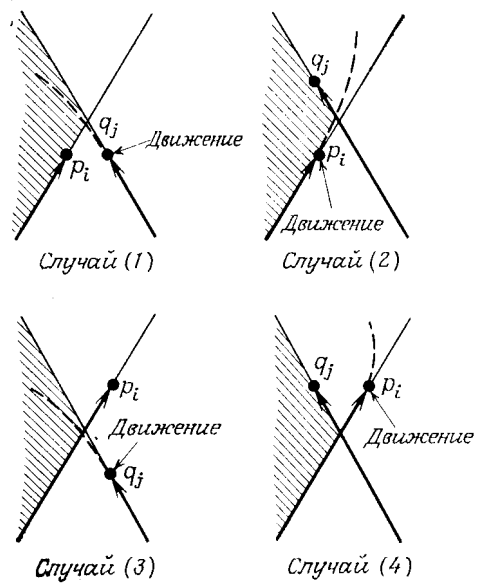



Рис. 7.8. Иллюстрация механизма продвижения. Мы продвигаемся по одной или по другой границе в зависимости от относительного расположения текущих вершин $p_i$ и $q_j$ (текущие ребра показаны жирными прямыми).

Идея заключается в том, чтобы *не двигаться по той границе* ($P$ или $Q$), *текущее ребро которой еще может содержать необнаруженную точку пересечения*. Тогда в случае (2) мы продвинемся по $P$, поскольку текущее ребро $\overline{q_{j-1}q_j}$ из $Q$ может содержать еще не обнаруженную точку пересечения; по тем же причинам в случае (3) мы продвинемся по границе $Q$. В случае (4) все пересечения на текущем ребре из $P$ уже обнаружены, в то время как текущее ребро из $Q$ все еще может содержать необнаруженное пересечение; поэтому мы продвинемся вдоль $P$. Наконец, в случае (1) выбор произволен, и выберем, например, $Q$. Рассмотрение этих четырех случаев полностью определяет механизм продвижения в данном методе, осуществляемый подпрограммой ДВИЖЕНИЕ. Выражение «ДВИЖЕНИЕ реализует случай ($j$)» (при $j = 1, 2, 3, 4$) означает, что продвижение, соответствующее случаю ($j$), выполняется корректно.

Если пренебречь для ясности и простоты вырожденными случаями (когда граница $P$ содержит вершину $Q$ и *наоборот*), которые требуют особого внимания[1]), то можно формализовать алгоритм. Заметим, что $p_L$ и $q_M$ непосредственно предшествуют $p_1$ и $q_1$ соответственно (это означает, что, как обычно, $p_0 = p_L$ и $q_0 = q_M$).

```
        procedure ПОСТРОЕНИЕ-ПЕРЕСЕЧЕНИЯ-
                  ВЫПУКЛЫХ-МНОГОУГОЛЬНИКОВ
begin   i := j := k := 1:
        repeat
          begin if (p_{i-1}p_i и q_{j-1}q_j пересекаются) then
                    вывод пересечения;
                    ДВИЖЕНИЕ (* i или j увеличивается *);
                    k := k + 1
          end;
        until  k = 2(L + M);
        if (не найдено пересечений) then
          begin if p_i ∈ Q then P ⊆ Q
                    else if q_j ∈ P then Q ⊆ P
                        else P ∩ Q = ∅
          end;
end.
```

Теперь установим корректность данного алгоритма. Если $p_i$ и $q_j$ принадлежат одному серпу, то подпрограмма ДВИЖЕНИЕ гарантирует достижение конечной точки пересечения данного серпа (как *очередного* пересечения), поскольку не допускается

[1]) Подробности можно найти в первоисточнике [O'Rourke *et al.* (1982)].



продвижения по той границе, текущее ребро которой может содержать искомую точку пересечения. Это в свою очередь гарантирует, что поскольку обе текущие вершины принадлежат одному и тому же серпу, то все серпы будут построены последовательно. Для завершения доказательства необходимо убедиться, что данный алгоритм обнаруживает хотя бы одну точку пересечения, если она существует. Заметим, что точки пересечения образуют пары, поэтому будет по крайней мере две точки

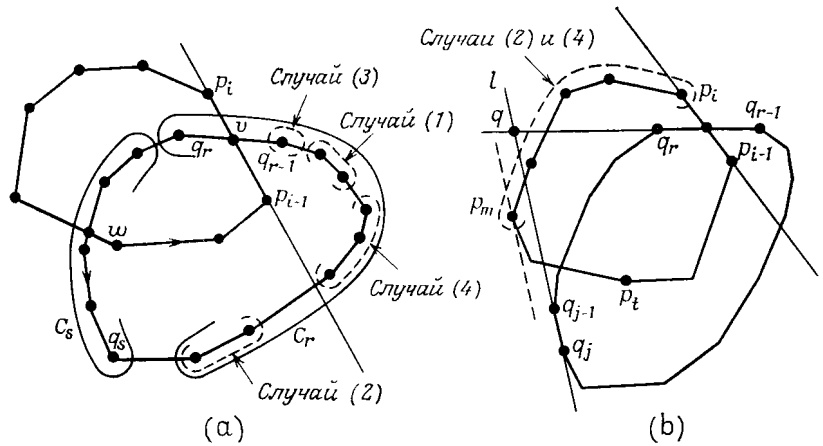



Рис. 7.9. Иллюстрация к доказательству того, что алгоритм находит по крайней мере одну точку пересечения, если $P$ и $Q$ пересекаются.

пересечения. Пусть $\overline{p_{i-1}p_i}$ — текущее ребро $P$ — содержит точку пересечения $v$ с ребром $\overline{q_{r-1}q_r}$, так что $q_r \in h(p_i)$ (рис. 7.9). Обозначим через $w$ следующую точку пересечения в серпе, начавшемся в точке $v$; существуют еще две важные точки в $Q$: вершина $q_r$, уже определенная, и вершина $q_s$, наиболее удаленная в $h(p_i)$ от прямой, несущей ребро $\overline{p_{i-1}p_i}$. Граница $Q$ разбивается этими двумя вершинами на две ориентированные цепи $C_r$ и $C_s$, которые оканчиваются в вершинах $q_r$ и $q_s$ соответственно. Считая, как обычно, что $\overline{q_{j-1}q_j}$ обозначает текущее ребро в $Q$, будем различать следующие два случая:

(1) $q_j \in C_r$. Подпрограмма ДВИЖЕНИЕ последовательно обрабатывает случаи (каждый из которых может оказаться пустым) (2), (4), (1) и (3), как показано на рис. 7.9(a). Продвижение ведется вдоль $Q$, в то время как ребро $\overline{p_{i-1}p_i}$ остается неизменным вплоть до того, как обнаруживается $v$.

(2) $q_j \in C_s$ (рис. 7.9(b)). Пусть $l$ — прямая, несущая $\overline{q_{j-1}q_j}$. Существуют две опорные прямые к $P$, параллельные $l$. Пусть $p_m$ — та вершина, принадлежащая одной из этих опорных пря-

мых, которая встретится первой при просмотре границы $P$, начиная с $p_i$. Очевидно, что подпрограмма ДВИЖЕНИЕ (случаи (2) и (4)) проходит от $p_i$ до $p_m$, в то время как ребро $\overline{q_{j-1}q_j}$ остается неизменным. Затем, если $p_m \notin h(q_j)$, то ДВИЖЕНИЕ далее проследует к первой вершине $p_t \in h(q_j)$. В любом случае текущими вершинами остаются $q_j$ и $p^* \in P$ ($p^* = p_m$ или $p^* = p_t$), лежащие внутри $h(q_j)$. Заметим, что данная ситуация не отличается от такой, как если бы одна вершина $q$ заменила подцепь $(q_r, q_{r+1}, \ldots, q_{j-1})$, где $q$ — пересечение прямых, несущих $\overline{q_{r-1}q_r}$ и $\overline{q_{j-1}q_j}$ соответственно. Поэтому можно считать, как будто $q_j$ и $p^*$ принадлежат к одному и тому же серпу, а в этом случае подпрограмма ДВИЖЕНИЕ обязательно приведет к конечной точке этого серпа.

Если ребро, подобное $\overline{p_{i-1}p_i}$, существует (т. е. границы $P$ и $Q$ пересекаются), то после $(L+M)$ шагов подпрограммы ДВИЖЕНИЕ оно обязательно будет достигнуто; действительно, после $(L+M)$ шагов граница по крайней мере одного многоугольника будет полностью пройдена, а каждый многоугольник имеет по крайней мере одно ребро, подобное $\overline{p_{i-1}p_i}$. Легко убедиться в том, что затем понадобится еще не более $(L+M)$

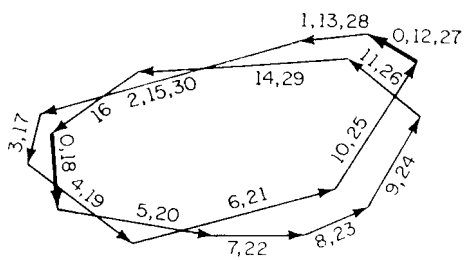



Рис. 7.10. Иллюстрация работы алгоритма О'Рурка и др. Ребро, помеченное числом $j$, обрабатывается подпрограммой ДВИЖЕНИЕ на $j$-й итерации. Начальные ребра выделены жирной линией.

дополнительных шагов подпрограммы ДВИЖЕНИЕ, чтобы построить всю границу $P \cap Q$. Итак, заключаем, что если после $2(L+M)$ шагов подпрограммы ДВИЖЕНИЕ данный алгоритм не смог обнаружить ни одной точки пересечения, то границы исходных многоугольников вовсе не пересекаются. С точки зрения оценки работы ДВИЖЕНИЕ завершается за время $O(L+M)$; кроме того, $O(L+M)$ шагов нужны, чтобы сделать выбор, если это необходимо, между следующими альтернативами: $P \subseteq Q$, $Q \subseteq P$, $P \cap Q = \varnothing$. Поэтому данный метод служит еще одним доказательством теоремы 7.3.

Иллюстрация работы этого алгоритма дана на рис. 7.10. Ребро там получает метку $j$, если цикл *repeat* (шаг подпрограммы ДВИЖЕНИЕ) достигает его на $j$-й итерации. Пара исходных ребер (одно из $P$ и одно из $Q$ выделены жирными линиями) корректно помечена нулями.



### 7.2.2. Пересечение звездных многоугольников

Поскольку звездные многоугольники легко описывать в полярной системе координат — таким же свойством обладают и выпуклые многоугольники, — то можно было бы ожидать, что и их пересечения удастся находить столь же быстро. Однако это не так, что и показано на рис. 7.11.

Пересечение $P$ и $Q$ уже само по себе оказывается не многоугольником, а объединением многих многоугольников. Если и $P$, и $Q$ имеют по $N$ вершин, а каждое ребро $P$ пересекает каждое ребро $Q$, то их пересечение имеет порядок $N^2$ вершин. Отсюда получаем тривиальную нижнюю оценку:

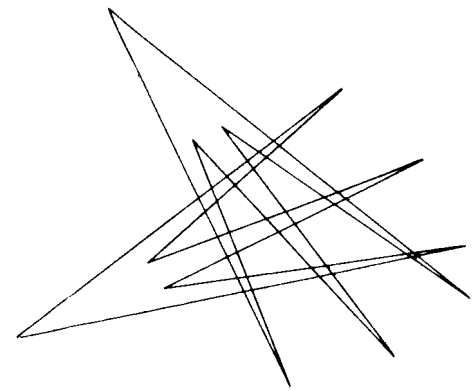


Рис. 7.11. Пересечение двух звездных многоугольников.

**Теорема 7.4.** *Поиск пересечения двух звездных $N$-угольников занимает в худшем случае $\Omega(N^2)$ времени.*

Это означает, что задача об удалении невидимых линий для произвольных многоугольников также должна потребовать в худшем случае квадратичного времени просто потому, что для того, чтобы нарисовать их пересечения, необходимо $\Omega(N^2)$ векторов. В следующем разделе будет показан удивительный результат, состоящий в том, что если нужно установить только факт пересечения $P$ и $Q$, то нет необходимости тратить так много времени.

### 7.2.3. Пересечение прямолинейных отрезков

Один из главных выводов вычислительной геометрии состоит в том, что большой набор, казалось бы, независимых задач можно решить одним и тем же способом, если только удается выделить общие для них алгоритмические особенности. В данном разделе показано, как множество различных приложений можно унифицировать и свести к решению задачи о том, будут или нет попарно непересекающимися $N$ прямолинейных отрезков из заданного множества.

**Задача С.2.1** (ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ). Даны $N$ прямолинейных отрезков на плоскости. Надо определить факт пересечения хотя бы двух из них.

Алгоритм решения этой задачи был бы весьма полезен для приложений типа трассировки или размещения, описанных в разд. 7.1.3. Прежде чем взяться за решение задачи С.2.1, опишем дополнительно ее важные приложения.

### 7.2.3.1. Приложения

**Задача С.2.2** (ПРОВЕРКА ПЕРЕСЕЧЕНИЯ МНОГОУГОЛЬНИКОВ). Даны два простых многоугольника $P$ и $Q$ с $M$ и $N$ вершинами соответственно. Пересекаются ли они?

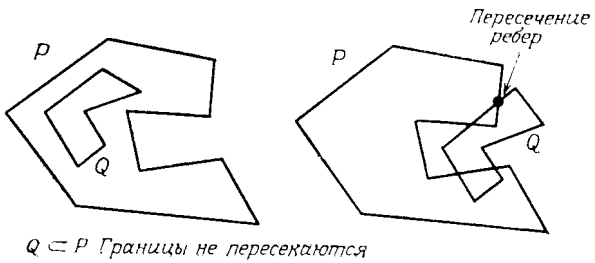



Рис. 7.12. $P \subset Q$, или $Q \subset P$, или же есть пара пересекающихся ребер.

Как указано в разд. 7.2.1 для выпуклых многоугольников, если $P$ и $Q$ пересекаются, то или $P$ содержит $Q$, или $Q$ содержит $P$, или некоторое ребро $P$ пересекает одно из ребер $Q$ (рис. 7.12).

Поскольку $P$ и $Q$ оба просты, то при любом пересечении их ребер эти ребра будут принадлежать разным многоугольникам. Обозначим через $T(N)$ время, необходимое для решения задачи С.2.1. Тогда можно определить факт пересечения ребер $P$ и $Q$ за $T(N+M)$ операций. И если пересечений не найдено, то останется только проверить $P \subset Q$ или $Q \subset P$.

Если $P$ лежит внутри $Q$, то *все* вершины $P$ лежат внутри $Q$, поэтому можно один раз применить проверку принадлежности точки за время $O(N)$ (по теореме 2.1), используя *любую* вершину $P$. Если эта вершина лежит вне $Q$, то можно тем же самым способом установить, верно ли, что $Q \subset P$, за время $O(M)$. Отсюда имеем следующую теорему:

**Теорема 7.5.** *Пересечение простых многоугольников преобразуется за линейное время к проверке пересечения прямолинейных отрезков*:

ПРОВЕРКА ПЕРЕСЕЧЕНИЯ МНОГОУГОЛЬНИКОВ $\propto_N$ ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ



Мы уже видели, что сложность алгоритмов, работающих с многоугольниками, может зависеть от того, известна или нет простота этих многоугольников. Например, выпуклую оболочку для простого многоугольника можно найти за время $\theta(M)$ (теорема 4.12), в то время как $\Omega(N \log N)$ является нижней границей для непростых многоугольников. Поэтому было бы полезно иметь алгоритм проверки простоты.

**Задача С.2.3** (ПРОВЕРКА ПРОСТОТЫ МНОГОУГОЛЬНИКА). Дан многоугольник. Прост ли он?

Простой и непростые многоугольники показаны на рис. 7.13. Поскольку многоугольник прост тогда и только тогда, когда

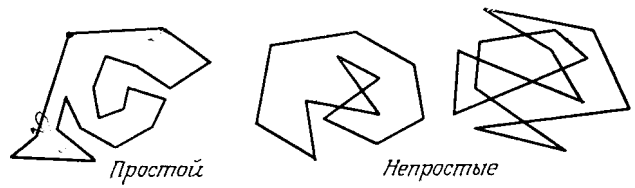



Рис. 7.13. Простой и непростые многоугольники.

никакая пара его ребер не пересекается, то немедленно имеем, что:

ПРОВЕРКА ПРОСТОТЫ МНОГОУГОЛЬНИКА $\propto_N$ ПРОВЕРКА ПЕРЕСЕЧЕНИЯ ПРЯМОЛИНЕЙНЫХ ОТРЕЗКОВ

### 7.2.3.2. Алгоритмы пересечения отрезков

Предположим, что заданы $N$ интервалов на действительной оси и необходимо узнать, не перекрываются ли какие-нибудь два из них. Ответ можно получить за время $O(N^2)$, проверив все пары интервалов, но тут же на ум приходит намного лучший алгоритм, основанный на сортировке. Если упорядочить $2N$ концевых точек этих интервалов и обозначить их как левые (Л) или правые (П), то эти интервалы не перекрываются тогда и только тогда, когда концы их образуют чередующуюся последовательность: Л, П, Л, П, ..., П, Л, П (рис. 7.14). Эту проверку можно провести за $O(N \log N)$ времени.

Хотелось бы ответить на два вопроса: можно ли улучшить этот алгоритм и обобщается ли он на случай двух измерений?

Чтобы получить нижнюю оценку, продемонстрируем соответствие между задачей о наложении интервалов и известным фундаментальным вопросом из теории множеств: ПРОВЕРКОЙ УНИКАЛЬНОСТИ ЭЛЕМЕНТОВ (даны $N$ действительных чисел; все ли они различны? (разд. 5.2)). Ранее было показано,

что эта задача легко разрешима за время $O(N \log N)$ методом сортировки, и, хотя нет простого способа показать, что сортировка необходима, тем не менее известно (из результатов Добкина и Липтона [Dobkin, Lipton (1976)] и Бен-Ора [Ben-Or (1983)]), что необходимо затратить $O(N \log N)$ времени в

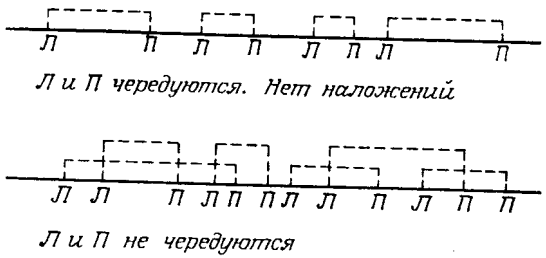



Рис. 7.14. Определение наложений интервалов.

модели вычислений типа дерева решений (следствие 5.1). Теперь покажем, что

УНИКАЛЬНОСТЬ ЭЛЕМЕНТОВ $\propto_N$ НАЛОЖЕНИЕ ИНТЕРВАЛОВ

Заданный набор из $N$ действительных чисел $\{x_i\}$ можно преобразовать в набор из $N$ интервалов $\{[x_i, x_i]\}$ за линейное время. Эти интервалы перекрываются тогда и только тогда, когда исходные числа не уникальны, и это доказывает следующую теорему:

**Теорема 7.6.** *Для того чтобы определить, перекрываются ли $N$ интервалов, необходимо и достаточно $\theta(N \log N)$ сравнений, при условии что можно использовать лишь алгебраические функции на входе.*

Насколько строгим является это ограничение на алгебраические функции? С одной стороны, оно запрещает использовать функцию «целая часть», которая, как было показано в разд. 6.4, является очень мощной операцией. (Напомним, что использование этой функции позволило Гонзалесу [Gonzales (1975)] решить задачу MAXGAP за $\theta(N)$ времени.) Методы, которые позволили бы доказать теорему 7.6 с использованием функции «целая часть», не известны, однако можно предположить, что ее добавление вряд ли повлияет на нижнюю оценку, *тем более* что теорема 7.6 приложима к случаю любого числа измерений. В частности, для проверки пересечения $N$ отрезков на плоскости необходимо $\Omega(N \log N)$ операций.

Теперь посмотрим, что же происходит на самом деле при сортировке для выявления наложений. Мотивом для этого исследования служит отсутствие естественного полного упорядочения отрезков на плоскости, поэтому на обобщение, основанное только на сортировке, не приходится рассчитывать. Однако если мы сможем понять существенные черты одномерного алгоритма, то нам удастся обобщить его и на плоский случай.



Два интервала перекрываются тогда и только тогда, когда они содержат хотя бы одну общую точку. Каждая точка на действительной оси связана с множеством, состоящим из накрывающих ее интервалов. Это соответствие определяет функцию $C$: $\mathbb{R} \to \{0, 1\}^N$, отображающую действительные числа на подмножества множества $\{1, \ldots, N\}$. Значение функции $C$ может меняться только на множестве из $2N$ концевых точек заданных интервалов. Если мощность $\{C(x)\}$ превышает мощность последнего множества, то интервалы перекрываются. Чтобы обнаружить это, отсортируем концы интервалов и создадим простейшую структуру данных, состоящую всего лишь из одного целого числа, т. е. числа интервалов, накрывающих текущую абсциссу. Просматривая концы отрезков слева направо, ВСТАВЛЯЕМ интервал в эту структуру данных в тот момент, когда встречен его левый конец, и УДАЛЯЕМ его в момент встречи правого конца. Всякая попытка ВСТАВИТЬ в структуру, в которой уже хранится число один, свидетельствует о наложении; в противном случае наложений нет. Поскольку обработка каждой концевой точки при таком подходе занимает лишь константное время, то процесс проверки (после сортировки) требует не более чем линейного времени.

При двух измерениях необходимо определить иное отношение порядка и использовать более совершенную структуру данных[1]). Рассмотрим пару непересекающихся отрезков $s_1$ и $s_2$ на плоскости. Будем считать, что $s_1$ и $s_2$ *сравнимы в абсциссе* $x$, если существует такая вертикаль, проходящая через $x$, которая пересекает оба отрезка. Введем отношение *выше в* $x$ следующим образом: $s_1$ выше $s_2$ в $x$ (пишется $s_1 >_x s_2$), если $s_1$ и $s_2$ сравнимы в $x$, а точка пересечения $s_1$ с вертикалью $x$ лежит выше точки пересечения $s_2$ с ней же[2]). На рис. 7.15 показаны следующие отношения между отрезками $s_1$, $s_2$, $s_3$ и $s_4$:

$$s_2 >_u s_4,\ s_1 >_v s_2,\ s_2 >_v s_4,\ s_1 >_v s_4.$$

Отрезок $s_3$ не сравним ни с каким другим отрезком.

[1]) Для простоты изложения предположим, что нет вертикальных отрезков и что никакие три отрезка не пересекаются в одной точке. Если нарушить любое из этих условий, то разрабатываемые алгоритмы станут длиннее в деталях, но не в асимптотической оценке времени работы.

[2]) Это отношение порядка и алгоритм, следующий из него, были предложены Дэном Хоуи. Доказательство корректности этого алгоритма было дано М. Шеймосом. Оба результата были опубликованы в работе [Shamos, Hoey (1976)].

Заметим, что отношение $>_x$ задает полное упорядочение, которое изменяется по мере того, как вертикаль скользит слева направо. Отрезки входят в это упорядочение и покидают его, но оно всегда остается полным. Упорядочение может изменяться только в трех случаях:

1. *Встретился левый конец отрезка* $s$. В этом случае $s$ надо добавить к структуре данных.

2. *Встретился правый конец отрезка* $s$. В этом случае $s$ надо удалить из структуры данных, поскольку он больше не сравним с оставшимися отрезками.

3. *Обнаружена точка пересечения отрезков* $s_1$ *и* $s_2$. Тогда $s_1$ и $s_2$ меняются местами в структуре данных.

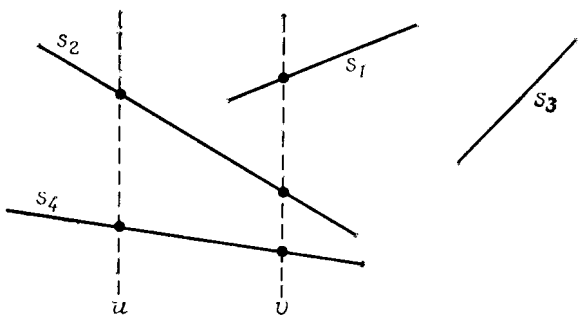



Рис. 7.15. Отношение порядка между отрезками.

Заметим, что необходимое условие пересечения двух отрезков $s_1$ и $s_2$ состоит в том, что существует такая абсцисса $x$, в которой $s_1$ и $s_2$ смежны в упорядочении $>_x$. Отсюда немедленно следует, что последовательность пересечений множества отрезков с вертикалью в $x$ (т. е. отношение $>_x$) содержит всю необходимую информацию для поиска пересечений отрезков и что естественным методом для решения задач этого класса является «плоское заметание» (см. разд. 1.2.2, где дана общая формулировка этого алгоритмического метода). Напомним, что в методе плоского заметания используются две главные структуры данных: *статус заметающей прямой* и *список точек событий*.

Как указано выше, статус заметающей прямой является описанием отношения $>_x$; значит, он представляет собой последовательность элементов (отрезков). Если вспомнить ранее описанный механизм, в котором отношение $>_x$ изменяется на конечном множестве абсцисс при плоском заметании, то станет очевидно, что в структуре данных $\mathscr{L}$, реализующей статус заметающей прямой, должны быть предусмотрены следующие операции:



a. ВСТАВИТЬ$(s, \mathscr{L})$. Вставить отрезок $s$ в полное упорядочение, представленное в $\mathscr{L}$.

b. УДАЛИТЬ$(s, \mathscr{L})$. Удалить отрезок $s$ из $\mathscr{L}$.

c. НАД$(s, \mathscr{L})$. Найти имя отрезка, расположенного непосредственно над $s$ в $\mathscr{L}$.

d. ПОД$(s, \mathscr{L})$. Найти имя отрезка, расположенного непосредственно под $s$ в $\mathscr{L}$.

Подобная структура данных известна как *словарь* (см. разд. 1.2.3), а все вышеописанные операции можно реализовать

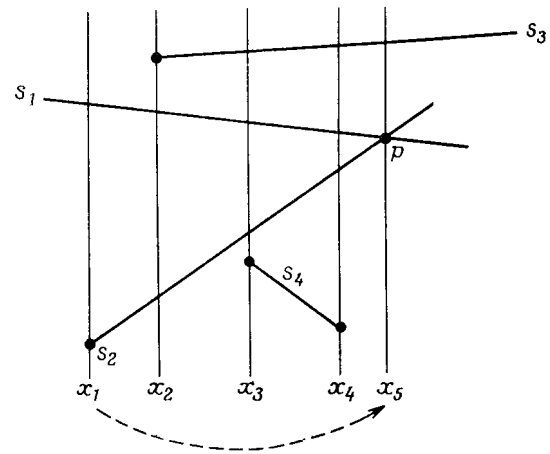



Рис. 7.16. Точка пересечения $p$ обнаружена при $x = x_1$, когда отрезки $s_1$ и $s_2$ впервые стали смежными. Однако события с абсциссами $x_2$, $x_3$ и $x_4$ необходимо обработать раньше, чем точку $p$ с абсциссой $x_5$.

за время, логарифмически связанное с его размером. На практике использование прошитого словаря (с прямым употреблением указателей, если адрес $s$ известен) позволяет реализовать функции НАД$(s)$ и ПОД$(s)$ за константное время.

Относительно списка точек событий заметим, что для регистрации всех пересечений нужно уметь поддерживать отношение $>_x$ в течение всего процесса заметания плоскости. Как указано ранее, упорядочение $>_x$ изменяется только в некоторых абсциссах (см. случаи 1, 2 и 3 выше), которые являются концами отрезков или точками их пересечения. В то время как все концы отрезков заданы *заранее*, точка пересечения, найденная плоским заметанием, динамически порождает событие, которое, вообще говоря, должно запоминаться и обрабатываться алгоритмом в нужный момент. Заметим, что в действительности, возможно, придется обработать несколько других событий за время, прошедшее с момента обнаружения какой-нибудь

точки пересечения до момента ее обработки. На рис. 7.16 дан пример подобной ситуации: там точка пересечения $p$ обнаружена в момент $x_1$ (когда отрезки $s_1$ и $s_2$ стали смежными); однако необходимо обработать абсциссы $x_2$, $x_3$ и $x_4$ перед обработкой соответствующего $p$ события $x_5$. Следовательно, показано, что для отчета обо всех пересечениях структура данных $\mathscr{E}$, создаваемая для работы со списком точек событий, должна поддерживать следующие операции (в $\mathscr{E}$ хранится полное упорядочение точек событий):

a. MIN($\mathscr{E}$). Определить наименьший элемент в $\mathscr{E}$ и удалить его.

b. ВСТАВИТЬ($x, \mathscr{E}$). Вставить абсциссу $x$ в полное упорядочение, хранимое в $\mathscr{E}$.

В дополнение к этим важнейшим операциям необходимо, чтобы $\mathscr{E}$ поддерживала также операцию:

c. ЧЛЕН($x, \mathscr{E}$). Узнать, является ли абсцисса $x$ членом $\mathscr{E}$.

Указанная структура данных представляет собой *очередь с приоритетом* (см. разд. 1.2.3), и, как хорошо известно, она поддерживает все три описанные операции за время, логарифмически связанное с ее размером.

Теперь можно описать алгоритм: при заметании плоскости вертикалью в каждой точке события структура данных $\mathscr{L}$ корректируется и все пары отрезков, которые становятся смежными при этой корректировке, проверяются на пересечение. Если какое-нибудь пересечение обнаружено впервые, о нем дается отчет (вывод на печать), и его абсцисса вставляется в список точек событий $\mathscr{E}$. Более формально приведенные рассуждения выражены в процедуре (здесь очередь $\mathscr{A}$ — рабочий параметр процедуры) [Bentley, Ottmann (1979)]:

```
    procedure ПЕРЕСЕЧЕНИЕ-ОТРЕЗКОВ
1.  begin сортируем 2 N концов отрезков лексикографически
          по x и y и помещаем их в приоритетную
          очередь 𝓔:
2.    𝓐 := ∅;
3.    while (𝓔 ≠ ∅) do
4.      begin p := MIN (𝓔);
5.            if (p — левый конец) then
6.               begin s := отрезок, концом которого служит p;
7.                     ВСТАВИТЬ(s, 𝓛);
8.                     s1 := НАД(s, 𝓛);
9.                     s2 := ПОД(s, 𝓛);
10.                    if (s1 пересекает s) then 𝓐 ⇐ (s1, s);
11.                    if (s2 пересекает s) then 𝓐 ⇐ (s, s2)
               end;
```



```
12.           else if (p — правый конец) then
                    begin s := отрезок, концом которого
                          служит p;
13.                       s1 := НАД(s, 𝓛);
14.                       s2 := ПОД(s, 𝓛);
15.                       if (s1 пересекает s2 справа от p)
                          then 𝓐 ⇐ (s1, s2);
16.                       УДАЛИТЬ(s, 𝓛)
                    end
17.           else(* p — точка пересечения *)
18.                 begin(s1, s2) := отрезки, пересекающиеся
                    в p; (* причем s1 = НАД(s2) слева от p *)
19.                       s3 := НАД(s1, 𝓛);
20.                       s4 := ПОД(s2, 𝓛);
21.                       if(s3 пересекает s2) then 𝓐 ⇐ (s3, s2);
22.                       if (s1 пересекает s4) then 𝓐 ⇐ (s1s4);
23.                       поменять местами s1 и s2 в 𝓛
                    end
              (* теперь найденные пересечения следует обработать *)
24.           while (𝓐 ≠ ∅) do
25.             begin (s, s') ⇐ 𝓐;
26.                   x := общая абсцисса s и s';
27.                   if (ЧЛЕН(x, 𝓔) = ЛОЖЬ) then
28.                     begin вывод (s, s');
29.                           ВСТАВИТЬ(x, 𝓔)
                      end
              end
        end
    end.
```

При таком алгоритме не будет пропущено ни одного из пересечений, поскольку могут пересечься только смежные отрезки, причем все смежные пары корректно проверяются по крайней мере однажды. Более того, каждое пересечение регистрируется ровно один раз, поскольку, когда его абсцисса вставляется в $\mathscr{E}$, проверка в строке 27 предупреждает нежелательные повторы[1]).

С точки зрения оценки заметим: операция строки 1 (начальная сортировка) выполняется за время $O(N \log N)$. Блоки 6—11, 13—16 и 18—23 выполняются каждый за время $O(\log N)$, поскольку каждая операция на $\mathscr{L}$ укладывается в

[1]) Для обработки возможной ситуации, когда разные пересечения обладают одной и той же абсциссой, достаточно так строить $\mathscr{E}$, чтобы точки пересечения перечислялись в лексикографическом порядке пар $(x, y)$.

такую временну́ю оценку в худшем случае, а проверка попарного пересечения требует константного времени. Для каждого события, т. е. для каждого исполнения главного цикла *while* (строка 3), эти три блока являются взаимоисключающими. Если через $K$ обозначить число пересечений, встреченных алгоритмом, то главный цикл *while* выполняется ровно $(2N+K)$ раз. Единственное обстоятельство, которое все еще заслуживает пристального внимания, относится к структуре данных $\mathscr{E}$, а именно сколько раз будет выполняться проверка, помещенная в строке 27? Между прочим, заметим, что какое-нибудь одно пересечение можно повторно обнаруживать очень много раз; однако если соотнести каждую пересекающуюся пару с тем выполнением главного цикла *while*, на котором она обнаружена, то станет ясно, что общее число обнаруженных пересечений равно $O(N+K)$. Поэтому строка 27 выполняется $O(N+K)$ раз, а каждое выполнение требует $O(\log(N+K))=O(\log N)$ времени, поскольку $K=\binom{N}{2}=O(N^2)$. Следовательно, общая оценка времени, потребляемого главным циклом *while*, равна $O((N+K)\log N)$, что, очевидно, превосходит оценку шага начальной сортировки. Отсюда имеем следующую теорему:

**Теорема 7.7** [Bentley, Ottmann (1979)]. *На множестве из $N$ отрезков можно обнаружить $K$ пересечений за время $O((N+K)\log N)$.*

Заметим, что вышеописанный алгоритм решает следующую задачу (отличную от поставленной в начале данного раздела):

**Задача С.1.2** (ПЕРЕСЕЧЕНИЕ ОТРЕЗКОВ). Даны $N$ прямолинейных отрезков. Надо найти все их пересечения.

Наиболее важной чертой приведенного выше алгоритма является простота, однако, характеристики его работы не оптимальны. Действительно, нижняя оценка по времени для этой задачи равна $\Omega(K+N\log N)$ поскольку компонента $\Omega(N\log N)$ следует из теоремы 7.6, а $O(K)$ тривиально получается из оценки объема выводного файла. Понятно, что такое несовпадение оценок привлекло к себе большой интерес исследователей, который достиг кульминаций после серии последовательных улучшений построения алгоритма с оптимальной временной оценкой $\theta(K+N\log N)$. Это сделали Чазелле и Эдельсбруннер [Chazelle, Edelsbrunner (1988)]. Эти результаты будут далее проиллюстрированы в разд. 7.4.

Если отказаться от мысли найти все пересечения и заняться соответствующей задачей проверки («Найти одно пересечение, если их число больше нуля»), то все сильно упрощается. Фактически можно работать как в одномерном алгоритме. Однако в данном случае нельзя ограничиться единственным объектом в структуре данных: она должна быть пригодна для хранения полного упорядочения на множестве отрезков, пересеченных заметающей прямой, и совпадает с использованным ранее словарем $\mathscr{L}$. Главное различие алгоритмов построения и проверки пересечений заключается в структуре данных для списка точек событий, который в последнем случае является простым массивом, содержащим $2N$ исходных концевых точек. Имеем следующий алгоритм:

```
    procedure ПРОВЕРКА-ПЕРЕСЕЧЕНИЯ-ОТРЕЗКОВ
begin сортируем лексикографически 2N концов по x
    и y и помещаем их в действительный массив
    ТОЧКА[1 : 2N];
    for i := 1 until 2N do
      begin p := ТОЧКА[i];
            s := отрезок, концом которого является p;
            if (p — левый конец) then
               begin ВСТАВИТЬ(s, ℒ);
                     s1 := НАД(s, ℒ);
                     s2 := ПОД(s, ℒ);
                     if (s1 пересекает s) then вывод (s1, s)
                     if (s2 пересекает s) then вывод (s2, s)
               end;
            else (* p — правый конец s *)
               begin s1 := НАД(s, ℒ);
                     s2 := ПОД(s, ℒ);
                     if (s1 пересекает s2) then вывод (s1, s2);
                     УДАЛИТЬ(s, ℒ)
               end
      end
end.
```

Следующая теорема устанавливает корректность этого алгоритма:

**Теорема 7.8.** *Алгоритм* ПРОВЕРКА-ПЕРЕСЕЧЕНИЯ-ОТРЕЗКОВ *находит пересечение, если оно существует.*

*Доказательство.* Поскольку алгоритм сообщает о пересечении, только если оно действительно найдено, то он не объявит о ложном пересечении, и поэтому заслуживает внимания лишь тот случай, когда существующее пересечение может остаться необнаруженным. Покажем, что данный алгоритм корректно находит самую левую из существующих точек пере-

сечения — $q_L$. Действительно, если $q_L$ совпадает с левым концом $p$ какого-нибудь отрезка, то $q_L$ обнаруживается при обработке $p$. Поэтому предположим, что $q_L$ — не левый конец отрезка. Слева от $q_L$ структура данных $\mathscr{L}$ содержит корректное представление отношения $>_x$. Поскольку слева от $q_L$ существует некий небольшой непустой интервал абсцисс, на котором два отрезка, пересекающиеся в $q_L$, становятся смежными, то $q_L$ обнаруживается так, как и ожидалось. Теорема доказана.

Несмотря на то что вышеописанный алгоритм кажется простым, он обладает рядом любопытных свойств. Так, хотя самое левое пересечение действительно находится, однако это вовсе не означает, что оно найдено *первым*. (Читатель может проверить свое понимание этого алгоритма, точно указав, какое пересечение обнаруживается первым.) Поскольку этот алгоритм выполняет только $O(N)$ проверок пересечения, он также может не обнаружить некоторые пересечения. Главный блок *if* (проверка пересечений) выполняется, очевидно, за время $O(\log N)$ в худшем случае. Итак, получаем в заключение следующую теорему (повторим опять, что оптимальность следует из теоремы 7.6):

**Теорема 7.9.** *Факт пересечения какой-нибудь пары из $N$ отрезков на плоскости можно определить за оптимальное время* $\theta(N \log N)$.

Из этого результата немедленно вытекает

**Следствие 7.1.** *Следующие задачи можно решить за время $O(N \log N)$ в худшем случае*:

**Задача С.2.2** (ПРОВЕРКА ПЕРЕСЕЧЕНИЯ МНОГОУГОЛЬНИКОВ). Пересекаются ли два заданных многоугольника?

**Задача С.2.3** (ПРОВЕРКА ПРОСТОТЫ МНОГОУГОЛЬНИКА). Прост ли заданный многоугольник?

**Задача С.2.4** (ПРОВЕРКА УКЛАДКИ ГРАФА). Пересекаются ли ребра заданной прямолинейной укладки планарного графа?

Другие результаты, полученные с использованием вариантов процедуры ПРОВЕРКА-ПЕРЕСЕЧЕНИЯ-ОТРЕЗКОВ, можно найти в работе [Shamos, Hoey (1976)]. Процитируем оттуда

**Следствие 7.2.** *Факт пересечения какой-нибудь пары из $N$ окружностей можно определить за время $O(N \log N)$.*



### 7.2.4. Пересечение полуплоскостей

Построение общей области $N$ полуплоскостей эквивалентно поиску области решений множества из $N$ линейных неравенств (*ограничений*) типа

$$a_i x + b_i y + c_i \leqslant 0, \quad i = 1, 2, \ldots, N. \tag{7.2}$$

Любое решение (7.2) обычно называется *допустимым* решением, а все их множество называется *допустимой областью*. Иллюстрация данной задачи приведена на рис. 7.17.

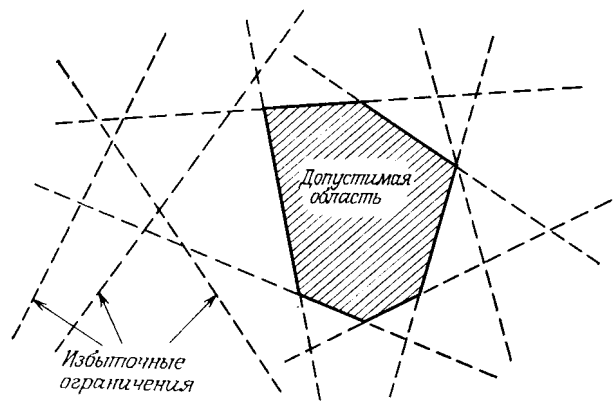



Рис. 7.17. Допустимая область множества линейных ограничений.

Существует простой квадратичный алгоритм для построения пересечения $N$ полуплоскостей. Предположим, что уже известно пересечение первых $i$ полуплоскостей. Это некая выпуклая многоугольная область, ограниченная не более чем $i$ сторонами, хотя и необязательно конечная. Пересечение этой области с очередной полуплоскостью есть не что иное, как ее разрезание прямой линией и сохранение правого или левого куска. Это можно проделать очевидным способом за время $O(i)$. Суммарная работа потребует $O(N^2)$ времени, однако преимущество данного алгоритма в том, что он открытый.

Рассмотрим вопрос о возможном улучшении алгоритма с помощью метода «разделяй и властвуй». Формальная постановка нашей задачи такова:

**Задача С.1.3** (ПЕРЕСЕЧЕНИЯ ПОЛУПЛОСКОСТЕЙ). Даны $N$ полуплоскостей $H_1, H_2, \ldots, H_N$. Нужно построить их пересечение: $H_1 \cap H_2 \cap \ldots \cap H_N$.

Поскольку оператор пересечения ассоциативен, то его компоненты можно произвольно разбить на части:

$$(H_1 \cap \ldots \cap H_{N/2}) \cap (H_{N/2+1} \cap \ldots \cap H_N). \tag{7.3}$$

Член в левых скобках является пересечением $N/2$ полуплоскостей и, следовательно, выпуклой многоугольной областью с не более чем $N/2$ сторонами. То же самое относится и к члену в правых скобках. Поскольку пересечение пары выпуклых многоугольных областей, содержащих по $k$ сторон каждая, можно построить за время $O(k)$ по теореме 7.3, то серединную операцию пересечения в формуле (7.3) можно проделать за время $O(N)$. Отсюда следует такой рекурсивный алгоритм:

**procedure** ПЕРЕСЕЧЕНИЕ ПОЛУПЛОСКОСТЕЙ

Вход: $N$ полуплоскостей, заданных ориентированными прямолинейными отрезками.

Выход: Их пересечение, выпуклая многоугольная область.

1. Разбиение заданных полуплоскостей на два подмножества приблизительно равной мощности.
2. Рекурсивное построение пересечения полуплоскостей для каждой из этих подзадач.
3. Слияние решений этих подзадач путем пересечения двух результирующих выпуклых многоугольных областей.

Если обозначить через $T(N)$ время, используемое этим алгоритмом для построения пересечения полуплоскостей, получаем

$$T(N) = 2T(N/2) + O(N) = O(N \log N). \tag{7.4}$$

Подведем итог:

**Теорема 7.10.** *Пересечение $N$ полуплоскостей можно построить за оптимальное время* $\theta(N \log N)$.

*Доказательство.* Верхняя оценка следует из уравнения (7.4). Для доказательства нижней оценки покажем, что

СОРТИРОВКА $\propto_N$ ПЕРЕСЕЧЕНИЕ ПОЛУПЛОСКОСТЕЙ.

Пусть заданы $N$ действительных чисел $x_1, \ldots, x_N$, и пусть $H_i$ — полуплоскость, содержащая начало координаты и определяемая касательной к параболе $y = x^2$ в точке с координатами $(x_i, x_i^2)$, т. е. прямой $y = 2x_i x - x_i^2$ (рис. 7.18). Пересечением таких полуплоскостей является выпуклая многоугольная область, последовательные ребра которой упорядочены по углу наклона. Если построить указанную область, то можно будет считывать заданные $\{x_i\}$ в отсортированном порядке. Теорема доказана.



Общий результат теоремы 7.10 имеет важное приложение:

**Следствие 7.3.** *Общую область $N$ выпуклых $k$-угольников можно построить за время $O(Nk \log N)$.*

*Доказательство.* Можно прямым методом получить оценку по времени $O(Nk \log Nk)$ путем построения пересечения $Nk$ левых полуплоскостей, ограничивающих исходные многоугольники. Для сокращения этой оценки будем обрабатывать заданные многоугольники как $N$ элементов, а не как набор из $Nk$ ребер. Обозначим через $T(N, k)$ время, необходимое для решения поставленной задачи. Пересечением $N/2$ штук $k$-угольников является выпуклый многоугольник с не более чем $Nk/2$ сторонами. Пересечение двух таких многоугольников можно построить за время $ckN$, где $c$ — константа. Поэтому рекурсивно подразбивая поставленную задачу, получаем, как и в случае задачи ПЕРЕСЕЧЕНИЕ ПОЛУПЛОСКОСТЕЙ:

$$T(N, k) = 2T(N/2, k) + ckN = O(Nk \log N),$$

что и требовалось доказать.

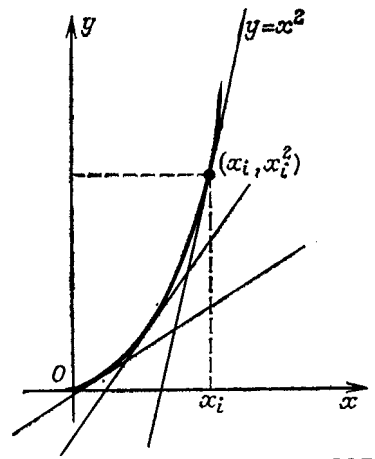



Рис. 7.18. Пример того, как СОРТИРОВКУ можно преобразовать в ПЕРЕСЕЧЕНИЕ ПОЛУПЛОСКОСТЕЙ.

### 7.2.5. Линейное программирование с двумя переменными

Постановка задачи (7.2) о пересечении $N$ полуплоскостей (обманчиво) напоминает стандартную постановку задачи линейного программирования [Gass (1969)] для двух переменных. В действительности же последняя задача формулируется так:

**Задача С.1.4** (ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ С ДВУМЯ ПЕРЕМЕННЫМИ). Нужно минимизировать функцию $ax + by$, при условии что

$$a_i x + b_i y + c_i \leqslant 0, \quad i = 1, \ldots, N. \tag{7.5}$$

По-прежнему допустимой областью задачи линейного программирования является множество точек $(x, y)$, удовлетворяющих ограничениям (7.5), которые определяют, очевидно,

пересечение $N$ полуплоскостей. Целевая функция определяет семейство параллельных прямых $ax + by + \lambda = 0$, где $\lambda$ — действительный параметр. Те прямые из этого семейства, которые являются опорными для допустимой области, проходят через ее вершины, доставляющие минимальное и максимальное значения целевой функции (рис. 7.19).

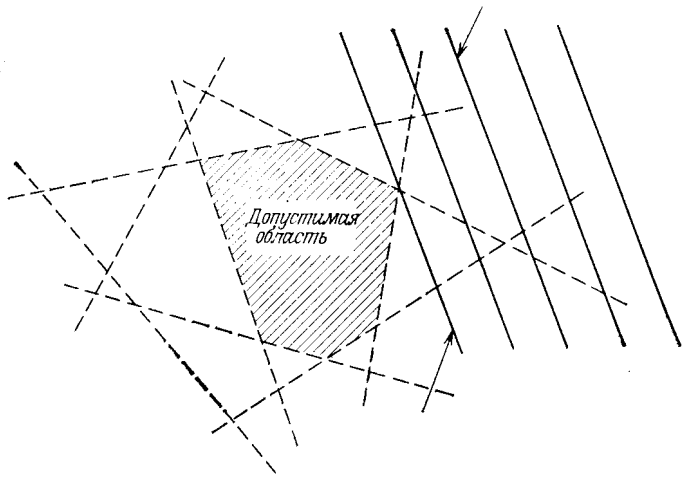



Рис. 7.19. Задача линейного программирования с двумя переменными. Верхняя стрелка указывает целевую функцию, заданную семейством параллельных прямых. Нижняя стрелка указывает минимизирующую вершину, определенную прямой, опорной к допустимой области.

Поскольку мы уже умеем определять прямые, опорные к выпуклому $N$-угольнику, за время $O(\log N)$ (см. разд. 3.3.6), то получаем преобразование

ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ С ДВУМЯ ПЕРЕМЕННЫМИ $\propto_N$ ПЕРЕСЕЧЕНИЕ ПОЛУПЛОСКОСТЕЙ,

и, используя результат предыдущего раздела (теорему 7.10), получаем следующую теорему:

**Теорема 7.11.** *Задачу линейного программирования с двумя переменными и $N$ ограничениями можно решить за время $O(N \log N)$. После того как она решена, экстремум новой целевой функции можно найти за время $O(\log N)$.*

Во-первых, сравним этот подход с симплекс-методом [Gass (1969)]. Если известно одно начальное допустимое решение, то симплекс-метод реализует движение от вершины к вершине по допустимой области, тратя $O(N)$ времени на



каждом шаге. Легко видеть, что в худшем случае в симплекс-методе придется посетить все вершины с затратой общего времени $O(N^2)$. (В этом отношении он очень похож на алгоритм Джарвиса из разд. 3.3.2.) Далее для максимизации новой целевой функции симплекс-метод должен проверить каждое ограничение, и поэтому он потратит $O(N)$ времени. Другими словами, симплекс-метод неоптимален.

Необходимо указать, что метод явного построения допустимого политопа нереалистичен для задачи линейного программирования при больших размерностях, поскольку число вершин может зависеть экспоненциально от числа размерностей (см. разд. 3.1).

Одна из поразительных черт алгоритма симплекс-метода состоит в том, что, хотя известна его экспоненциальная зависимость от числа измерений в худшем случае (и квадратичная оценка при двух переменных), его реальное поведение на практике почти всегда превосходно[1]). Аналогичное поведение демонстрирует и метод, основанный на пересечении полуплоскостей для широкого класса исходных данных [Bentley, Shamos (1978)]. Этот подход уже использовался нами, например, в разд. 4.1.1, а именно: если математическое ожидание размеров решений подзадач мало, то шаг слияния в алгоритме «разделяй и властвуй» можно реализовать за менее чем линейное время. Это будет иметь место, если многие из полуплоскостей *избыточные*, т. е. не образуют ребер допустимой области. А сейчас покажем, что для случайных исходных данных, как можно ожидать, большинство полуплоскостей будут избыточными.

Интуитивно понятно и практически легко представимо какое-либо естественное распределение вероятности положения случайных точек на плоскости; менее очевидно то, как можно моделировать случайный выбор полуплоскостей. Однако если прибегнуть к фундаментальному геометрическому преобразованию, введенному в разд. 1.3.3 и известному как *поляритет* (двойственность) (которое найдет себе важное применение в разд. 7.3), то можно сделать следующее наблюдение. Поляритет — это инволютивное преобразование «точка $\leftrightarrow$ прямая». С каждой прямой свяжем полуплоскость (содержащую начало координат). Предположим, что нам досталось множество $S$ из $N$ случайных точек, лежащих в какой-нибудь области (скажем, в единичном круге), и мы отображаем их в полуплоскости. Точки, лежащие на выпуклой оболочке $S$, перейдут в такое множество прямых, каждая из которых содержит ребро из общей области пересечения этих полуплоскостей. Поэтому, вспоминая результаты, процитированные в разд. 4.1.1, полу-

[1]) То есть полиномиально. — *Прим. перев.*

чим, что математическое ожидание числа *неизбыточных* полуплоскостей в множестве из $N$ полуплоскостей для данного случая равно $O(N^p)$, $p < 1$; похожие результаты справедливы и для других, достаточно естественных случайных моделей. Отсюда следует, что для задачи пересечения $N$ полуплоскостей получается линейная оценка среднего поведения алгоритма. А это немедленно ведет к оценке $O(N)$ для среднего поведения алгоритма решения задачи линейного программирования с двумя переменными. Мы видим, что математическое ожидание числа избыточных полуплоскостей — таких, которые не определяют ребер допустимой области, — очень велико, что может, в частности, объяснить прекрасное поведение симплекс-метода.

Однако нельзя успокаивать себя тем, что *математическое ожидание* временно́й оценки алгоритма построения допустимой области пропорционально $O(N)$, поскольку, как упоминалось в разд. 7.1.4, такое построение *не является необходимым* для решения задачи линейного программирования! Многие годы это наблюдение было бесплодным, до тех пор пока недавно чрезвычайно удачный метод, основанный на нем, не был независимо открыт Меджиддо [Megiddo (1983)] и Дайером [Dyer (1984)]. Этот метод (который легко адаптируется для случая трех измерений и был обобщен Меджиддо [Megiddo (1983)] на любое число измерений) отбрасывает не только избыточные ограничения (т. е. такие, которые не нужны и в задаче о пересечении полуплоскостей), но также и ограничения, гарантирующие отсутствие вершины, доставляющей экстремум целевой функции (называемой *оптимальной вершиной*). Этот метод основан на применении к точкам плоскости такого линейного преобразования, при котором целевая функция становится равной одной из двух координат, скажем ординате этой плоскости. После этого задача сводится к поиску экстремального значения некой кусочно-линейной выпуклой функции от абсциссы. Ключевой момент заключается в том, что поскольку требуется лишь определить экстремальное значение $x_0$, то нет нужды явно строить эту выпуклую функцию, которая неявно задана множеством линейных ограничений.

Более точно, исходную задачу [1])

$$\begin{cases} \text{минимизировать } ax + by \\ \text{при условиях } a_i x + b_i y + c_i \leqslant 0, \quad i = 1, 2, \ldots, N, \end{cases} \tag{7.6}$$

можно преобразовать, полагая $Y = ax + by$ и $X = x$, следующим образом (поскольку здесь $a$ и $b$ одновременно не равны

[1]) В данной формулировке предположим без потери общности, что целевую функцию нужно *минимизировать*.



нулю, предположим без потери общности, что $b \neq 0$):

$$\begin{cases} \text{минимизировать } Y \\ \text{при условиях } \alpha_i X + \beta_i Y + c_i \leqslant 0, \quad i = 1, 2, \ldots, N, \end{cases} \tag{7.7}$$

где $\alpha_i = (a_i - (a/b)b_i)$, а $\beta_i = b_i/b$. В новой задаче нужно вычислить наименьшее значение $Y$ на вершинах выпуклого многоугольника $P$ (допустимой области), определенного этими ограничениями (рис. 7.20). Чтобы избежать построения всей границы $P$, поступим следующим образом. В зависимости от того,

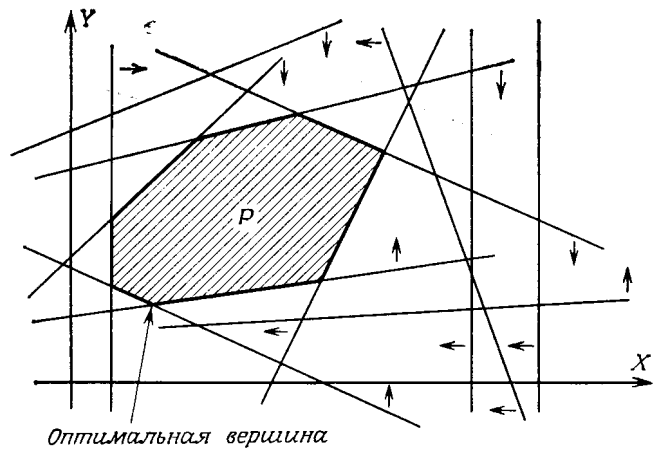



Рис. 7.20. После преобразования координат нужно найти наименьшую ординату в допустимой области.

будет ли $\beta_i$ нулем, отрицательным или положительным числом, разобьем множество индексов $\{1, \ldots, N\}$ на подмножества $I_0$, $I_-$, $I_+$ соответственно. Все ограничения с индексами из $I_0$ являются вертикальными прямыми (т. е. параллельными оси $Y$) и определяют допустимый интервал для $X$ следующим образом (рис. 7.21):

$$u_1 \leqslant X \leqslant u_2,$$
$$u_1 = \max\{-c_i/\alpha_i\colon i \in I_0\},$$
$$u_2 = \min\{-c_i/\alpha_i\colon i \in I_0\}.$$

С другой стороны, полагая $\delta_i \triangleq -(\alpha_i/\beta_i)$ и $\gamma_i \triangleq -(c_i/\beta_i)$, получаем, что все ограничения из $I_+$ имеют вид

$$Y \leqslant \delta_i X + \gamma_i, \quad i \in I_+,$$

так что они, вместе взятые, определяют кусочно-линейную, выпуклую вверх функцию $F_+(X)$ вида

$$F_+(X) \triangleq \min_{i \in I_+} (\delta_i X + \gamma_i).$$

Аналогично ограничения из $I_-$ в совокупности определяют кусочно-линейную, выпуклую вниз функцию $F_-(X)$ вида

$$F_-(X) \triangleq \max_{i \in I_-} (\delta_i X + \gamma_i).$$

Затем получаем преобразованное ограничение $F_-(X) \leqslant Y \leqslant F_+(X)$, а поскольку решается задача линейного программирования на минимум, то $F_-(X)$ и является нашей целевой функцией. Задача принимает следующий вид:

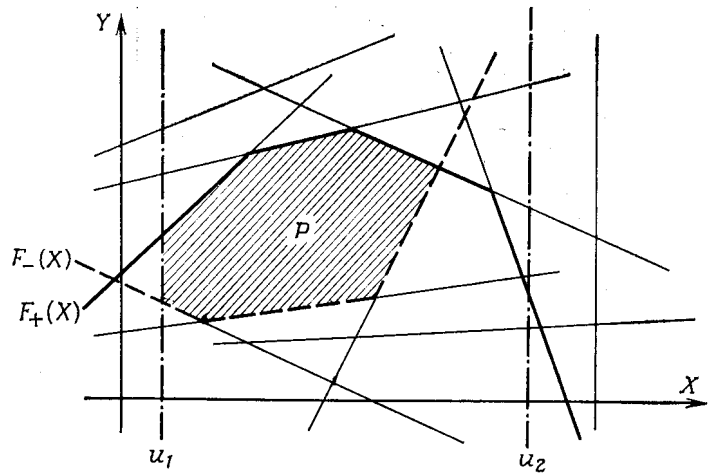



Рис. 7.21. Иллюстрация функций $F_-(X)$, $F_+(X)$ и величин $u_1$, $u_2$ при переформулировании задачи линейного программирования.

$$\begin{cases} \text{минимизировать } F_-(X) \\ \text{при условиях } F_-(X) \leqslant F_+(X), \\ \qquad\qquad u_1 \leqslant X \leqslant u_2. \end{cases}$$

Эта новая ситуация изображена на рис. 7.21, где показаны связи между $u_1$, $u_2$, $F_-(X)$, $F_+(X)$ и границей $P$.

Элементарной операцией, используемой в этом методе, является вычисление функций $F_-(X')$, $F_+(X')$ и их наклонов по обе стороны от заданного значения $X' \in X$. (Обозначим через $f_-^{(L)}(X')$ и $f_-^{(R)}(X')$ наклоны $F_-(X')$ слева и справа от $X'$ соответственно; аналогично определяются $f_+^{(L)}$ и $f_+^{(R)}$.) Покажем теперь, что эту элементарную операцию, названную *вычислением функций*, можно выполнить за время $O(N)$. Обращаясь для краткости лишь к $F_-(X)$, имеем выражение

$$F_-(X') = \max_{i \in I_-} (\delta_i X' + \gamma_i),$$

которое можно вычислить за время, пропорциональное $|I_-| = O(N)$. Если существует только одно значение $i$, равное $i_0$, на котором достигается $F_-(X')$, то $f_-^{(L)}(X') = f_-^{(R)}(X') = \delta_{i_0}$, иначе (если существуют два таких значения $i_1$ и $i_2$), то $f_-^{(L)}(X') = \min(\delta_{i_1}, \delta_{i_2})$ и $f_-^{(R)}(X') = \max(\delta_{i_1}, \delta_{i_2})$, поскольку $F_-$ выпукла вниз.



Справедлино утверждение, что для любого $X' \in [u_1, u_2]$ можно получить один из следующих результатов за время $O(N)$:

1. $X'$ недопустимо, а задача не имеет решений;
2. $X'$ недопустимо, но известно, по какую сторону от $X'$ (слева или справа) могут лежать все допустимые значения $X$;
3. $X'$ допустимо, и известно, по какую сторону от $X'$ лежит минимум $F_-(X)$;
4. $X'$ доставляет минимум функции $F_-(X)$.

Действительно, если функция $H(X) = F_-(X) - F_+(X)$ положительна в точке $X'$, то $X'$ недопустимо. Рассматривая наклоны $F_-(X)$ и $F_+(X)$ при $X = X'$, имеем (рис. 7.22): если $f_-^{(L)}(X') > f_+^{(L)}(X')$, то $H(X)$ возрастает в $X'$, и допустимые значения $X$ могут располагаться только слева от $X'$ (рис. 7.22(a), случай 2); если $f_-^{(R)}(X') < f_+^{(R)}(X')$, то аналогично допустимые $X$ могут быть только справа от $X'$ (рис. 7.22(b), случай 2); наконец, если $f_-^{(L)}(X') \leqslant f_+^{(L)}(X')$ и $f_-^{(R)}(X') \geqslant f_+^{(R)}(X')$, то $H(X)$ достигает минимума на $X'$ — задача неразрешима (рис. 7.22(c), случай 1).

Пусть теперь $H(X') \leqslant 0$, т. е. $X'$ допустимо. Оставляем в качестве упражнения вопрос о выборе между случаями 3 и 4 опять на базе информации о четырех наклонах: $f_-^{(L)}(X')$, $f_-^{(R)}(X')$, $f_+^{(L)}(X')$, $f_+^{(R)}(X')$.

Теперь стратегия решения начинает проясняться. Нужно пытаться так выбирать абсциссу $X'$, в которой производится вычисление, чтобы если алгоритм и не завершается сразу же, то по крайней мере *фиксированную долю* $\alpha$ от числа активных в данный момент ограничений можно было «вывести из игры» или отсечь (причем каждое отсекаемое ограничение, наверняка, не должно содержать крайних вершин). Если эта цель достигнута, то после $\log_{1/(1-\alpha)} N$ шагов мощность множества ограничений становится достаточно малой и приемлемой для прямого решения задачи. При таком предположении на $i$-м шаге число активных ограничений не превосходит $(1-\alpha)^{i-1}N$, и требуемая обработка завершится за время, не большее чем $K(1-$

$-\alpha)^{i-1}N$, где $K$ — некая константа. Поэтому суммарное время работы $T(N)$ оценивается сверху следующим образом:

$$T(N) \leqslant \sum_{i=1}^{\log_{1/(1-\alpha)} N} K(1-\alpha)^{i-1} N < \frac{KN}{\alpha},$$

т. е. оно линейно зависит от $N$, и это оптимально! Сейчас будет показано, что можно получить величину $\alpha = 1/4$.

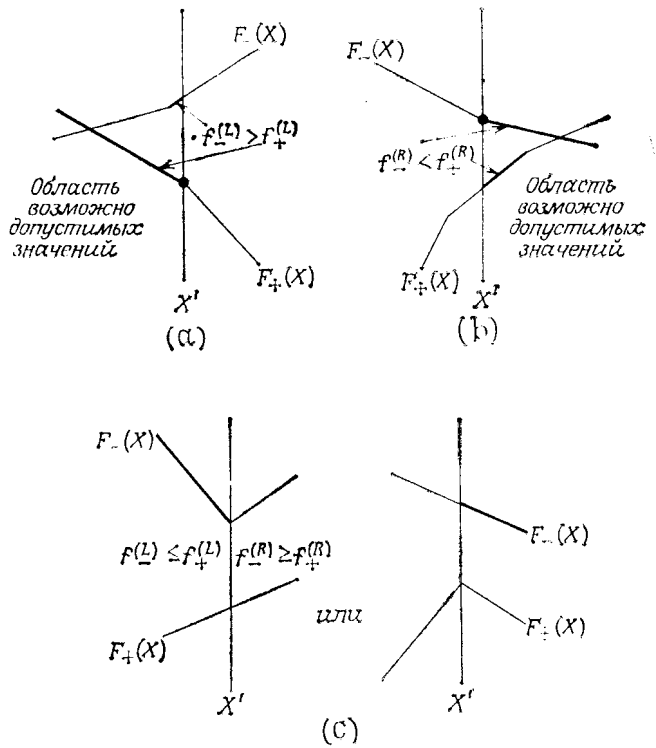



Рис. 7.22. Иллюстрация возможных случаев, при которых $F_-(X) > F_+(X)$.

На начальной стадии пусть $I_-$ и $I_+$ будут множествами индексов, определенных ранее, а $|I_+| + |I_-| = M$ (т. е. на этой стадии имеется $M$ активных ограничений, разбитых на два множества $S_+$ и $S_-$). Разобьем каждое из множеств $S_+$ и $S_-$ на пары ограничений, причем по одному ограничению в каждом множестве может остаться без пары. Допустим $i, j \in I_+$ и обратимся к рис. 7.23. Если $\delta_i = \delta_j$, то соответствующие прямые параллельны и одну из них можно сразу удалить (в данном случае ту, у которой больше значение $\gamma$) (рис. 7.23(a)). В противном случае, обозначив через $X_{ij}$ абсциссу точки пересечения ограничений, имеем следующие три случая: если $X_{ij} < u_1$, то удаляем ограничение с бо́льшим значением $\delta$ (рис. 7.23(b)); если $X_{ij} > u_2$, то удаляем ограничение с меньшим значением $\delta$ (рис. 7.23(c)); если же $u_1 \leqslant X_{ij} \leqslant u_2$, то ничего не удаляем



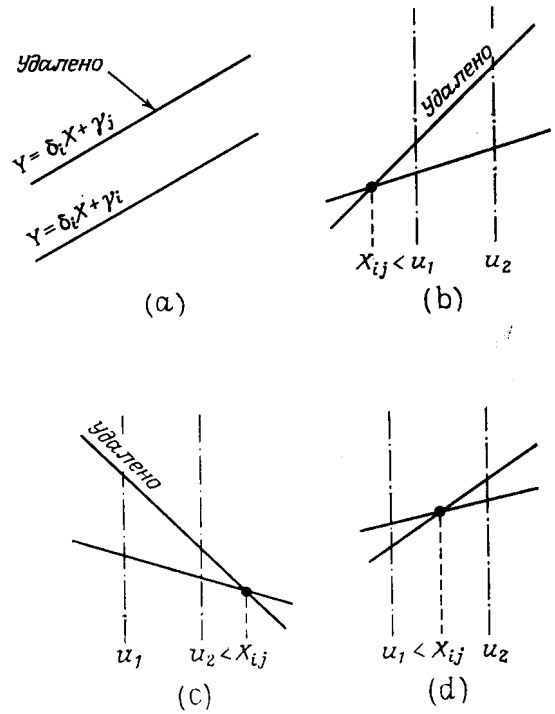



Рис. 7.23. Для каждой пары ограничений находим, к какому из этих четырех случаев она относится, после чего одно из ограничений удаляется.

(рис. 7.23(d)). Затем точно так же обрабатывается множество $S_-$. Для тех пар, ни один из членов которых не был удален, вычисляем абсциссу точки их пересечения.

Итак, если $k$ ограничений были удалены сразу, то получаем множество из $\lfloor (M-k)/2 \rfloor$ абсцисс пересечений, причем тратим на это $O(M)$ времени. Затем применяем к этому множеству абсцисс алгоритм поиска медианы [Blum, Floyd, Pratt, Rivest, Tarjan (1973); Schönhage, Paterson, Pippenger (1976)], обладающий линейной оценкой по времени, и получаем его медиану $\bar{X}$. Значение $\bar{X}$ является абсциссой, для которой проводится очередное «вычисление»; очевидно, что эта операция займет $O(M)$ времени. Если в точке $\bar{X}$ нет экстремума, то поло-

вина из вычисленных абсцисс $\{X_{ij}\}$ лежит в той области, о которой известно, что в ней нет экстремума. Для каждого $X_{ij}$ из этой области можно удалить одно ограничение из пары непосредственно. (Например, если $X_{ij}$ находится слева от $\bar{X}$, а оптимум — справа, то из пары прямых, пересекающихся в $X_{ij}$, удаляем ту, у которой меньше наклон, и т. д.) Завершаем шаг алгоритма с тем результатом, что удалено не менее $k + \lceil \lfloor (M-k)/2 \rfloor /2 \rceil \geqslant \lfloor M/4 \rfloor$ ограничений. Это подтверждает получение ранее обещанного значения $\alpha = 1/4$, и мы имеем следующий удивительный результат:

**Теорема 7.12.** *Задачу линейного программирования с двумя переменными и $N$ ограничениями можно решить за оптимальное время* $\theta(N)$.

Как упоминалось ранее, этот метод можно модифицировать для соответствующей трехмерной задачи, опять получая алгоритм с оптимальной оценкой по времени $\theta(N)$. Читатель отсылается к ранее цитированным статьям Меджиддо и Дайера для изучения деталей этого замечательного метода.

**Замечание 1.** Как показал Меджиддо, общий метод, описанный выше, можно применить к немного отличающимся ситуациям. Одним из таких примеров является вычисление наименьшей окружности, охватывающей множество $S = \{p_1, p_2, \ldots, p_N\}$, состоящее из $N$ точек, где $p_i = (x_i, y_i)$. Как показано в разд. 6.4, определение наименьшей охватывающей $S$ окружности означает вычисление

$$\min_{x,\,y} \max_{i=1}^{N} (x_i - x)^2 + (y_i - y)^2. \tag{7.8}$$

Если ввести переменную $z$, то выражение (7.8) можно переформулировать как следующую задачу математического программирования:

$$\begin{cases} \text{минимизировать } z \\ \text{при условиях } z \geqslant (x_i - x)^2 + (y_i - y)^2, \quad i = 1, \ldots, N. \end{cases} \tag{7.9}$$

(Заметим, что значение координаты $z$ в точке решения равно квадрату радиуса этой окружности.) Задача (7.9) не является задачей линейного программирования, поскольку ее ограничения являются квадратичными. Однако исходное ограничение можно переписать в виде

$$z \geqslant -2x_i x - 2y_i y + c_i + (x^2 + y^2), \tag{7.10}$$

где $c_i = x_i^2 + y_i^2$. Отсюда видно, что правая часть (7.10) состоит из линейного выражения $-2x_i x - 2y_i y + c_i$, зависящего от $i$, и из квадратичной составляющей $(x^2 + y^2)$, не зависящей от $i$.



Множество линейных ограничений

$$z \geqslant -2x_i x - 2y_i y + c_i, \quad i = 1, 2, \ldots, N, \tag{7.11}$$

определяет полиэдр, граница которого задается уравнением $z = f(x, y)$, где $f$ — выпуклая вниз функция. Поскольку $(x^2 + y^2)$ — тоже выпуклая вниз функция, то функция $\varphi(x, y) \triangleq f(x, y) + x^2 + y^2$ является суммой двух выпуклых вниз функций. В качестве простого упражнения читатель может проверить сходство поверхности $z = \varphi(x, y)$ с полиэдральной поверхностью в том смысле, что она имеет вершины, образованные пересечением трех [1]) дуг парабол (ребер), лежащих в вертикальных плоскостях, и что ее гранями являются участки поверхностей параболоидов. Двумерной иллюстрацией служит рис. 7.24. Простой анализ позволяет убедиться, что описанный выше метод удаления ограничений полностью применим также к данной ситуации. Единственное существенное различие заключается в том, что вершина $w$, доставляющая минимальное значение $z$ (среди других вершин), может не совпадать с искомым минимумом (хотя в случае линейного программирования эти минимумы всегда совпадают). Поэтому поиск завершается проверкой тех «ребер» функции $z = \varphi(x, y)$, которые инцидентны вершине $w$, и, если это необходимо, локализацией минимума на одном из этих ребер.

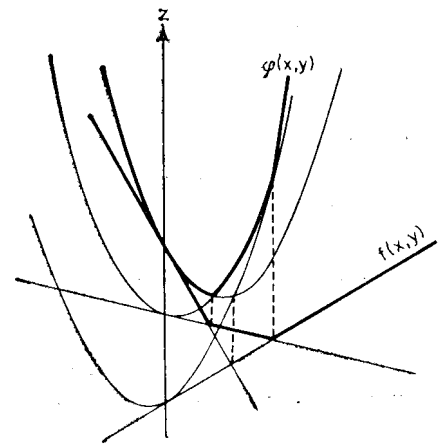



Рис. 7.24. Иллюстрация плоского сечения поверхностей $z = f(x, y)$ и $z = \varphi(x, y) = f(x, y) + (x^2 + y^2)$.

Следовательно, наименьшую охватывающую окружность можно получить в результате решения варианта трехмерной задачи линейного программирования. Комбинируя этот результат с аналогом теоремы 7.12 для трех измерений, получаем следующее усиление теоремы 6.14:

**Следствие 7.4.** *Наименьшую окружность, охватывающую $N$-точечное множество, можно вычислить за оптимальное время* $\theta(N)$.

[1]) Здесь для простоты исключается возможность существования четырех точек из $S$, лежащих на одной окружности.

**Замечание 2.** Особенно интересным приложением линейного программирования является задача о линейной разделимости (рассматриваемая здесь в $E^d$).

**Определение 7.2.** Говорят, что два точечных множества $S_1$ и $S_2$ в $E^d$ являются *линейно разделимыми*, если существует такое линейное $(d-1)$-мерное многообразие $\pi$, что $S_1$ и $S_2$ лежат по разные его стороны. (Для двух измерений $\pi$ — прямая, а для трех — обычная плоскость.)

Покажем теперь, что линейная разделимость является задачей линейного программирования. Пусть даны два множества точек в $d$-мерном пространстве: $S_1=\{(x_1^{(i)}, \ldots, x_d^{(i)}): i=1, \ldots, |S_1|\}$ и $S_2=\{(x_1^{(j)}, \ldots, x_d^{(j)}): j=|S_1|+1, \ldots, |S_1|+|S_2|\}$, где $|S_1|+|S_2|=N$; ищем одну из плоскостей $p_1x_1+\ldots+p_dx_d+p_{d+1}=0$, удовлетворяющих условиям:

$$\begin{cases} p_1a_1^{(i)}+\ldots+p_da_d^{(i)}+p_{d+1}\leqslant 0 & \text{при } 1\leqslant i\leqslant |S_1|, \\ p_1a_1^{(i)}+\ldots+p_da_d^{(i)}+p_{d+1}\geqslant 0 & \text{при } |S_1|+1\leqslant i\leqslant N. \end{cases}$$

Очевидно, что эта задача является задачей линейного программирования, и, согласно предыдущим результатам, ее можно решить за время $O(N)$ при двух и трех измерениях.

### 7.2.6. Ядро плоского многоугольника

В разд. 2.2.1 было показано, что поиск точки в ядре звездного многоугольника является важным шагом при предобработке, необходимой для ответа на соответствующий вопрос о принадлежности. Тогда мы отложили разработку алгоритма построения ядра до тех пор, пока не появятся необходимые для этого средства. Задача ставится следующим образом:

**Задача С.1.5** (ЯДРО МНОГОУГОЛЬНИКА). Дан $N$-вершинный (не обязательно простой) многоугольник на плоскости. Надо построить его ядро.

Во-первых, заметим, что ядром является пересечение $N$ полуплоскостей. Действительно, каждое ребро многоугольника $P$ определяет одну полуплоскость, в которой должно лежать это ядро (рис. 7.25). Такие полуплоскости называются внутренними (или левыми полуплоскостями сообразно с обходом границы ядра против часовой стрелки). Известно [Yaglom, Boltyanskii (1961)], что ядро многоугольника является пересечением его левых полуплоскостей. Поэтому непосредственно получаем, что

ЯДРО МНОГОУГОЛЬНИКА $\propto_N$ ПЕРЕСЕЧЕНИЕ ПОЛУПЛОСКОСТЕЙ,

и, используя результат разд. 7.2.4 (теорему 7.10), имеем

**Следствие 7.5.** *Ядро произвольного $N$-угольника можно построить за время $O(N\log N)$.*

Заметим, однако, что нижняя оценка $\Omega(N\log N)$, доказанная в теореме 7.10, неприменима к задаче о построении ядра, поскольку ребра простого многоугольника не могут занимать

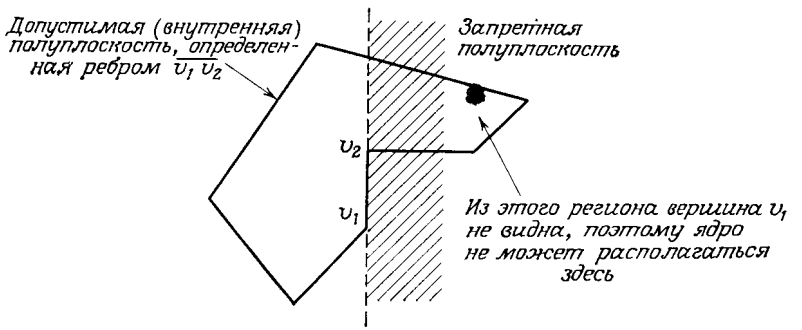



Рис. 7.25. Каждое ребро многоугольника $P$ определяет допустимую полуплоскость.

произвольных положений, а значит, нет сводимости к задаче о сортировке. Другими словами, нет оснований считать, что необходимо затратить более чем линейное время для построения ядра; более того, сейчас будет показано, что такой алгоритм существует [Lee, Preparata (1978)].

Входной многоугольник $P$ представлен последовательностью вершин $(v_0, v_1, \ldots, v_{N-1})$, где $N\geqslant 4$, а $e_i=\overline{v_{i-1}v_i}$ при $i=1, 2, \ldots, N-1$ есть ребро, связывающее вершины $v_{i-1}$ и $v_i$ ($e_0=\overline{v_{N-1}v_0}$). Для простоты ссылок опишем $P$ циклическим списком чередующихся вершин и ребер $v_0e_1v_1e_2\ldots e_{N-1}v_{N-1}e_0v_0$. Предположим также, что граница $P$ ориентирована против часовой стрелки, т. е. внутренность $P$ лежит слева от каждого из ребер. Будем говорить, что вершина $v_i$ *вогнута*, если внутренний угол при $v_i$ больше 180°, и *выпукла* в противном случае; без потери общности предположим, что нет внутренних углов, равных 180°. Если $K(P)$ (ядро $P$) не пусто, то результатом работы также станет последовательность вершин и ребер $K(P)$.

Алгоритм просматривает вершины $P$ по порядку и строит последовательность выпуклых многоугольников $K_1, K_2, \ldots, K_{N-1}$. Каждый из этих многоугольников может быть либо ограниченным, либо нет. Позднее будет показано (лемма 7.1), что $K_i$ является общим пересечением полуплоскостей, лежащих слева от ориентированных ребер $e_0, e_1, \ldots, e_i$. Отсюда, очевидно, следует, что $K_{N-1} = K(P)$ и что $K_1 \supseteq K_2 \supseteq \ldots \supseteq K_i$; последнее означает, что существует такое $r_0 > 1$, что $K_i$ будет неограниченным или ограниченным в зависимости от условия $i < r_0$ или $i \geqslant r_0$ соответственно.

Если точки $w_i$ и $w_{i+1}$ лежат на прямой, несущей $e_{s_i}$ — ребро $P$, то обозначим через $w_i e_{s_i} w_{i+1}$ отрезок этой прямой,

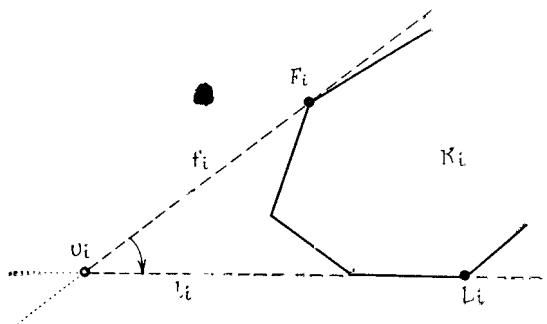



Рис. 7.26. Иллюстрация к определению вершин $F_i$ и $L_i$.

заключенный между $w_i$ и $w_{i+1}$ и ориентированный от $w_i$ к $w_{i+1}$. Если многоугольник $K_i$ не ограничен, то два его ребра — лучи; поэтому обозначим через $\Lambda e w$ луч, заканчивающийся в точке $w$ и ориентированный как ребро $e$, а через $weΛ$ луч, дополняющий первый до прямой.

В процессе работы алгоритма граница $K$ хранится в виде дважды связанного списка вершин и связывающих их ребер. Этот список будет линейным или циклическим в зависимости от того, является ли $K_i$ соответственно неограниченным или ограниченным. В первом случае первый и последний члены списка называются соответственно *головой* и *хвостом списка*.

Среди вершин $K_i$ выделим две — $F_i$ и $L_i$, определяемые следующим образом. Рассмотрим пару опорных к $K_i$ прямых, проходящих через вершину $v_i$ из $P$ (рис. 7.26). Пусть $f_i$ и $l_i$ — такие два луча на этих прямых, которые содержат опорные точки, и названные так, что угол, отсчитанный по часовой стрелке от $f_i$ до $l_i$ в клине плоскости, содержащем $K_i$, не превосходит $\pi$ (рис. 7.26). Обозначим через $F_i$ такую точку, общую для $f_i$ и $K_i$, которая наиболее удалена от $v_i$; аналогично определяется



и точка $L_i$. Эти две вершины играют важную роль при построении $K_{i+1}$ по $K_i$.

Если в $P$ нет вогнутых вершин, то $P$ выпуклый и, очевидно, $K(P) = P$. Поэтому пусть $v_0$ — вогнутая вершина $P$. Теперь можно описать алгоритм построения ядра.

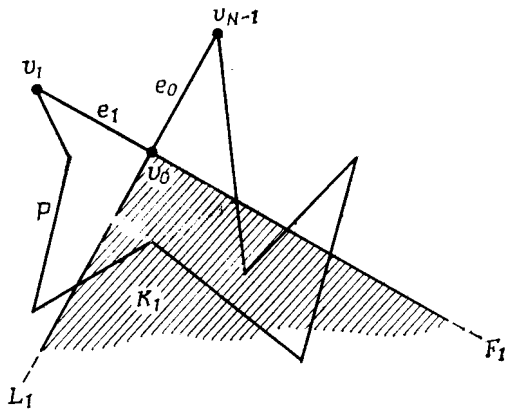



Рис. 7.27. Пример многоугольника $K_1$.

**Начальный шаг.** (* $K_1$ обозначает пересечение полуплоскостей, лежащих слева от ребер $e_0$ и $e_1$, см. рис. 7.27 *)

$K_1 :=$ $\Lambda e_1 v_0 e_0 \Lambda$;
$F_1 :=$ точка на бесконечности на луче $\Lambda e_1 v_0$;
$L_1 :=$ точка на бесконечности на луче $v_0 e_0 \Lambda$.

**Общий шаг.** (Переход от $K_i$ к $K_{i+1}$.) Предположим, что вершины $K_i$ занумерованы последовательно при обходе против часовой стрелки: $w_1, w_2, \ldots$. Будем различать следующие случаи:

(1) *Вершина $v_i$ вогнута* (рис. 7.28 и 7.29).

(1.1) *$F_i$ лежит на луче $\Lambda e_{i+1} v_{i+1}$ или справа от него* (рис. 7.28). Обходим границу $K_i$ *против часовой стрелки, начиная с $F_i$*, до тех пор, пока не достигнем единственного ребра $\overline{w_{t-1} w_t}$ из $K_i$, которое пересекает луч $\Lambda e_{i+1} v_{i+1}$, или пока не достигнем $L_i$, не найдя подобного ребра. В последнем случае закончим работу алгоритма ($K(P) = \varnothing$). В первом случае предпримем следующие действия:

(1.1.1) находим точку $w'$ пересечения $\overline{w_{t-1} w_t}$ с $\Lambda e_{i+1} v_{i+1}$;

(1.1.2) обходим границу $K_i$ *по часовой стрелке от $F_i$* до тех пор, пока не достигнем ребра $\overline{w_{s-1} w_s}$, пересекающего $\Lambda e_{i+1} v_{i+1}$ в точке $w''$ (это наверняка произойдет, если $K_i$ ограничен) или (если только $K_i$ неограничен) пока не достигнем головы списка, не

найдя подобного ребра. В первом случае, обозначая $K_i = \alpha w_s \ldots w_{t-1}\beta$ (где $\alpha$ и $\beta$ — последовательности чередующихся ребер и вершин), полагаем $K_{i+1} := \alpha w'' e_{i+1} w' \beta$; во втором случае (когда $K_i$ неограничен) необходимо проверить, будет $K_{i+1}$ ограниченным или нет. Если наклон луча $\Lambda e_{i+1} v_{i+1}$ заключен в пределах между наклонами начального и конечного лучей $K_i$, то $K_{i+1} := \Lambda e_{i+1} w' \beta$

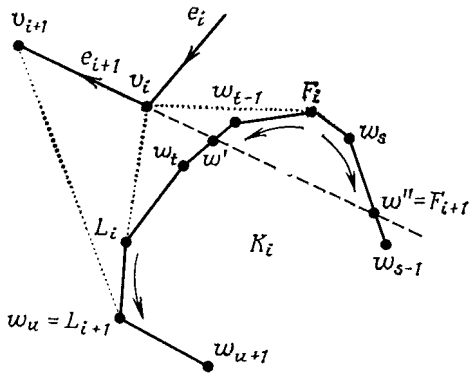



Рис. 7.28. Действия на общем шаге, если $v_i$ — вогнутая вершина, а $F_i$ лежит на луче $\Lambda e_{i+1} v_{i+1}$ или справа от него.

также неограничен. В противном случае начнем обход границы $K_i$ *по часовой стрелке от хвоста списка* до тех пор, пока не достигнем ребра $\overline{w_{r-1}w_r}$, которое пересекает луч $\Lambda e_{i+1} v_{i+1}$ в точке $w''$; обозначая $K_i = \gamma w_{t-1} \delta w_r \eta$, полагаем $K_{i+1} := \delta w'' e_{i+1} w'$, и список *становится циклическим.*

Выбор $F_{i+1}$ производится так: если $\Lambda e_{i+1} v_{i+1}$ имеет ровно одну точку пересечения с $K_i$, то $F_{i+1} :=$ (бесконечная точка луча $\Lambda e_{i+1} v_{i+1}$); иначе $F_{i+1} := w''$. Для определения $L_{i+1}$ обходим $K_i$ против часовой стрелки от $L_i$ до тех пор, пока или не найдена такая вершина $w_u$ на $K_i$, что $w_{u+1}$ лежит слева от луча $v_{i+1}(\overrightarrow{v_{i+1}w_u})\Lambda$, или не исчерпан весь список $K_i$, а подобная вершина не найдена. В первом случае $L_{i+1} := w_u$; в другом случае (который может встретиться, только если $K_i$ неограничен) $L_{i+1} := L_i$.

(1.2) *$F_i$ лежит слева от луча* $\Lambda e_{i+1} v_{i+1}$ (рис. 7.29). В этом случае $K_{i+1} := K_i$, но $F_i$ и $L_i$ нужно скорректировать. Для определения $F_{i+1}$ обходим $K_i$ *против часовой стрелки от* $F_i$ до тех пор, пока не найдена такая вершина $w_t$ на $K_i$, что $w_{t+1}$ лежит справа от луча $v_{i+1}(\overrightarrow{v_{i+1}w_t})\Lambda$; затем положим $F_{i+1} := w_t$. Определение точки $L_{i+1}$ проводится так же, как в случае (1.1).



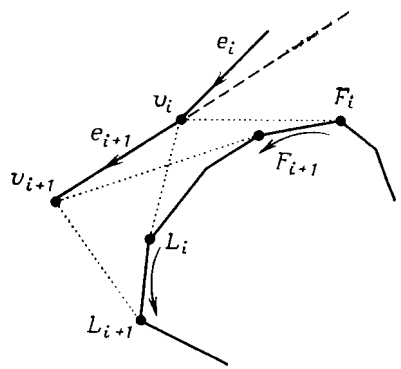



Рис. 7.29. Действия на общем шаге, если $v_i$ — вогнутая вершина, а $F_i$ лежит слева от луча $\Lambda e_{i+1} v_{i+1}$.

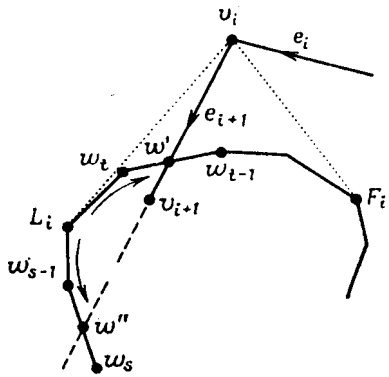



Рис. 7.30. Действия на общем шаге, если $v$ — выпуклая вершина, а $L_i$ лежит *на луче* $v_i e_{i+1} \Lambda$ или справа от него.

(2) *Вершина $v_i$ выпукла* (рис. 7.30 и 7.31).

(2.1) *$L_i$ лежит на луче $v_i e_{i+1} \Lambda$ или справа от него* (рис. 7.30). Обходим границу $K_i$ *по часовой стрелке от* $L_i$ до тех пор, пока не достигнем или того единственного ребра $\overline{w_{t-1}w_t}$,

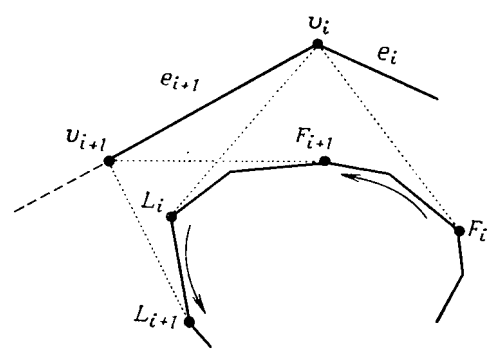



Рис. 7.31. Действия на общем шаге, если $v_i$ — выпуклая вершина, а $L_i$ лежит слева от луча $v_i e_{i+1} \Lambda$.

которое пересекает луч $v_i e_{i+1} \Lambda$, или вершины $F_i$, не обнаружив подобного ребра. В последнем случае заканчиваем алгоритм ($K(P) = \varnothing$). В первом случае предпримем следующие действия:

(2.1.1) находим точку $w'$ пересечения $\overline{w_{t-1}w_t}$ с $v_i e_{i+1} \Lambda$;

(2.1.2) обходим границу $K_i$ *против часовой стрелки от* $L_i$ до тех пор, пока не достигнем или единственного ребра $\overline{w_{s-1}w_s}$, пересекающего луч $v_ie_{i+1}\Lambda$ в точке $w''$ (что наверняка произойдет, если $K_i$ ограничен), или хвоста списка, не найдя подобного ребра (только в случае неограниченности $K_i$). Обозначая $K_i = \alpha w_t \ldots w_{s-1}\beta$, в первом случае полагаем $K_{i+1} := \alpha w' e_{i+1} w''\beta$; во втором случае (когда $K_i$ неограничен) нужно проверить, будет $K_{i+1}$ ограниченным или нет. Если наклон луча $v_ie_{i+1}\Lambda$ заключен в пределах между наклонами начального и конечного лучей $K_i$, то $K_{i+1} := \alpha w' e_{i+1}\Lambda$ также неограничен. В противном случае начнем обход границы $K_i$ *против часовой стрелки от головы списка* до тех пор, пока не найдено ребро $\overline{w_{r-1}w_r}$, которое пересекает луч $v_ie_{i+1}\Lambda$ в точке $w''$; обозначая $K_i := \gamma w_{r+1}\delta w_t\eta$, полагаем $K_{i+1} := \delta w' e_{i+1} w''$, и список *становится циклическим.*

Выборы точек $F_{i+1}$ и $L_{i+1}$ зависят от позиции $v_{i+1}$ на луче $v_ie_{i+1}\Lambda$ и от того, будет ли этот луч иметь одну или две точки пересечения с $K_i$. Рассмотрим эти два луча по отдельности:

(2.1.3) $v_ie_{i+1}\Lambda$ пересекает $K_i$ в точках $w'$ и $w''$. Если $v_{i+1} \in [v_ie_{i+1}w']$, то выбираем $F_{i+1}$ так же, как в случае (1.2). Иначе положим $F_{i+1} := w$. Если $v_{i+1} \in [v_ie_{i+1}w'']$, то положим $L_{i+1} := w''$. Иначе $L_{i+1}$ выбирается так же, как в случае (1.1), за исключением того, что $K_{i+1}$ *обходится против часовой стрелки от* $w''$.

(2.1.4) $v_ie_{i+1}\Lambda$ пересекает $K_i$ только в точке $w'$. Если $v_{i+1} \in [v_ie_{i+1}w']$, то выбираем $F_{i+1}$ так же, как в случае (1.2); иначе положим $F_{i+1} := w'$. В качестве $L_{i+1}$ выберем бесконечную точку на луче $v_ie_{i+1}\Lambda$.

(2.2) $L_i$ *лежит слева от луча* $v_ie_{i+1}\Lambda$ (рис. 7.31). В этом случае $K_{i+1} := K_i$. $F_{i+1}$ определяется так же, как в случае (1.2). Если $K_i$ ограничен, то $L_{i+1}$ определяется так же, как в случае (1.1); иначе $L_{i+1} := L_i$.

Корректность данного алгоритма доказывается в следующей лемме, где через $H_j$ обозначена полуплоскость, лежащая слева от прямой $\Lambda e_j\Lambda$.

**Лемма 7.1.** *Многоугольник* $K_{i+1}$ *является пересечением полуплоскостей* $H_0, H_1, \ldots, H_{i+1}$ *при* $i = 0, 1, \ldots, N-2$.



*Доказательство.* По индукции. Заметим, что $K_1$ по определению есть пересечение $H_0$ и $H_1$ (начальный шаг данного алгоритма). Предположим индуктивно, что $K_i = H_0 \cap H_1 \cap \ldots \cap H_i$. Поскольку во всех случаях, рассмотренных для общего шага, конструктивно строилось пересечение $K_i$ и $H_{i+1}$, то доказательство завершено.

Хотя лемма 7.1 гарантирует, что изложенный алгоритм корректно строит $K(P)$, необходимо проделать небольшую, но важную модификацию общего шага для повышения его эффективности. Фактически существуют случаи (рис. 7.32) многоугольников с пустым ядром, для которых можно затратить

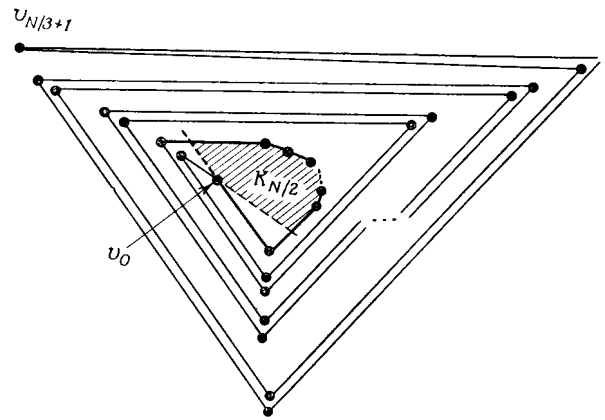



Рис. 7.32. Случай, когда немодифицированный алгоритм затрачивает $O(N^2)$ времени. $N$-сторонний многоугольник $K_{N/2}$ заштрихован. Затем алгоритм делает вокруг $K_{N/2}$ $N/12$ оборотов, прежде чем достигает вершины $V_{N/3+1}$, в которой выясняется, что $K(P) = \varnothing$.

$O(N^2)$ времени на всю работу алгоритма. Этого можно избежать путем проведения дополнительной проверки для определения ситуации, показанной на рис. 7.32. Здесь для краткости опущены детали этой проверки, а читатель отсылается к работе [Lee, Preparata (1978)], которая содержит полное доказательство. Достаточно сказать, что после проведения обсуждаемой модификации алгоритм гарантированно остановится после того, как он обойдет границу $P$, совершив поворот на угол $3\pi$ вокруг любой точки частичного ядра $K_i$.

Теперь оценим эффективность этого алгоритма. Удобно проанализировать порознь два основных типа действий, производимых алгоритмом построения ядра. Первое связано с корректировкой ядра путем пересечения $K_i$ с лучом $\Lambda e_{i+1}\Lambda$ для получения $K_{i+1}$; второе связано с корректировкой $F_i$ и $L_i$ и состоит

в обходах $K_i$ против часовой стрелки (или *прямых* обходах) для получения новых опорных вершин (однако заметим, что в ряде случаев, таких как (1.1) и (2.1), корректировка $K_i$ непосредственно приводит к корректировке той или иной опорной вершины).

Начнем с рассмотрения корректировок пересечений. В случае (1.1) (когда алгоритм не завершается) обходим $K_i$, начиная с $F_i$, как по часовой стрелке, так и против нее (при этом обходе ищется также $F_{i+1}$). Обозначим через $v_i$ общее число ребер, пройденных перед обнаружением двух точек пересечения $w'$ и $w''$. При этом фактически $v_i - 2$ ребер удаляются из $K_i$ (те, которые заключены между $w_s$ и $w_{t-1}$ на рис. 7.28), а поскольку все удаленные ребра коллинеарны различным ребрам $P$, то $\sum (v_i - 2) \leqslant N$. Итак, $\sum v_i$ — общее число вершин, пройденных алгоритмом при обработке случая (1.1), — ограничено сверху числом $3N$, т. е. равно $O(N)$. Такое же рассуждение с небольшими видоизменениями применимо к случаю (2.1).

Теперь рассмотрим те корректировки опорных вершин $F$ и $L$, которые явно выполняются в процессе построения пересечения. Такие корректировки проводятся для $L$ во всех случаях — (1.1), (1.2), (2.3), (2.4), а для $F$ в случаях (1.2) и (2.4). Заметим, что во всех этих случаях опорные вершины *продвигаются вперед* по границе $K_i$. Рассмотрим, например, корректировку $L$ в случае (1.1); другие случаи можно трактовать аналогично. Рассмотрим множество тех ребер $K_{i+1}$, которые пройдены алгоритмом перед определением $L_{i+1}$; посещение ребра, следующего непосредственно за $L_{i+1}$, будет называться *перелетом*. Очевидно, что при обработке случая (1.1) общее число перелетов равно $O(N)$, поскольку совершается не более одного перелета в расчете на вершину $P$. Теперь докажем, что если игнорировать перелеты, то любое ребро будет пройдено не более чем дважды. Предположим обратное, что при обработке $v_i$ одно из ребер было пройдено в третий раз. Поскольку обход всегда ведется в положительном направлении, отсюда следует, что граница $P$ обернулась вокруг $K_i$ по меньшей мере дважды, т. е. существует такая точка $q \in K_i$, вокруг которой граница при обходе повернулась на угол $\geqslant 4\pi$, а это противоречит ранее высказанному утверждению.

Поэтому оценка корректировки в случае (1.1), так же как и в случаях (1.2), (2.3) и (2.4), равна $O(N)$. Наконец, все корректировки $F$ и $L$ заканчиваются нахождением точек $w'$ и $w''$. Следовательно, время работы всего алгоритма пропорционально числу вершин $P$ и доказана

**Теорема 7.13.** *Ядро $N$-вершинного многоугольника можно вычислить за оптимальное время* $\theta(N)$.



## 7.3. Трехмерные приложения

### 7.3.1. Пересечение выпуклых полиэдров

Задача ставится следующим образом:

**Задача С.1.6** (ПЕРЕСЕЧЕНИЕ ВЫПУКЛЫХ ПОЛИЭДРОВ). Даны два выпуклых полиэдра — $P$ с $L$ вершинами и $Q$ с $M$ вершинами. Требуется построить их пересечение.

Для доказательства выпуклости полиэдра, образованного пересечением $P \cap Q$, используются те же аргументы, что и в теореме 7.2. Самый грубый подход к вычислению $P \cap Q$ состоит в проверке пересечения каждой грани $P$ с каждой гранью $Q$. Если такие пересечения существуют, то результат можно просто построить; если же попарных пересечений нет, то $P \cap Q$ пусто или один из полиэдров расположен внутри другого. Поскольку поверхность каждого полиэдра топологически является планарным графом, то из теоремы Эйлера о планарных графах следует, что изложенный подход может потребовать $O((L+M)^2)$ времени[1]).

Коль скоро этот подход отвергается как негодный, другой попыткой может стать адаптация к случаю трех измерений тех двух методов построения пересечения пары выпуклых многоугольников, которые были приведены в разд. 7.2.1. Метод Шеймоса — Хоуи [Shamos, Hoey (1976)] (разбиение плоскости и вычисление пересечения $P$ и $Q$ в каждой области этого разбиения) можно было бы обобщить, разрезая пространство семейством параллельных плоскостей, каждая из которых проходит через вершины $P \cup Q$. Однако в каждом из получающихся при этом слоев пространства (или «полосах», как их часто называют) соответствующая порция каждого полиэдра не является простым геометрическим объектом, как было в плоском случае. В действительности получается некий «блин», ограниченный с двух сторон плоскими многоугольниками, в каждом из которых может быть $O(|\text{число граней}|)$ ребер. Поэтому этот подход привел бы также к $O((L+M)^2)$-алгоритму. Что же касается метода [O'Rourke, Chien, Olson, Naddor (1982)], то обобщить его кажется невозможно.

Абсолютно иной подход, который, конечно, можно успешно приспособить и к случаю двух измерений, был предложен Маллером и Препаратой [Muller, Preparata (1978)] и обсуждается ниже. Альтернативный и очень привлекательный подход [Hertel, Melhorn, Mäntylä, Nievergelt (1985)], основанный на простран-

[1]) Из теоремы Эйлера следует лишь, что число граней $F$ не превосходит $O(N)$, где $N$ — число вершин графа. В самом деле, по формуле (1.2) $F \leqslant 2N - 4$. Отсюда следует, что более точная оценка времени грубого алгоритма равна $O(L \cdot M)$. — *Прим. перев.*

ственном заметании, был предложен совсем недавно, и он будет кратко рассмотрен в разд. 7.4.

Центральная идея метода Маллера — Препараты состоит в том, что если одна из точек пересечения — $p$ известна, то пересечение можно построить с использованием известной техники двойственности. Действительно, если точка $p$ из $P \cap Q$ существует, то простым переносом можно совместить с ней начало координат. Поэтому можно предположить, что начало пространственной системы координат лежит *внутри* как $P$, так и $Q$. Обозначая через $x_1$, $x_2$ и $x_3$ координаты в этом пространстве, свяжем с каждой гранью $f$ из $P$ или $Q$ некое полупространство:

$$n_1(f)\,x_1 + n_2(f)\,x_2 + n_3(f)\,x_3 \leqslant d(f). \tag{7.12}$$

Поскольку любая точка из $P \cap Q$ удовлетворяет всем неравенствам (7.12) для каждой грани каждого полиэдра, а начало координат $(0, 0, 0)$ является точкой из $P \cap Q$, то все неравенства можно нормировать так, что $d(f) = 1$. (Напомним, что по предположению начало координат лежит внутри как $P$, так и $Q$.)

Если теперь интерпретировать тройку $(n_1(f), n_2(f), n_3(f))$ как точку, то получится эффективно определенное инволютивное преобразование между плоскостями граней (т. е. плоскостями, которые несут грани) и точками. Это преобразование является обычным преобразованием двойственности (поляритетом) относительно единичной сферы с центром в начале координат, которое обсуждалось в разд. 1.3.3. При таком преобразовании точки на расстоянии $l$ от начала координат отображаются на плоскости на расстоянии $1/l$ от этого начала и *наоборот*.

Введем обозначение $\delta(\ )$ для этого преобразования, т. е. $\delta(p)$ обозначает плоскость, двойственную к точке $p$, а $\delta(\pi)$ — точку, двойственную к плоскости $\pi$.

Двойственное преобразование выпуклого многоугольника имеет смысл только тогда, когда он содержит внутри себя начало координат. Действительно, в этом случае имеется следующее важное свойство:

**Лемма 7.2.** *Пусть $P$ — выпуклый многоугольник, содержащий начало координат, а $S$ — множество точек, двойственных к плоскостям, несущим грани $P$. Тогда любая точка $p$, лежащая внутри $P$, отображается в такую плоскость $\delta(p)$, которая не пересекается с полиэдром* $\mathrm{conv}(S)$.

*Доказательство.* Пусть неравенство $a_i x_1 + b_i x_2 + c_i x_3 \leqslant 1$ определяет полупространство, ограниченное гранью $f_i$ из $P$, при $i = 1, \ldots, |S|$. (Напомним, что это полупространство содержит начало координат.) Теперь рассмотрим любую точку $\bar{p} = (\bar{x}_1, \bar{x}_2, \bar{x}_3)$, лежащую внутри $P$. Имеем

$$a_i \bar{x}_1 + b_i \bar{x}_2 + c_i \bar{x}_3 < 1, \quad i = 1, 2, \ldots, |S|.$$

Эти неравенства можно интерпретировать следующим образом. Плоскость $\delta(\bar{p})$, двойственная к $\bar{p}$, имеет уравнение $\bar{x}_1 x_1 + \bar{x}_2 x_2 + \bar{x}_3 x_3 = 1$; если взять точку $\delta(\pi_i)$, двойственную к плоскости $\pi_i$, несущей грань $f_i$, то данные неравенства означают, что все точки $\delta(\pi_i)$ при $i = 1, 2, \ldots, |S|$ лежат по одну сторону от плоскости $\delta(\bar{p})$ и не лежат на ней, т. е. $\delta(\bar{p})$ не имеет пересечений с $\mathrm{conv}(S)$. Что и требовалось доказать.

Полученный результат приводит к следующему утверждению:

**Теорема 7.14.** *Если $P$ — выпуклый полиэдр, содержащий начало координат, то таким же является и двойственный к нему полиэдр $P^{(\delta)}$.*

*Доказательство.* Пусть $\pi_i$ — плоскость, несущая грань $f_i$ из $P$. Определим точечное множество $S \triangleq \{\delta(\pi_i)\colon i = 1, \ldots, |S|\}$ и рассмотрим его выпуклую оболочку $\mathrm{CH}(S)$. Докажем, что каждая точка $S$ является вершиной $\mathrm{CH}(S)$. Действительно, если $\delta(\pi_i)$ лежит внутри $\mathrm{conv}(S)$, то $\pi_i$ не имеет пересечений с $P$ по лемме 7.2, а это противоречит тому, что $\pi_i$ несет грань $f_i$. Поэтому $\mathrm{CH}(S)$ — это двойственный к $P$ полиэдр $P^{(\delta)}$. Что и требовалось доказать.

Приведенное доказательство демонстрирует также тот факт, что если $P$ содержит начало координат, то любая внешняя к $P$ плоскость отображается в точку внутри $P^{(\delta)}$ и наоборот. Хотя эти соображения приложимы ко всем полиэдрам, удовлетворяющим предположениям теоремы 7.14, далее будем считать, что начало координат лежит строго внутри $P$ (и, следовательно, $P^{(\delta)}$). Читатель может проверить, что отсюда следует ограниченность как $P$, так и $P^{(\delta)}$.

Пусть $V_P^{(\delta)}$ и $V_Q^{(\delta)}$ обозначают множество вершин соответственно $P^{(\delta)}$ и $Q^{(\delta)}$. Если вспомнить, что каждый элемент $V_P^{(\delta)} \cup V_Q^{(\delta)}$ является двойственным к какой-нибудь из плоскостей, несущих грань одного из исходных полиэдров, и заметить, что $P \cap Q$ — это множество таких точек, которые одновременно лежат во всех полупространствах, определенных этими гранями и содержащими начало координат, то можно утверждать, что полиэдр $P \cap Q$ является двойственным к выпуклой оболочке $V_P^{(\delta)} \cup V_Q^{(\delta)}$. Значит, для вычисления $P \cap Q$ можно использовать алгоритм Препараты — Хонга (см. разд. 3.4.3), который определяет $\mathrm{conv}(V_P^{(\delta)} \cup V_Q^{(\delta)})$ за время $O(N \log N)$ (где $N \triangleq L + M$);

если затем взять фигуру, двойственную к этому результату, то получится искомый полиэдр. Двумерной иллюстрацией этого метода служит рис. 7.33.

Здесь необходимо сделать замечание. Изложенный метод включает обработку множества троек $\{(n_1(f), n_2(f), n_3(f)\colon f$ — грань $P$ или $Q\}$. Эти тройки интуитивно интерпретировались как точки, но с тем же успехом их можно было бы считать плоскостями. Единственной причиной предпочтения первой интерпретации является то, что получаемый при этом метод полностью

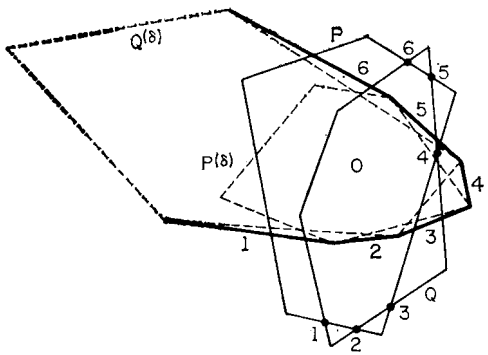



Рис. 7.33. Иллюстрация метода построения пересечения двух выпуклых многоугольников с использованием двойственного подхода. Два исходных многоугольника изображены тонкими сплошными линиями, а двойственные им фигуры (также выпуклые многоугольники) — штриховыми линиями. Опорные прямые выпуклой оболочки объединения последних фигур показаны жирными линиями, каждая из которых является двойственной к одной из вершин искомого пересечения (двойственные элементы помечены одинаково.

опирается на интуицию, а при второй интерпретации это менее заметно. Следовательно, двойственность поляритета не изменяет природы геометрических объектов (в отличие от преобразования инверсии — см. разд. 6.3.1.1), но существенно помогает нашей интуиции. Итак, если одна точка внутри $P \cap Q$ известна, то пересечение можно построить стандартными алгоритмами. Но как найти такую точку, если $P \cap Q$, вообще говоря, непусто? Поступим следующим образом.

Пусть дан выпуклый полиэдр $P$, рассмотрим множество его опорных плоскостей, параллельных оси $x_3$, которые для краткости назовем *вертикальными*. Пересечением $P$ с его вертикальными опорными плоскостями будет, вообще говоря, кольцеобразная область $R(P)$ на поверхности $P$, которая в невырожденных случаях сводится к кольцу его ребер. Проекцией $R(P)$ на плоскость $(x_1, x_2)$ является выпуклый многоугольник $P^*$ (рис. 7.34), совпадающий с выпуклой оболочкой проекций точек из $P$ на эту плоскость.



Область $R(P)$ легко вычисляется. Для любой грани $f_i$ из $P$ нормалью к $f_i$ является вектор $(n_1(f_i), n_2(f_i), n_3(f_i))$. Он перпендикулярен $f_i$ и направлен внутрь $P$. Пусть $f_i$ и $f_j$ — грани, смежные ребру $e$ из $P$. Тогда $e \in R(P)$ в том и только том случае, когда

$$n_3(f_i) \cdot n_3(f_j) \leqslant 0. \tag{7.13}$$

Следовательно, начнем с обхода ребер $P$ до тех пор, пока не обнаружим такое ребро $e$, которое принадлежит $R(P)$, проверяя условие (7.13). Затем выберем одну из двух вершин $e$, назвав ее $v$. Среди ребер, инцидентных $v$, есть одно или два новых

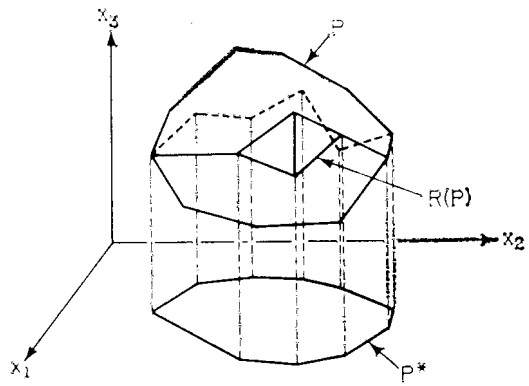



Рис. 7.34. Выпуклый полиэдр $P$, кольцеобразная область $R(P)$ и многоугольная проекция $P^*$.

ребра, отличных от $e$, которые принадлежат $R(P)$ и могут быть без труда найдены. (Предполагается, конечно, что поверхность $P$ — планарный граф — представлена структурой данных типа РСДС, которая была описана в разд. 1.2.3.2.) Поэтому можно продолжить построение $R(P)$, которое будет завершено при повторной встрече исходного ребра $e$. После вычисления $R(P)$ можно тривиально получить $P^*$. Итак, получены два первых шага алгоритма:

**Шаг 1.** Построить $P^*$ и $Q^*$. (Это займет $O(N)$ времени.)
**Шаг 2.** Построить пересечение многоугольников $P^*$ и $Q^*$ (используя любой из линейных по времени алгоритмов, описанных в разд. 7.1). Если $P^* \cap Q^* = \varnothing$, то останов, ибо $P \cap Q$ при этом также пуст. (Этот шаг займет $O(N)$ времени.)

Если $P^* \cap Q^* \neq \varnothing$, то обозначим через $\mathcal{D}$ замкнутую (выпуклую) область, ограниченную $P^* \cap Q^*$. Через каждую точку $p = (\bar{x}_1, \bar{x}_2)$ из $\mathcal{D}$ проведем вертикаль. Ее пересечением с $P$ является отрезок $e_P(p)$; аналогично определяется и $e_Q(p)$ (как

$e_P(p)$, так и $e_Q(p)$ могут выродиться каждый в единственную точку). Наша цель состоит в том, чтобы решить вопрос о существовании некой точки $p$ из $\mathcal{D}$, для которой $e_P(p)$ и $e_Q(p)$ перекрываются (рис. 7.35). Полагая без потери общности, что существует такая точка $u \in \mathcal{D}$, для которой низ отрезка $e_P(u)$ лежит выше верха отрезка $e_Q(u)$, определим *ближние стороны* $P$ и $Q$ как множества точек соответственно **низ**$[e_P(u)]$ и

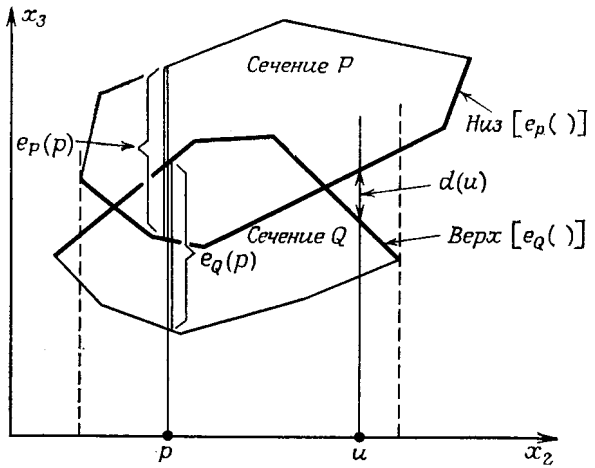



Рис. 7.35. Поперечное сечение $P$ и $Q$ плоскостью $x_1 = \bar{x}_1$. Иллюстрация к определению $e_P(\ )$, $e_Q(\ )$ и $d(\ )$.

**верх**$[e_Q(u)]$ и функцию $d(u)$, именуемую *вертикальным расстоянием*, следующим образом:

$$d(u) := \textbf{низ}\ [e_P(u)] - \textbf{верх}\ [e_Q(u)]. \tag{7.14}$$

Итак, следует решить вопрос о существовании такой точки $u \in \mathcal{D}$, чтобы $d(u) \leqslant 0$. Ответ на этот вопрос будет, разумеется, получен, если определить такую точку $\bar{u}$, в которой достигает минимума функция $d$:

$$d(\bar{u}) = \min_{u \in \mathcal{D}} d(u).$$

Очевидно, на практике поиск точки $u$ будет фактически прекращен только при условии, что $d(\bar{u}) > 0$, причем в этом случае два наших полиэдра не пересекаются. В противном случае этот поиск можно прекратить, как только будет обнаружена некая точка $v \in \mathcal{D}$, для которой $d(v) \leqslant 0$.

Метод решения данной задачи, который сейчас будет описан, принадлежит Дайеру [Dyer (1980)], а технически он значительно проще первоначального метода [Muller, Preparata (1978)] (хотя он и не демонстрирует сколько-нибудь значительного улучшения эффективности).



Прежде всего заметим, что функция $d(u)$ выпукла на $\mathcal{D}$. Действительно, пусть $S_P$ — такой участок поверхности $P$, на котором $x_3 = \textbf{низ}[e_P(u)]$ для $u \in \mathcal{D}$; аналогично определяется $S_Q$, где $x_3 = \textbf{верх}[e_Q(u)]$. Другими словами, $S_P$ и $S_Q$ — *противолежащие стороны* двух полиэдров. Если теперь спроецировать $S_P$ на плоскость $x_3 = 0$, то получится планарный граф $G_P$, состоящий из выпуклых ячеек; аналогично определяется $G_Q$.

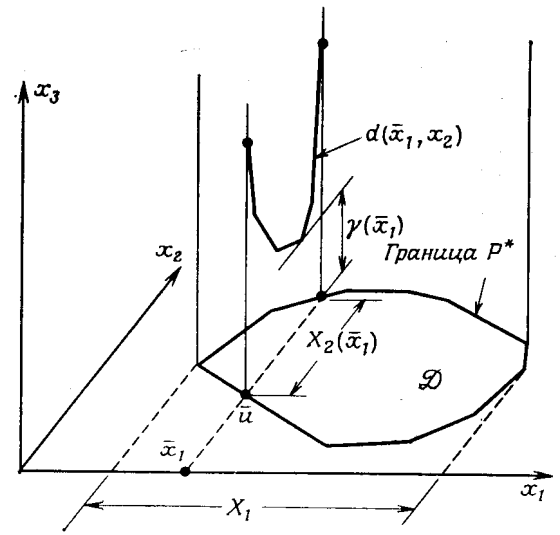



Рис. 7.36. Иллюстрация к определениям величин $X_1$, $X_2(\bar{x}_1)$ и $\gamma(x_1)$.

Представим себе наложение $G_P$ и $G_Q$. Результирующий граф $G^*$ является пересечением двух плоских карт, а его вершинами являются исходные вершины $G_P$ и $G_Q$ (условно именуемые *настоящими вершинами*) и точки пересечения ребер $G_P$ с ребрами $G_Q$ (условно именуемые *псевдовершинами*). Поэтому область $\mathcal{D}$ разбивается графом $G^*$ на множество выпуклых подобластей. Заметим, что внутри любой подобласти из $G_P$ функция **низ**$[e_P(u)]$ линейна по координатам $(x_1, x_2)$ точки $u$; точно так же, как и функция **верх**$[e_Q(u)]$ внутри любого региона $G_Q$. Поэтому внутри любого региона, образованного графом $G^*$ в $\mathcal{D}$, функция $d(u)$ линейна по $x_1(u)$ и $x_2(u)$. Более того, функция **низ**$[e_P(u)]$ выпукла вниз, а **верх**$[e_Q(u)]$ выпукла вверх; отсюда следует, что $d(u)$ является функцией выпуклой вниз. Значит, минимум $d(u)$ находится в одной из вершин $G^*$. Заметим, что число вершин графа $G^*$ может достигать $O(N^2)$:

фактически можно без труда построить два таких планарных графа, каждый из которых имеет по $v$ вершин, чтобы при их наложении получалось $(v-1)^2$ точек пересечения их ребер. Обратимся к рис. 7.36; пусть $X_1$ будет интервалом

$$[\min_{u \in \mathcal{D}} x_1(u), \max_{u \in \mathcal{D}} x_1(u)].$$

Пусть $X_2(\bar{x}_1)$ обозначает интервал

$$[\min_{x_1(u)=\bar{x}_1} x_2(u), \max_{x_1(u)=\bar{x}_1} x_2(u)] \quad \text{для} \quad \bar{x}_1 \in X_1$$

(т. е. отрезок, получающийся в сечении $\mathcal{D}$ прямой $x_1 = \bar{x}_1$). Поскольку $d(x_1, x_2) = d(u)$ выпукла в $\mathcal{D}$, то по хорошо известному результату из выпуклого анализа [Rockafellar (1970)] функция

$$\gamma(x_1) \triangleq \min_{x_2 \in X_2(x_1)} d(x_1, x_2) \quad \text{при} \quad x_1 \in X_1$$

будет выпуклой в $X_1$. Поскольку

$$\min_{x_1 \in X_1} \gamma(x_1) = \min_{u \in \mathcal{D}} d(u),$$

то задача свелась к оценке минимума $\gamma(x_1)$ на интервале $X_1$.

Прежде всего отметим, что вычисление $\gamma(\bar{x}_1)$ при заданном $\bar{x}_1$ выполнимо за время $O(N)$. Рассматривая только один из полиэдров, скажем $P$, определим прямым методом за время

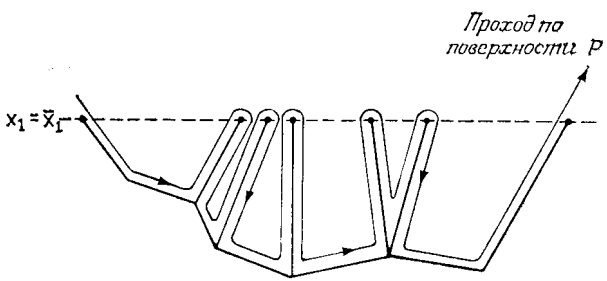



Рис. 7.37. Вычисление функции **низ**$[e_P(\ )]$ на многоугольнике, образованном в сечении $P$ плоскостью $x_1 = \bar{x}_1$, производится простым проходом по РСДС.

$O(L)$ значение функции **низ**$[e_P(\bar{u})]$ для точки $\bar{u}$ в плоскости $x_3 = 0$ на пересечении прямой $x_1 = \bar{x}_1$ с границей $P^*$. Затем простым проходом по поверхности $P$ с помощью РСДС можно определить также за время $O(L)$ значения функции **низ**$[e_P(\ )]$, соответствующие пересечениям каждого ребра из $P$ с плоскостью $x_1 = \bar{x}_1$ (рис. 7.37). Таким же способом можно обработать $Q$ за время $O(M)$. Функции **низ**$[e_P(v)]$ и **верх**$[e_Q(v)]$ для $x_1(v) = \bar{x}_1$ показаны на рис. 7.38. Значит, функция $d(v)$



при $v \in \mathcal{D}$ и $x_1(v) = \bar{x}_1$ вычисляется за время $O(L+M) = O(N)$, при этом также вычисляется ее минимум $\gamma(\bar{x}_1)$.

Следующим вопросом является вычисление экстремума унимодальной функции[1]) (заметим, что выпуклая функция унимодальна). Хорошо известен и прост тот факт, что экстремум унимодальной функции, определенной на $N$ дискретных точках действительной оси, можно определить методом модифицированного двоичного поиска, проведя $O(\log N)$ вычислений функции и $O(\log N)$ сравнений.

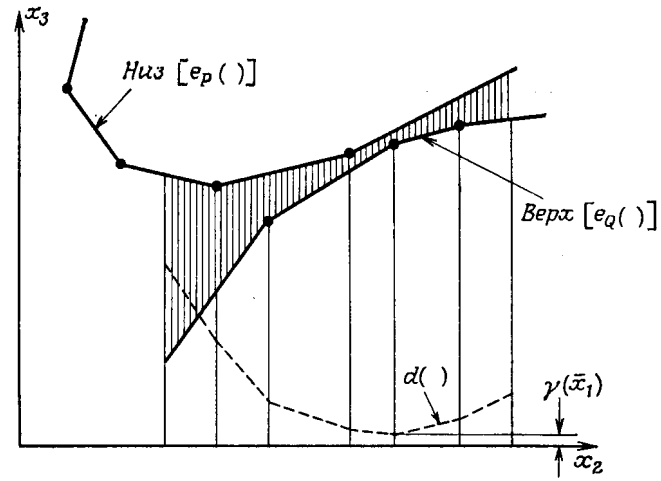



Рис. 7.38. Иллюстрация функций $d(v)$ и $\gamma(\bar{x}_1)$. Показано сечение $P$ и $Q$ плоскостью $x_1 = \bar{x}_1$.

Теперь можно сформулировать третий шаг поиска, предположив без потери общности, что $L \leqslant M$:

**Шаг 3.** Отсортировать множество $\{x_1\colon x_1 = x_1(v)$, где $v$ — вершина на нижней стороне $P\}$ в порядке возрастания; обозначим через $V$ упорядоченную последовательность. Поскольку $\gamma(x_1)$ унимодальна на последовательности $V$, то надо найти единственный отрезок $[x_1', x_1'']$, ограниченный точками из $V$, который гарантированно содержит минимум $\gamma(x_1)$ при $x \in X_1$. (Очевидно, что этот поиск прекратится, если будет найдено такое значение $\bar{x}_1$, при котором $\gamma(\bar{x}_1) = 0$.)

Чтобы проанализировать время работы шага 3, заметим прежде всего, что $O(L \log L)$ времени уйдет на начальную сортировку абсцисс. Затем для заданного $x_1$ вычисление $\gamma(x_1)$

[1]) Функция $f(x)$ называется *унимодальной вниз* на интервале $[x_1, x_2]$, если существует такое $x_0 \in [x_1, x_2]$, что $f(x)$ не возрастает на $[x_1, x_0]$ и не убывает на $[x_0, x_2]$. Аналогично определяется и *унимодальная вверх* функция.

можно выполнить за время $O(N)$ методом, который был описан ранее. Наконец, благодаря унимодальности $\gamma$ на $X_1$ методом модифицированного двоичного поиска получаем отрезок $[x_1', x_1'']$, вычислив $\gamma$ $O(\log L)$ раз. Итак, общее время работы составляет ровно $O(L\log L)+O(N\log L)=O(N\log L)$.

Заметим теперь, что по построению отрезок $[x_1', x_1'']$ не содержит внутри себя абсцисс вершин нижней стороны $P$. Поэтому полоса $x_1'\leqslant x_1\leqslant x_1''$ не содержит внутри себя ни одной вершины $G_P$, а, значит, множество $\mathscr{E}$ из ребер $G_P$, пересекающих эту полоосу, можно упорядочить (рис. 7.39). Рассмотрим

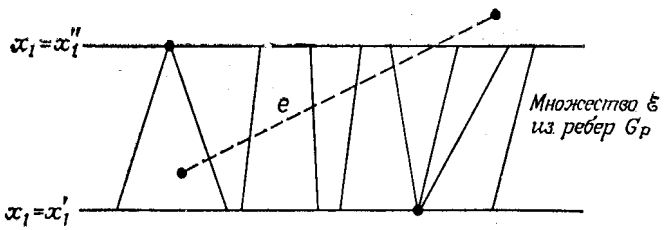



Рис. 7.39. Часть проекции нижней стороны $P$ на плоскость $x_3=0$ в пределах полосы $[x_1', x_1'']$. Внутри этой полосы нет вершин графа $G_P$.

такое ребро $e$ из $G_Q$, которое имеет непустое пересечение с полосой $x_1'\leqslant x_1\leqslant x_1''$. Функция $d(x_1, x_2)$ вдоль этого ребра выпукла вниз, а ее минимум совпадает с одной из вершин графа $G^*$ (эта вершина является либо одним из концов $e$, либо точкой пересечения $e$ с одним из ребер $\mathscr{E}$). Поскольку последние ребра упорядочены, то этот минимум можно определить весьма эффективно. Действительно, предположим, что элементы $\mathscr{E}$ организованы в дереве двоичного поиска элементарными операциями которого являются определение точки пересечения $v$ пары ребер и вычисление функции $d(v)$ (эти операции реализуются за время $O(1)$). Кроме того, для обработки одного ребра из $G_Q$ требуется $O(\log L)$ времени, что следует из предшествовавшего обсуждения свойств унимодальных функций. Итак, получен последний шаг процедуры поиска:

**Шаг 4.** Определить минимум функции $d(x_1, x_2)$ на каждом ребре $e$ из $G_Q$, которое пересекает полосу $x_1'\leqslant x_1\leqslant x_1''$. Наименьший из этих минимумов является глобальным минимумом $d$ на $\mathscr{D}$. (Вновь заметим, что поиск прекращается, если обнаружено такое ребро, на котором минимум $d$ неположителен.)

Поскольку число ребер, обработанных на шаге 4, не превосходит числа ребер в $Q$ (равного $O(M)$), а обработка одного ребра требует $O(\log L)$ времени, то шаг 4 завершается за время



$O(M\log L)$. Отмечая, что, поскольку и $L<N$, и $M<N$, постольку потребуется $O(N\log N)$ времени на обнаружение или такой точки $u$, для которой $d(u)\leqslant 0$, а следовательно, она принадлежит $P\cap Q$, или факта пустоты множества $P\cap Q$. Объединяя этот результат с ранее полученным результатом, который относился к построению $P\cap Q$ при условии, что одна из его точек известна, получаем следующую теорему:

**Теорема 7.15.** *Пусть даны два выпуклых полиэдра: $P$ с $L$ вершинами и $Q$ с $M$ вершинами; обозначим $M=L+M$. Для построения их пересечения $P\cap Q$ нужно затратить $O(N\log N)$ времени.*

К сожалению, единственная нижняя оценка, которую мы можем предложить для этой задачи, тривиальна — $\Omega(N)$. Поэтому неизвестно, является ли результат теоремы 7.15 оптимальным. На сегодняшний день небезосновательно предположение, что может существовать метод решения этой задачи, обладающий линейной оценкой по времени.

### 7.3.2. Пересечение полупространств

Самым естественным подходом к решению любой трехмерной задачи является попытка обобщения наиболее известного метода решения соответствующей двумерной задачи. В нашем случае двумерным аналогом является задача о пересечении полуплоскостей, решенная в разд. 7.2.4 методом «разделяй и властвуй», на шаге «слияние» которой решалась задача о пересечении двух выпуклых многоугольников. Соответственно в случае трех измерений шагом слияния становится пересечение двух выпуклых полиэдров, которое изучалось в предыдущем разделе. Однако если пересечение двух выпуклых многоугольников с $L$ и $M$ вершинами (где $L+M\triangleq N$) можно построить за время $\theta(N)$, то пересечение двух выпуклых полиэдров с аналогичными параметрами строится уже за время $O(N\log N)$. Поэтому обобщение двумерного алгоритма дало бы оценку $O(N\log^2 N)$. Противопоставление этого результата нижней оценке $\Omega(N\log N)$, полученной в разд. 7.2.4, заставляет полагать, что, прежде чем констатировать факт разрыва между оценками, который мы не можем заполнить, стоило бы попытаться поискать какой-нибудь другой метод решения.

И действительно, сейчас мы покажем, что существует оптимальный метод [Preparata, Muller (1979); Brown (1978)], который, не относясь к типу «разделяй и властвуй», использует алгоритмы как пересечения полиэдров, так и разделяющей плоскости (см. разд. 7.2.5). Как и описанный метод построения пересечения полиэдров, этот метод основан на активном использовании двойственного подхода по отношению к единичной

сфере. Необходимо указать, что для упрощения иллюстраций, а в некоторых случаях и просто для опоры на интуицию (иначе нам пришлось бы представлять себе четырехмерные объекты!) все наши примеры будут относиться к двумерным ситуациям. Однако важно подчеркнуть, что это не повлияет на справедливость доказательств.

Общая задача ставится следующим образом:

**Задача С.1.7** (ПЕРЕСЕЧЕНИЕ ПОЛУПРОСТРАНСТВ). Найти множество решений $(x, y, z)$ (*допустимых точек*) для множества из $N$ линейных неравенств:

$$a_i x + b_i y + c_i z + d_i \leqslant 0 \quad \text{при} \quad i = 1, 2, \ldots, N, \tag{7.15}$$

где $a_i$, $b_i$, $c_i$ и $d_i$ — действительные числа, не равные одновременно нулю.

Решение ищется в форме описания выпуклой полиэдральной поверхности, ограничивающей область допустимых точек. Качественно это решение может принимать одну из следующих форм (с увеличением размерности):

1. Пустое множество (уравнения (7.15) несовместны).
2. Точка.
3. Отрезок.
4. Выпуклый плоский многоугольник.
5. Выпуклый полиэдр.

Случаи 2, 3 и 4 вырождены; интересны лишь случаи 1 и 5.

Один простой случай получается, когда для всех $i = 1, \ldots, N$ выполняется $d_i \leqslant 0$. Действительно, при этом начало координат $(0, 0, 0)$ удовлетворяет всем ограничениям, т. е. оно содержится внутри общей области этих полупространств. Тогда, произведя двойственное преобразование плоскости $\pi_i$, заданной уравнением $a_i x + b_i y + c_i z + d_i = 0$, в точку $(a_i/d_i, b_i/d_i, c_i/d_i)$ и вычислив за время $O(N \log N)$ выпуклую оболочку полученного множества точек, мы построим выпуклый полиэдр, двойственным к которому будет искомое пересечение.

Ситуация становится значительно более сложной, когда $d_i$ произвольны. Для простоты предположим здесь, что никакие из $d_i$ не равны нулю, так что множество индексов $\{1, \ldots, N\}$ можно разбить на такие два подмножества $I_+$ и $I_-$, что $i \in I_+$, если $d_i > 0$, и $i \in I_-$ если $d_i < 0$. Предположим также без потери общности, что $d_i = \pm 1$ [1]).

Чтобы понять этот механизм, удобно рассмотреть нашу задачу в однородных координатах, осуществив преобразование

$$x = \frac{x_1}{x_4}; \quad y = \frac{x_2}{x_4}; \quad z = \frac{x_3}{x_4}. \tag{7.16}$$

[1]) Задача в общей постановке, допускающей $d_i = 0$, изложена в работе [Preparata, Muller (1979)].



Как было показано в разд. 1.3.2, в котором были введены однородные координаты, любую точку $(x, y, z)$ из $E^3$ можно интерпретировать как прямую $(cx, cy, cz, c)$ из $E^4$, где $c \in \mathbb{R}$. Если обозначить координаты в $E^4$ через $x_1$, $x_2$, $x_3$ и $x_4$, то в такой интерпретации обычное пространство $E^3$ становится гиперплоскостью $x_4 = 1$ в $E^4$ (см. рис. 7.40, показывающий аналогию с размерностью, на единицу меньшей). Точно так же если спроецировать гиперплоскость $x_4 = 1$ на единичную гиперсферу

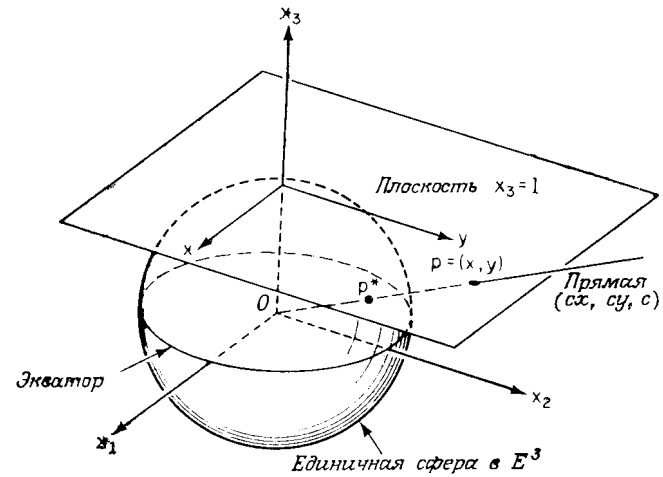



Рис. 7.40. Иллюстрация соответствия между обычными координатами в $E^2$ (на плоскости $x_3 = 1$) и однородными координатами в $E^3$. Каждая точка на $E^2$ соответствует прямой, проходящей через начало координат в $E^3$.

$S^4$, центр которой помещен в начало координат, то каждой точке $p$ из $E^3$ будет соответствовать единственная точка $p^*$ на $S^4$ (при $x_4 > 0$), получающаяся при пересечении $S^4$ с отрезком, соединяющим $p$ с началом координат $O$. Точки на $S^4$ с $x_4 > 0$ и $x_4 < 0$ образуют соответственно *положительную и отрицательную открытые полусферы*; те точки, для которых $x_4 = 0$, *экватор*, соответствуют бесконечно удаленной плоскости в $E^3$.

Преобразование координат (7.16) переводит каждое полупространство $a_i x + b_i y + c_i z + d_i \leqslant 0$ из $E^3$ в полупространство

$$a_i x_1 + b_i x_2 + c_i x_3 + d_i x_4 \leqslant 0$$

в $E^4$, для которого ограничивающая его гиперплоскость проходит через начало координат, а вектор $\mathbf{v}_i = (a_i, b_i, c_i, d_i)$ нормален к этой гиперплоскости и *направлен вне* этого полупространства. Указанная гиперплоскость пересекает $S^4$ по «большой окружности» $C_i$. Кроме того, ограничение $a_i x + b_i y +$

$+ c_i z + d_i \leqslant 0$ определяет единственную полусферу, ограниченную этой большой окружностью $C_i$. Эта полусфера содержит начало координат $E^3$ тогда и только тогда, когда $d_i = -1$.

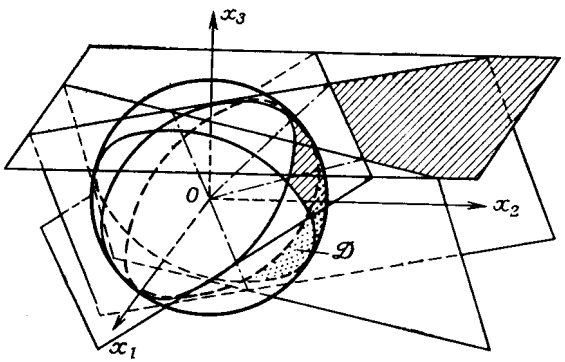



Рис. 7.41. Пересечением полусфер на $S^3$, которые соответствуют ограничениям (полуплоскостям на $E^2$), является связная область $D$ на поверхности $S^3$.

Поэтому для заданного множества из $N$ линейных ограничений в $E^3$ их общая область соответствует связной области $\mathscr{D}$ (возможно, пустой), лежащей на поверхности $S^4$ (рис. 7.41). Область $\mathscr{D}$ разбивается на две (возможно, пустые) связные области $\mathscr{D}_+$ и $\mathscr{D}_-$, лежащие на соответственно положительной

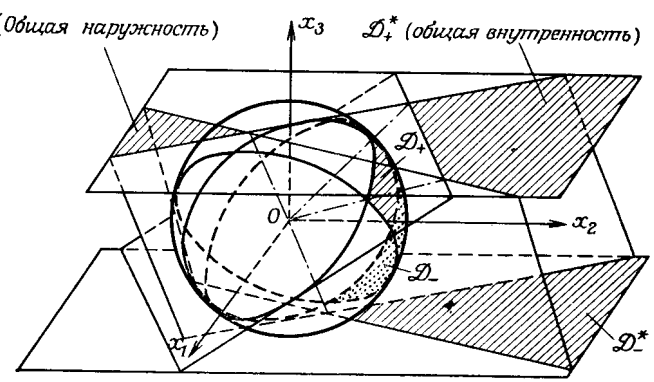



Рис. 7.42. Соотношения между областями $D_+$, $D_-$, а также общими внутренностью и внешностью (наружностью) для заданных ограничений.

и отрицательной полусферах (рис. 7.42). Теперь предположим, что реализуется проецирование $\mathscr{D}$ с центром в начале координат, причем $\mathscr{D}_+$ проецируется на выпуклую область $\mathscr{D}_+^*$, лежащую на гиперплоскости $x_4 = 1$ (т. е. в обычное пространство



$E^3$), а $\mathscr{D}_-$ проецируется на выпуклую область $\mathscr{D}_-^*$, лежащую на гиперплоскости $x_4 = -1$. Точки на $\mathscr{D}_+^*$ являются общим пересечением (или *общей внутренностью*) полупространств, определенных ограничениями. С другой стороны, точки на $\mathscr{D}_-^*$ имеют любопытную интерпретацию. Поскольку $x_4 = -1$, то $\mathscr{D}_-^*$ является множеством точек $(x_1, x_2, x_3, -1)$, для которых

$$a_i x_1 + b_i x_2 + c_i x_3 - d_i \leqslant 0 \quad \text{для} \quad i = 1, 2, \ldots, N.$$

Если теперь отобразить каждую такую точку, лежащую на гиперплоскости $x_4 = 1$, преобразованием симметрии по отношению к началу координат в $E^4$ (это преобразование заменяет $x_i$ на $-x_i$ для $i = 1, 2, 3$), то получится множество таких точек, для которых $a_i x_1 + b_i x_2 + c_i x_3 + d_i \geqslant 0$, $i = 1, \ldots, N$, или

$$a_i x + b_i y + c_i z + d_i \geqslant 0, \quad i = 1, 2, \ldots, N. \tag{7.17}$$

Если сравнить эти неравенства с ограничениями (7.15), то станет очевидно, что они определяют *общую внешность* заданных полупространств. Следовательно, на гиперплоскости $x_4 = 1$ объединенное множество точек может состоять из двух отдельных неограниченных выпуклых множеств, одно из которых будет общей внутренностью, а другое — общей внешностью заданных полупространств, как это показано для случая двух измерений на рис. 7.42. Назовем такую ситуацию *гиперболическим* случаем. Если все точки решения проецируются на положительную (или отрицательную ) открытую полусферу, то соответствующее множество из $E^3$ будет ограниченным, и такая ситуация называется *эллиптическим* случаем. *Параболический* случай имеет место тогда, когда существует единственное, но неограниченное множество в $E^3$, т. е. когда на $S^4$ имеются как экваториальные, так и положительные (или отрицательные) точки. Во всех трех случаях множества точек образуют то, что мы будем называть *обобщенным* выпуклым полиэдром. Только эллиптический случай соответствует обычному ограниченному полиэдру.

Нашей целью является совместное вычисление и общей внутренности, и общей внешности. Заметим, что область $\mathscr{D}$ всегда содержится в одной из полусфер на $S^4$, ограниченной гиперплоскостью $\pi$, заданной уравнением $p_1 x_1 + p_2 x_2 + p_3 x_3 + p_4 x_4 = 0$. Луч $(\rho p_1, \rho p_2, \rho p_3, \rho p_4)$ при $\rho \geqslant 0$, нормальный к $\pi$, протыкает $S^4$ в точке $u$. Если приложить к $E^4$ преобразование вращения $R$, которое переведет начало координат $E^3$ в точку $u$, то *вся* область $\mathscr{D}$ окажется в положительной полусфере. Ограничимся далее только эллиптическим случаем, который решается ранее описанным методом. Если бы был известен по-

лиэдр, образованный пересечением полупространств, то, применив к нему обратное преобразование вращения $R^{-1}$, мы перевели бы его опять в искомые общую внутренность и общую внешность. Поэтому цель состоит в поиске вектора $\mathbf{p} = (p_1, p_2, p_3, p_4)$.

Поскольку $\mathscr{D}$ содержится в полупространстве $p_1x_1 + p_2x_2 + p_3x_3 + p_4x_4 \geqslant 0$, то, полагая $\mathbf{v}_i = (a_i, b_i, c_i, d_i)$, получаем

$$\mathbf{p} \cdot \mathbf{v}_i^T \leqslant 0, \quad i = 1, 2, \ldots, N$$

(где «·» обозначает, как обычно, скалярное произведение). Это можно переписать так:

$$a_ip_1 + b_ip_2 + c_ip_3 + d_ip_4, \quad i = 1, 2, \ldots, N,$$

т. е.

$$\begin{cases} a_ip_1 + b_ip_2 + c_ip_3 + p_4 \leqslant 0 & \text{при} \quad d_i = +1, \\ -a_ip_1 - b_ip_2 - c_ip_3 + p_4 \geqslant 0 & \text{при} \quad d_i = -1. \end{cases}$$

Теперь дадим интерпретацию этим условиям. Пусть $p_1x + p_2y + p_3z + p_4 = 0$ обозначает плоскость в $E^3$. В зависимости от того, будет $d_i = +1$ или $d_i = -1$, точка $(a_i, b_i, c_i)$ или точка $(-a_i, -b_i, -c_i)$ будет двойственной к плоскости $a_ix + b_iy + c_iz + d_i = 0$ при обычном поляритете. Поэтому плоскость $a_ix + b_iy + c_iz + d_i = 0$ *разделяет* двойственные образы тех плоскостей, чьи индексы принадлежат соответственно $I_+$ и $I_-$.

Это немедленно приводит к методу поиска $\mathbf{p}$:

1. Образовать двойственные множества $S^+$ из $\{\pi_i\colon i \in I_+\}$ и $S^-$ из $\{\pi_i\colon i \in I_-\}$. Построить выпуклые оболочки $S^+$ и $S^-$. [За время $O(N \log N)$.]

2. Если пересечение этих выпуклых оболочек непусто (т. е. обладает внутренностью), то множество неравенств несовместно. Иначе, построить разделяющую плоскость $p_1x + p_2y + p_3z + p_4 = 0$. [За время $O(N)$, используя метод, упомянутый в замечании 2 в разд. 7.2.5.]

Теперь можно построить соответствующее преобразование вращения $E^4$ и применить его за время $O(N)$, после чего завершить решение задачи, затратив дополнительно $O(N \log N)$ времени, с помощью методов, описанных ранее.

**Теорема 7.16.** *Общую внутренность и общую внешность множества из $N$ полупространств в $E^3$ можно построить за время $\theta(N \log N)$ (т. е. за оптимальное время).*

Оптимальность следует из того факта, что нижняя оценка $\Omega(N \log N)$, полученная для пересечения $N$ полуплоскостей, тривиально применима к рассматриваемой задаче.



## 7.4. Замечания и комментарии

Методы и результаты, продемонстрированные в предыдущих разделах, не покрывают полного каталога богатой литературы по задачам о геометрических пересечениях. В данном разделе делается попытка по крайней мере частично устранить этот пробел посредством краткого упоминания многих дополнительных важных результатов.

Подход, предложенный Чазелле и Добкином [Chazelle, Dobkin (1980)] для решения задач о пересечениях, аналогичен обработке массовых запросов. В частности, вместо того, чтобы рассматривать пару геометрических объектов и вычислять их пересечение, они предложили рассматривать большие множества таких объектов и предобрабатывать их так, чтобы потом их попарные пересечения можно было проверять очень быстро (задача типа С.3). Они занимались объектами типа точек, прямых, плоскостей, многоугольников и полиэдров; для любых пар объектов такого типа (т. е. пар (многоугольник, многоугольник) или (плоскость, полиэдр) и т. п.) они предложили алгоритмы проверки факта их пересечения, принадлежащие классу «полилог» [1]). Например, случай пары (точка, полиэдр) был решен ими методом локализации точки на плоскости (разд. 2.2) путем стереографического проецирования этих полиэдра и точки на подходящую плоскость; в свою очередь решение этой задачи приводит путем прямого применения дихотомии (разд. 1.3.3) к решению задачи типа (плоскость, полиэдр).

Еще один класс задач о пересечениях, к которому, в частности, относится задача о пересечении отрезков, представленная в разд. 7.2.3, это отчет обо всех $K$ пересекающихся парах на множестве из $N$ объектов. Цель состоит в разработке алгоритмов, время работы которых равнялось бы $O(f(N) + K)$, т. е. оно содержит фиксированную верхнюю оценку $O(f(N))$, а кроме того, пропорционально размеру искомого подмножества. В алгоритме из разд. 7.2.3 эта цель не достигается, поскольку оценка времени его работы равна $O((K + N)\log N)$, а соответствующая нижняя оценка для данной задачи равна $\Omega(K + N \log N)$. Это несовпадение оценок, привело к большой исследовательской работе. Поскольку была широко распространена уверенность в том, что несовпадение оценок объясняется скорее неэффективностью предложенного алгоритма, чем неадекватностью нижней оценки, то велся поиск более эффективных методов решения для частных случаев задачи, когда на отрезки налагаются некоторые ограничения. Первый

[1]) Это — сокращение словосочетания «полиномиально-логарифмический» и относится к оценкам вида $O(\log^c n)$. — *Прим. перев.*

шаг в этом направлении был сделан Нивергельтом и Препаратой [Nievergelt, Preparata (1982)], которые рассмотрели пересечение двух плоских карт (случай, когда отрезками являются ребра двух планарных графов, уложенных на плоскости); они показали, что если грани каждого из графов выпуклы, то достижима оценка по времени $O(N \log N + K)$. Фактически требование выпуклости не обязательно, как следует из более позднего результата, полученного Мэйрсоном и Столфи [Mairson, Stolfi (1983)]: пересечение двух наборов $A$ и $B$, каждый из которых состоит из непересекающихся отрезков, и таких, что $|A|+|B|=N$, можно определить за время $O(N \log N + K)$, где $K$ — число пересечений. Еще один пример задач подобного рода возникает, когда каждый набор ($A$ и $B$) содержит параллельные отрезки; очень простой метод решения этой задачи, обладающий такой же временно́й оценкой, будет изложен в гл. 8. Все эти методы относятся к типу плоского заметания; для них в общем случае операция динамической вставки точки пересечения в список точек событий требует $O(K \log K)$ времени. Полностью отказавшись от использования методов типа заметающей прямой, Чазелле [Chazelle (1983a)] предложил алгоритм для пересечения отрезков, основанный на иерархическом подходе и требующий $O(K + N \log^2 N/\log\log N)$ времени. Затем Чазелле и Эдельсбруннер [Chazelle, Edelsbrunner (1988)] предложили алгоритм решения данной задачи с оптимальной оценкой по времени равной $O(N \log N + K)$. Этот алгоритм является оригинальной комбинацией ряда известных методов, таких как плоское заметание, деревья отрезков, топологическое заметание, амортизация затрат и т. д. Хотя основным методом здесь по-прежнему осталось плоское заметание, однако, алгоритм поддерживает не просто статус заметающей прямой, а скорее всю сцену справа от нее, образованную пересекающими ее отрезками. В отличие от близкого к оптимальности алгоритма Бентли — Оттмана, использующего $O(N)$ памяти, в обсуждаемом алгоритме используется $O(N+K)$ памяти. Следовательно, вопрос о возможности одновременного достижения оптимальных оценок по времени памяти остается открытым.

Если вместо перечисления пересечений $N$ отрезков требуется просто получить их число (подсчет числа пересечений отрезков), то задача существенно изменяется. Существует очевидная преобразуемость этих двух задач, поскольку любой алгоритм перечисления пересечений можно использовать для решения соответствующей задачи подсчета; однако эффективность может ухудшиться, так как число пересечений $K$ может достигать величины $\theta(N^2)$. Лучшим результатом, не зависящим от $K$ среди известных на сей день, является алгоритм, принадлежащий Чазелле [Chazelle (1983a)], который требует $O(N^{1,695} \log N)$ времени и использует линейную память.



В разд. 7.2.3.2 было показано, что для определения факта отсутствия пересечений среди $N$ отрезков на плоскости достаточно $\Omega(N \log N)$ операций. Что произойдет, если потребовать, чтобы эти $N$ отрезков были ребрами какого-нибудь многоугольника? В этом случае задача определения пересечения сводится к проверке простоты многоугольника, и было бы интересно исследовать вопрос о том, приводит ли это дополнительное условие к упрощениям алгоритма. По сей день для задачи о ПРОВЕРКЕ ПРОСТОТЫ МНОГОУГОЛЬНИКА не удается дать оценку сложности. Недавно появилось сообщение о прогрессе в решении близкой задачи о ПЕРЕСЕЧЕНИИ РЕБЕР МНОГОУГОЛЬНИКА; для заданного $N$-угольника $P$ требуется перечислить все его ребра, которые пересекаются (с какими-нибудь другими ребрами). Для этой задачи необходимо $\Omega(N \log N)$ операций, как показал Яромчик, используя общую теорию Бен-Ора [Jaromczyk (1984)]. К сожалению, эта оценка не повлияла на оценку сложности решения задачи о ПРОВЕРКЕ ПРОСТОТЫ МНОГОУГОЛЬНИКА, поскольку преобразование последней к задаче о ПЕРЕСЕЧЕНИИ РЕБЕР МНОГОУГОЛЬНИКА направлено в обратную сторону. Поэтому вопрос об оценке сложности задачи о ПРОВЕРКЕ ПРОСТОТЫ МНОГОУГОЛЬНИКА остается замечательной нерешенной проблемой вычислительной геометрии.

Оптимальный по времени алгоритм построения пересечения двух выпуклых многоугольников, изложенный в разд. 7.2.1, является плоским заметанием. Его непосредственное обобщение на случай трех измерений оказывается неоптимальным по причинам, приведенным в начале разд. 7.3.1. Однако корректное обобщение было недавно получено Гертелем и др. [Hertel, Melhorn, Mäntylä, Nievergelt (1985)]; их метод не только является очень важным примером алгоритма пространственного заметания, но и представляет собой альтернативу технике построения пересечения многоугольников, изложенной в разд. 7.3.1, и может служить эталоном элегантности (а также высокой производительности). Чтобы дать представление об этом методе, возьмем его двумерный аналог в форме плоского заметания, при котором вычисляются точки, являющиеся пересечениями границ двух многоугольников $P_1$ и $P_2$; затем обходом попеременно по границам $P_1$ и $P_2$ между точками их пересечения строим границу $P_1 \cap P_2$. Паре ребер многоугольника $P_i$ ($i = 1, 2$), пересеченных заметающей прямой в $E^2$, соответствует кольцо из многоугольников (именуемое *короной*), пересеченных заметающей плоскостью в $E^3$, а роль точек пересечения в $E^2$ играют многоугольные кольца в $E^3$. Авторы метода

показали, что пространственное заметание эффективно корректирует эти короны и определяет по меньшей мере одну из вершин в каждом многоугольном кольце, начиная с которой можно легко построить всю границу. Доказано, что этот алгоритм требует $O(N \log N)$ времени. Соответствующая задача проверки, безусловно, не сложнее задачи построения, а, вероятно, даже проще. И действительно, Добкин и Киркпатрик [Dobkin, Kirkpatrick (1984)] изобрели алгоритм для решения этой задачи с оценкой $\theta(N)$, а также отметили, что Дайер независимо получил аналогичный результат.

Задача построения ядра простого многоугольника (разд. 7.2.6) представляет интересный большой класс, к которому относят *задачи видимости* (ядро фактически — это множество таких точек, из которых видимы все вершины). Подобные задачи возникают во многих приложениях в машинной графике, анализе сцен, робототехнике и т. д. Основная идея заключается в том, что две точки $p_1$ и $p_2$ (обоюдно) видимы, если отрезок $\overline{p_1p_2}$ не пересекает никакой «запретной» кривой. Видимость обычно определяется по отношению к некой точке $p$. В плоском случае, если запретными кривыми являются отрезки, то локус точек, видимых из $p$ и называемых *многоугольником видимости* для $p$, представляет собой звездный многоугольник (возможно, неограниченный). Многоугольник видимости ограничен, если $p$ лежит внутри простого многоугольника $P$, а граница $P$ является запретной кривой. Для случая, когда множество отрезков на плоскости представляет собой единственный простой многоугольник, известны по меньшей мере два алгоритма с линейной оценкой по времени [ElGindy, Avis (1981); Lee (1983b)]. Используемый при этом подход близок к идее сканирования по Грэхему: здесь сначала определяется одна точка, видимая из $p$, а затем (используя свойство простоты) вершины $P$ сканируются, а невидимые из $p$ участки $P$ удаляются. В том случае, когда множество отрезков представляет собой $m$ несвязанных выпуклых многоугольников, обладающих в сумме $N$ ребрами, а $K$ — число ребер в искомом ядре, данную задачу можно решить за время $O(m \log N + K)$ [Lee, Chen (1985)]. Эдельсбруннер и др. [Edelsbrunner, Overmars, Wood (1983)] также изучали эту задачу и исследовали вопрос о поддержании многоугольников видимости в том случае, когда разрешены операции вставки или удаления, и, кроме того, вопрос о когерентности, когда разрешены перемещения направления взгляда или заданных точек. Реализуя теоретико-множественные операции на многоугольниках видимости, можно определить специальные типы видимости; за сжатым обзором этого материала читатель отсылается к работе Туссена [Toussaint (1980)].



## 7.5. Упражнения

1. Заданы два выпуклых многоугольника $P_1$ и $P_2$, число вершин которых в сумме равно $N$. Построить алгоритм типа плоского заметания, который вычислял бы пересечение $P_1$ и $P_2$ за время $O(N)$ следующим образом:
(a) определял бы $\mathcal{I}$-множество всех пересечений границ $P_1$ и $P_2$;
(b) используя $\mathcal{I}$, строил бы границу $P_1 \cap P_2$.

2. Задан набор $S$, состоящий из $N$ отрезков на плоскости; построить алгоритм, который решал бы вопрос о существовании прямой $l$, пересекающей все отрезки из $S$ (прокалывающей прямой), и, если она есть, строил бы ее.

3. Задан набор $S$, состоящий из $N$ отрезков на плоскости, и пробный отрезок $s^*$; надо предварительно обработать $S$ так, чтобы получить эффективный алгоритм для проверки факта пересечения $s^*$ с произвольным элементом $S$.

4. Пусть $P$ — выпуклый полиэдр. Запрос заключается в проверке пересечения отрезком $s$ из $E^3$ полиэдра $P$. Предполагая, что работа ведется с типовыми запросами, предварительно обработать $P$ и построить эффективный алгоритм обработки запросов.

5. Заданы два выпуклых полиэдра $P$ и $Q$, число вершин которых в сумме равно $N$. Можете ли вы придумать алгоритм, *проверяющий* факт пересечения $P$ и $Q$ за время $\theta(N)$?

6. Преобразовать алгоритм построения пересечения двух выпуклых полиэдров (данный в разд. 7.3.1) в тест, проверяющий факт линейной разделимости двух полиэдров $P$ и $Q$ и, если таковая обнаружена, строящий одну из разделяющих плоскостей $\pi$. Этот алгоритм должен затрачивать время, линейно зависящее от общего числа вершин в $P$ и $Q$.

ГЛАВА 8

# Геометрия прямоугольников

Методы, которые были ранее разработаны для задач о пересечении плоских выпуклых многоугольников, безусловно, применимы и к прямоугольникам. Точно так же набор отрезков, каждый из которых параллелен или перпендикулярен любому другому элементу из этого набора, можно обработать обычными методами решения задач о пересечениях, которые были описаны в предыдущих главах. Однако весьма специальный вид отрезков, параллельных ровно двум ортогональным прямым, или прямоугольников, сторонами которых являются подобные отрезки, заставляет предположить, что можно разработать и более эффективные методы, учитывающие специфику этих в высшей степени структурированных случаев. А в процессе изучения задач этого класса можно приобрести более глубокое понимание геометрических свойств, которыми обладают вышеупомянутые объекты — прямоугольники и ортогональные отрезки, — а также охарактеризовать более широкий класс задач, к которым применимы указанные методы.

Все это было бы интересным, хотя и довольно бесполезным упражнением, если бы прямоугольники и ортогональные отрезки не являлись важнейшими компонентами многих приложений, часть из которых обладает чрезвычайной важностью в области современных передовых технологий, особенно в области цифровых схем с очень высоким уровнем интеграции (СБИС). Поэтому, перед тем как начать подробное обсуждение алгоритмического материала, стоит потратить немного времени на описание ряда важных приложений.

## 8.1. Некоторые приложения геометрии прямоугольников

### 8.1.1. Проектирование СБИС

Фотошаблоны, используемые при изготовлении интегральных схем, зачастую представляют собой наборы прямоугольников, стороны которых параллельны двум взаимно ортогональным направлениям. Каждый из этих прямоугольников является или участком проводника, или областью ионной имплантации, или межслойным контактом и т. п., как показано на рис. 8.1 [Lauther (1978); Baird (1978); Mead, Conway (1979)].



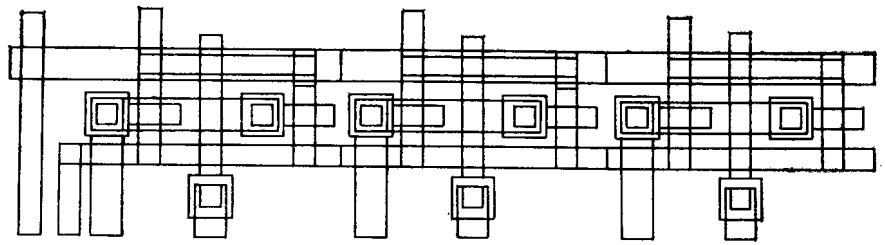


Рис. 8.1. Типичная геометрия одной интегральной схемы.

Эти прямоугольники размещаются на плоскости в соответствии с конкретными техническими требованиями, которые определяют минимальные зазоры между одними типами прямоугольников, минимальные наложения для других типов и т. д. Условия на зазоры и наложения возникают из-за конфликтующих требований минимизировать общую используемую площадь (это очевидное экономическое ограничение) и выполнить необходимые электрические условия (изоляцию или контакт) с учетом возможных отклонений при производстве. (Очевидно, что, чем меньше зазор, тем больше вероятность возникновения нежелательного контакта.)

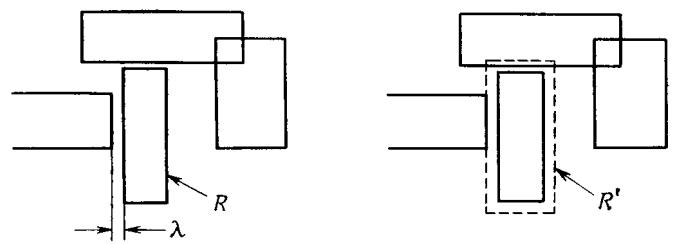



Рис. 8.2. Проверить величину минимального зазора $\lambda$ вокруг прямоугольника $R$ можно путем расширения $R$ во все стороны на $\lambda$ (для формирования $R'$) и контроля наложений.

Следовательно, важнейшей задачей при проектировании фотошаблонов ИС (интегральных схем) является проверка выполнения проектных норм, или, выражаясь иначе, «контроль соблюдения проектных норм». На рис. 8.2 показан пример того,

как можно проверить нарушение минимальной величины $\lambda$ зазора между прямоугольником $R$ и соседствующими с ним прямоугольниками. Для этого можно увеличить $R$, добавив к нему полосу ширины $\lambda$, и проверить, не будет ли увеличенный прямоугольник $R'$ пересекаться с какими-нибудь из своих соседей. Значит, задача проверки зазора преобразуется в задачу о пересечении прямоугольников. Разновидности подобного подхода можно использовать при проверке выполнения и других технических требований, таких как наложения, покрытия и т. п.

В то время как описанные выше этапы связаны в определенном смысле с «синтаксической» корректностью фотошаблона, другая важная задача заключается в проверке того, что

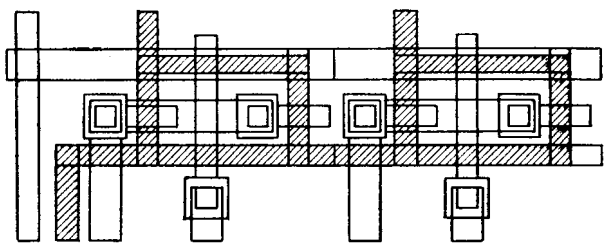


Рис. 8.3. Определение компонент связности на множестве прямоугольников можно использовать при решении задачи выделения электрических цепей.

фотошаблон функционально корректно реализует заданную схему. Первым этапом этой задачи является интерпретация геометрии фотошаблона как электрической схемы, т. е. идентификация таких подмножеств прямоугольников, все элементы которых электрически связаны. Такая работа называется *восстановлением принципиальной схемы*. Выражаясь несколько более абстрактными терминами, можно представить себе граф, вершины которого соответствуют прямоугольникам и такой, что пара вершин соединена ребром, если соответствующие прямоугольники имеют непустое пересечение. В такой формулировке восстановление принципиальной схемы сводится к задаче обнаружения *компонент связности* данного графа. Однако выделение компонент связности дает несколько меньше информации, чем требуется для указанного восстановления, поскольку на самом деле нужно очертить границу (так называемый *контур*) каждой компоненты связности (рис. 8.3).

Итак, мы видим, что проектирование интегральных схем служит источником многих геометрических задач, связанных с прямоугольниками, из числа которых были проиллюстрированы лишь некоторые важные задачи.



### 8.1.2. Конфликтующие запросы к базам данных

Одна интересная геометрическая модель была разработана недавно для конфликтующих запросов к базе данных от нескольких пользователей [Yannakakis, Papadimitriou, Kung (1979)]. Обычно *транзакция* одного пользователя является последовательностью шагов, каждый из которых адресуется к одному элементу базы данных (*переменной*) и корректирует его. Такая корректировка переменной, однако, имеет определенную продолжительность. В течение этого промежутка времени необходимо предотвращать любые попытки других пользователей обращаться к этой же переменной. (Более конкретный пример: если запрашиваемой переменной является пассажирское место в авиалайнере, то очевидно, что для сохранения правильного учета все транзакции, использующие эту переменную, должны обрабатываться последовательно.) Весьма распространенным методом разрешения таких конфликтов является *блокировка* [Eswaran, Gray, Lorie, Traiger (1976)], когда пользователь «блокирует» переменную в начале корректировки и «деблокирует» в конце. Блокированная переменная недоступна другим пользователям.

Эту ситуацию можно промоделировать следующим образом. Работа каждого пользователя является последовательностью шагов типа «блокировка $x$», «корректировка $x$» и «деблокировка $x$», где $x$ — переменная. Если существует $d$ пользователей, то каждому из них сопоставляем одну из осей $d$-мерного декартова пространства $E^d$ и для каждой оси устанавливаем взаимно однозначное соответствие между шагами транзакции и точками с целочисленными координатами. При этом корректировке переменной соответствует интервал (интервал блокировки) на оси, который ограничен точками, соответствующими шагам блокировки и деблокировки. Поэтому, если $k$ пользователей ($k \leqslant d$) обращаются к одной и той же переменной, то декартово произведение соответствующих интервалов блокировки образует $k$-мерный прямоугольник. Для случая двух пользователей $U_1$ и $U_2$ эта ситуация проиллюстрирована на рис. 8.4(а). Состояние базы данных представимо точкой на координатной плоскости $(U_1, U_2)$, фиксирующей историю работы каждого пользователя; например, точка с координатами (8, 3) на рис. 8.4(а) обозначает, что шаги 8 и 3 пользователей соответственно $U_1$ и $U_2$ выполнялись одновременно. Ясно, что в системе транзакций с блокировкой точка состояния не может лежать внутри прямоугольника, связанного с переменной.

Как указывалось ранее, любой пользователь может обращаться к нескольким переменным. Каждый доступ к переменной $x$ состоит из последовательности шагов «блокировка $x$»,

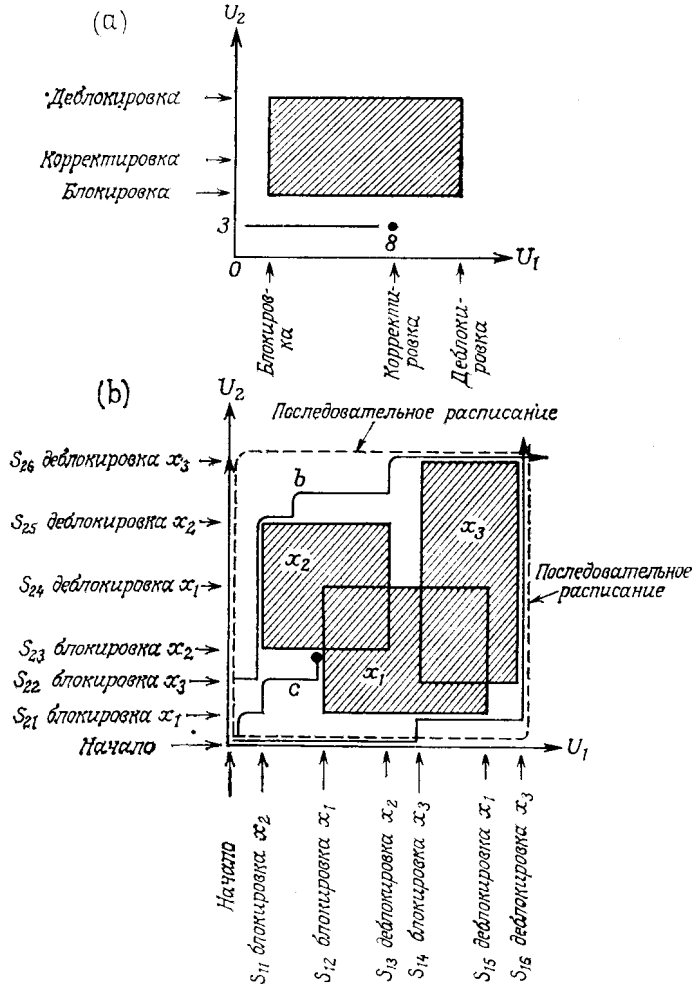



Рис. 8.4. Моделирование базы данных с блокировкой транзакций. (a) — два пользователя, одна переменная; (b) — два пользователя, три переменные.

«корректировка $x$» и «деблокировка $x$»; последовательность шагов определенного пользователя (его работа) изображается посредством множества точек на соответствующей ему оси, как это показано на рис. 8.4(b) для случая двух пользователей.

*Расписание* иллюстрирует реальное изменение базы данных и складывается случайным образом путем объединения работ каждого из пользователей. Поскольку работа каждого является последовательностью шагов, то допустимое расписание изображается монотонно неубывающей ступенчатой кривой на плоскости $(U_1, U_2)$ рис. 8.4(b). Начальный участок такой кривой описывает *частичное* расписание.



*Последовательным* называется такое расписание, при котором пользователи работают поочередно, т. е. пользователь получает доступ к базе данных и сохраняет его до конца своей работы. Очевидно, что для $d$ пользователей существует $d!$ последовательных расписаний (а для двух пользователей будет ровно два таких, которые показаны на рис. 8.4(b)). Система работы с базой данных считается *безопасной*, если результат

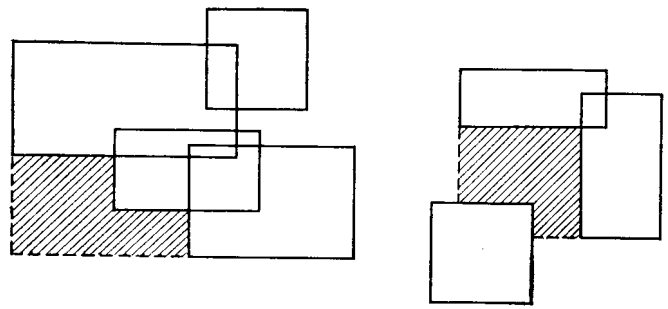


Рис. 8.5. Иллюстрация ЮЗ-замыкания для объединения прямоугольников. Любое расписание, исключающее эту область, будет свободным от тупиков.

работы по любому законному расписанию (т. е. расписанию, обходящему прямоугольники) совпадает с результатом работы по последовательному расписанию. Это условие можно выразить в чисто геометрических терминах [Yannakakis, Papadimitriou, Kung (1979)], если потребовать, чтобы любое безопасное расписание было *гомотопно* (т. е. преобразуемо непрерывной деформацией, избегающей прямоугольники) последовательному расписанию.

С другой стороны, такая система может достигнуть условия безвыходности (*тупика*), в котором она будет пребывать неопределенное время без постороннего вмешательства. Эта ситуация представлена, например, частичным расписанием $c$, показанным на рис. 8.4(b) и состоящим из последовательности шагов $S_{21}$, $S_{11}$, $S_{22}$, $S_{12}$, $S_{23}$. Шаг $S_{12}$ (блокировка $x_1$) не может выполниться, поскольку $x_1$ уже блокирована (на шаге $S_{21}$); поэтому $S_{12}$ должен ожидать, а управление передается пользователю $U_2$, которому нужно выполнить шаг $S_{23}$ (блокировку $x_2$). Но $x_2$ уже заблокирована (на шаге $S_{11}$), поэтому никакая дальнейшая работа невозможна, ибо оба пользователя пытаются проделать невыполнимые шаги.

Условия *безопасности* и *отсутствия тупиков* хорошо видны на диаграмме транзакций к базе данных. Прежде всего, кривая,

изображающая расписание, не может заходить внутрь объединения прямоугольников транзакций. Это объединение будет представлять собой набор компонент связности (рис. 8.5). Но одно это условие не гарантирует, что в системе будут отсутствовать тупики. На самом деле нужно описать некую область, содержащую прямоугольники транзакций и такую, что в любом расписании (т. е. в любой монотонно неубывающей кривой), внешнем по отношению к ней, не было бы тупиков. Такая область показана на рис. 8.5, где заштрихованные области были добавлены к объединению прямоугольников транзакций. Теперь интуитивно ясно, что подобное расширение объединения прямоугольников удовлетворяет заданному требованию. Эта область называется ЮЗ-*замыканием* (юго-западным замыканием) объединения прямоугольников [Yannakakis, Papadimitriou, Kung (1979); Lipski, Papadimitriou (1981); Soisalong Soininen, Wood (1982)] и будет строго определена в разд. 8.6.

## 8.2. Область применения результатов

Хотя наборы прямоугольников и были исходными объектами, вызвавшими бóльшую часть объема исследовательской работы в данной области, и практически все результаты формулируются в терминах взаимно ортогональных направлений (обычно координатных осей), было бы уместно попытаться именно сейчас определить максимальную сферу применения данной теории.

Начнем с рассмотрения двух наборов $S_1$ и $S_2$, каждый из которых содержит параллельные отрезки, причем направления отрезков из $S_1$ и $S_2$ взаимно ортогональны. Прямые, несущие эти отрезки, образуют сетку на плоскости. Представим теперь точки в однородных координатах $(x_1, x_2, x_3)$ (см. разд. 1.3.2 и 7.3.2) и применим к точкам этой плоскости обычное невырожденное преобразование проецирования, т. е. преобразование, описываемое невырожденной матрицей третьего порядка $\mathscr{A}$. Хорошо известно (см., например, [Ewald (1971)] или [Birkhoff, MacLane (1965)]), что это преобразование переводит точки в точки, а прямые в прямые и *сохраняет инцидентность*, т. е. любое такое преобразование не меняет структуру сетки. Если матрица $\mathscr{A}$ имеет вид

$$\mathscr{A} = \left|\begin{array}{c|c} \mathscr{B} & \begin{matrix}0\\0\end{matrix} \\ \hline 0\ 0 & a \end{array}\right|,$$

где $\mathscr{B}$ — невырожденная матрица второго порядка, и $a \neq 0$, то прямая на бесконечности переходит сама в себя, а два направления, которые были вначале ортогональны, становятся теперь произвольными. Если же невырожденная матрица $\mathscr{A}$ не обладает блочной формой, то одна или обе точки, лежащие на бесконечности на двух указанных исходных направлениях, перейдут в конечные точки, причем пропадает даже свойство параллельности прямых в каждом из наборов. Одна из типичных ситуаций такого рода показана на рис. 8.6.



Приведенные соображения, по сути, характеризуют область применения так называемой геометрии прямоугольников и приводят к следующим определениям:

**Определение 8.1.** Множество четырехугольников $\mathscr{R}$ называется *изотетичным*[1]), если стороны каждого из элементов $\mathscr{R}$ принадлежат одному из двух пучков прямых линий с центрами в точках $p_1$ и $p_2$ соответственно ($p_1$ и $p_2$ могут принадлежать прямой на бесконечности в этой плоскости), и все элементы $\mathscr{R}$ лежат полностью по одну сторону от прямой, проходящей через точки $p_1$ и $p_2$.

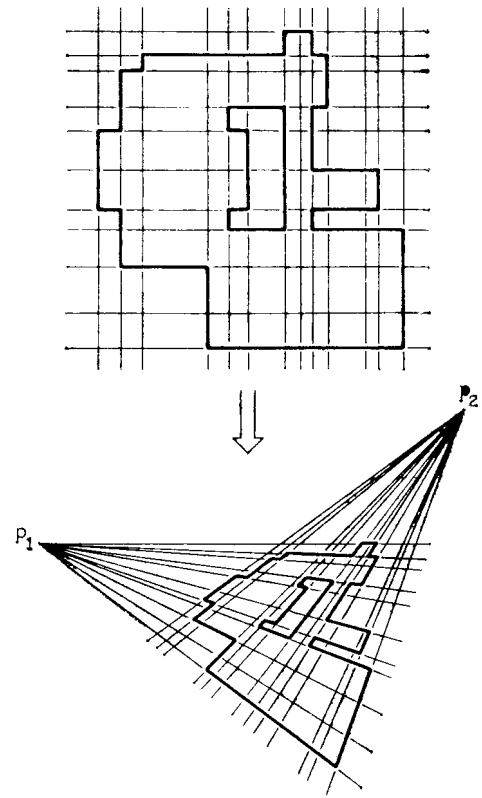



Рис. 8.6. Пример невырожденного проективного преобразования применительно к плоской фигуре, образованной ортогональными отрезками.

После этого пояснения для простоты представления далее будем рассматривать тот случай, когда $p_1$ и $p_2$ определяют ортогональные направления.

[1]) «Изотетичные» означает «одинаково расположенные» от греческих слов isos (одинаковый) и tithenai (располагать). Существует заметная терминологическая неопределенность данного понятия между оценками понятий «прямолинейные», «ортогональные», «выравненные», «изоориентированные». Хотя смыслы последних понятий однозначны (за исключением, вероятно, понятия «прямолинейные»), понятие «изотетичные» по смыслу ближе всего именно к нему.

Алгоритмы, о которых будет идти речь в данной главе, связаны с определением различных свойств множеств прямоугольников (или ортогональных отрезков). Можно предложить одну фундаментальную классификацию этих алгоритмов, зависящую от базисного типа операций, т. е. зависящую от того, будем ли мы работать в статической или динамической среде. В первом случае вся информация, необходимая для решения задачи, полностью известна до начала вычислений; во втором случае допустимы также динамические корректировки данных путем операций вставки и удаления.

Операции этих двух типов — статические и динамические — подразумевают, как и следовало ожидать, использование существенно различающихся структур данных. В то время как работа в статическом режиме в настоящее время хорошо изучена, исследование более сложных динамических операций пребывает в стадии активного развития. Поэтому целесообразно дать здесь детальное описание алгоритмов именно статического типа; отсылки к продолжающимся исследованиям алгоритмов динамического типа будут приведены в конце настоящей главы.

Последующее изложение будет относиться в основном к двумерным приложениям. Но там, где это уместно, будет дано обобщение на $d$-мерный случай либо о нем будет упоминаться.

## 8.3. Общие замечания по алгоритмам статического типа

Уникальная особенность набора изотетичных прямоугольников (или, что то же самое, двух ортогональных наборов параллельных отрезков) состоит в том, что плоскость разбивается на полосы (например, на вертикальные), в каждой из которых задача становится одномерной. (Аналогично пространство $E^d$ можно разбить на $(d-1)$-мерные бруски.) В частности, если взять абсциссы вертикальных сторон этих прямоугольников, то внутри каждой полосы, заключенной между смежной парой этих абсцисс, все вертикальные сечения будут одинаковыми и будут состоять из ординат тех горизонтальных сторон, которые пересекают эту полосу. Поэтому такие вертикальные сечения на плоскости разбиваются на классы эквивалентности (каждый из этих классов будет множеством сечений, заключенных между парой последовательных абсцисс вертикальных сторон прямоугольников). Кроме того, будет показано, что сечение, принадлежащее определенной полосе, можно получить в результате небольшой модификации сечений, принадлежащих одной или двум смежным полосам.



Некоторые из интересных с вычислительной точки зрения задач, связанных с множеством прямоугольников (таких, как определение площади их объединения, периметра этого объединения, его контура, регистрации пересечений и т. д.), обладают одним очень полезным свойством. Произвольная вертикальная прямая определяет собой две полуплоскости; решение же исходной задачи, как правило, соотносится весьма просто с решениями аналогичных подзадач для каждой из этих полуплоскостей. Это простое соотношение иногда выступает в виде теоретико-множественного объединения (для отчета о пересечениях), или арифметической суммы (для вычисления площади и периметра), или простого сцепления компонент (для контура). Во всех этих случаях пересечение разделяющей прямой с множеством прямоугольников содержит всю необходимую для комбинации частных решений информацию. Более того, предполагается, что решение, получаемое в результате реализации операций статического типа для, скажем, левой полуплоскости является *окончательным* (т. е. не изменяемым корректировками, возможными *в будущем*), так что глобальное решение можно получить *последовательным* расширением текущего решения вправо. Мы видим, что это типичная ситуация, когда оправданно использование метода плоского заметания, а именно: заметающая прямая параллельна одному из направлений, а движение осуществляется вдоль перпендикулярного ему направления.

Обращая внимание на характерное вертикальное сечение множества прямоугольников, которое является просто последовательностью ординат, заметим, что, поскольку пересеченные отрезки параллельны, последовательность ординат этого сечения всегда будет подпоследовательностью упорядоченной последовательности ординат всех горизонтальных отрезков. Путем предварительной нормализации, при которой множество ординат сортируется, а каждая ордината заменяется ее номером в этом упорядочении, последовательность ординат можно уподобить ряду последовательных целых чисел. Поэтому кажется, что для поддержки статуса заметающей прямой нет необходимости в повторной сортировке общей отсортированной последовательности, поскольку можно воспользоваться более эффективными структурами данных, подобранными именно для этой ситуации. Такие структуры полностью определяются (по крайней мере скелетно) множеством ординат *всех* горизонтальных отрезков, а каждое отдельное сечение можно получить, пользуясь «скелетной» метафорой, путем «наделения плотью» части скелета. Одним из примеров такой структуры является *дерево отрезков*, которое уже изучалось и использовалось ранее в данной книге (см. разд. 1.2.3.1); другой пример — *де-*

*рево интервалов* [McCreight (1981); Edelsbrunner (1980)], которое будет описано в разд. 8.8.1.

Для удобства напомним, что каждый узел $v$ из дерева отрезков характеризуется интервалом $[B[v], E[v]]$ и некоторыми дополнительными параметрами, необходимыми для выполнения особых вычислений. Один из этих параметров $C[v]$ — *степень* узла — возникает во всех приложениях, поскольку он обозначает число отрезков, отнесенных к узлу $v$; другие же параметры имеют смысл только для тех или иных приложений.

Элементарными операциями на дереве отрезков являются вставка и удаление. В частности, если $[b, e]$ обозначает интервал, то на узле $v$ определены операции ВСТАВИТЬ$(b, e; v)$ и УДАЛИТЬ$(b, e; v)$. Обычно в следующих разделах процедура ВСТАВИТЬ будет иметь такой вид:

```
        procedure ВСТАВИТЬ(b,e; v)
1.  begin if (b ≤ B[v]) and (E[v] ≤ e) then C[v] := C[v] + 1
2.        else begin if (b < ⌊B[v] + E[v])/2⌋) then
                        ВСТАВИТЬ(b,e; ЛСЫН[v]),
3.                  if (⌊(B[v] + E[v])/2⌋ < e) then
                        ВСТАВИТЬ(b,e; ПСЫН[v])
              end;
4.        КОРРЕКТИРОВАТЬ(v) (* корректировка особых
          параметров v *)
    end.
```

Важнейшей характеристикой этой процедуры, изменяющейся от приложения к приложению, является функция КОРРЕКТИРОВАТЬ $(v)$ из строки 4, частные случаи которой будут обсуждаться в соответствующих местах. Аналогичные соображения применимы и к процедуре УДАЛИТЬ$(b, e; v)$.

В заключение заметим, что метод плоского заметания и использование структур данных, ориентированных на хранение интервалов, являются естественными средствами для решения множества задач, возникающих в геометрии прямоугольников. Приведенные ниже разделы посвящены детальному разбору специальных методов.

## 8.4. Мера и периметр объединения прямоугольников

В одной статье [Klee (1977)], которая представляет заметный интерес и может считаться началом целого направления исследований, Кли поставил следующий вопрос: «Заданы $N$ интервалов $[a_1, b_1], \ldots, [a_N, b_N]$ на действительной прямой, надо найти меру их объединения. Какой может быть трудоемкость этой процедуры?».



Такая постановка дает простейший (одномерный) пример задачи о мере объединения. (Будем называть задачу о вычислении меры объединения для произвольного числа измерений задачей О.1.[1]) Кли сразу же построил алгоритм решения этой задачи с оценкой $O(N \log N)$, основанный на предварительной сортировке абсцисс $a_1, b_1, \ldots, a_N, b_N$ в массиве $X[1:2N]$. Дополнительное свойство этого массива состоит в том, что если $a_i$ помещено в $X[h]$, $b_j$ — в $X[k]$ и $a_i = b_j$, то $h < k$ (т. е. правая концевая точка помещается в нем после левой точки, обладающей такой же абсциссой). Вычисление завершается простым просмотром массива $X[1:2N]$ за линейное время (нижеследующий алгоритм представляет собой адаптацию алгоритма Кли, удобную для последующих обобщений):

```
        procedure МЕРА-ОБЪЕДИНЕНИЯ-ИНТЕРВАЛОВ
1.  begin X[1:2N] := упорядоченная последовательность
          абсцисс концов интервалов;
2.        X[0] := X[1];
3.        m := 0; (* m — мера *)
4.        C := 0; (* C — число перекрывающихся интервалов *)
5.        for i := 1 until 2N do
6.            begin if (C ≠ 0) then m := m + X[i] − X[i − 1];
7.                  if (X[i] — левая концевая точка) then
8.                      C := C + 1 else C := C − 1
              end
    end.
```

В своей статье Кли также отметил, что, хотя сортировка является ключом к получению вышеизложенного результата, она не является *априорно* необходимой, и поставил вопрос о существовании какого-нибудь метода решения, требующего $o(N \log N)$ операций.

Ответ на этот вопрос был получен Фредменом и Вайде [Fredman, Weide (1978)] для модели вычислений с линейным деревом решений (к которой относится и вышеизложенный алгоритм); применение общего метода Бен-Ора (разд. 1.4) расширяет пределы этого результата на произвольные алгебраические деревья решений. Закономерно, что доказательства подобного типа укладываются в уже знакомую схему, встречавшуюся ранее в связи с исследованиями выпуклых оболочек (разд. 3.2), максимумов на множестве векторов (разд. 4.1.3), уникальности элементов (разд. 5.2) и максимального промежутка (разд. 6.4). В основе лежит симметрическая группа степени $N$, которая вновь является ключом к доказательству. Здесь будет использован следующий вычислительный прототип:

[1]) О — аббревиатура слова «ортогональность». — *Прим. перев.*

**Задача О.2** (ε-БЛИЗОСТЬ). Дано $N+1$ действительных чисел $x_1, x_2, \ldots, x_N$ и $\varepsilon > 0$. Надо определить, будут ли какие-нибудь два числа $x_i$ и $x_j$ $(i \neq j)$ находиться на расстоянии, меньшем чем ε одно от другого.

Прежде всего покажем, что можно легко установить преобразование

ε-БЛИЗОСТЬ $\propto_N$ МЕРА ОБЪЕДИНЕНИЯ ИНТЕРВАЛОВ.

Действительно, построим интервалы $[x_i, x_i + \varepsilon]$ для $i = 1, 2, \ldots, N$. Эти интервалы будут входом для процедуры МЕРА-ОБЪЕДИНЕНИЯ, результатом работы которой будет вещественное число $m$. Теперь ясно, что никакие два числа из

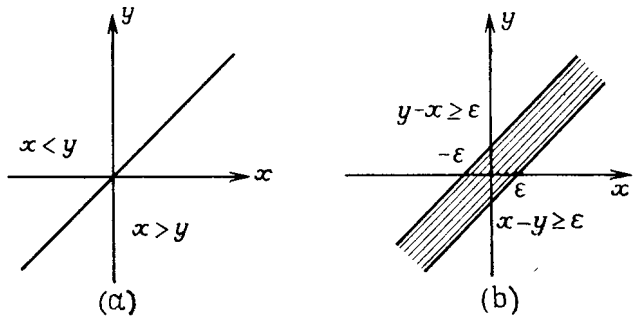



Рис. 8.7. Иллюстрация (в случае $E^2$) различия между множествами истинности в задачах об УНИКАЛЬНОСТИ ЭЛЕМЕНТОВ (a) и ε-БЛИЗОСТИ (b).

$\{x_1, \ldots, x_N\}$ не будут находиться на расстоянии, меньшем чем ε друг от друга, тогда и только тогда, когда $m = N\varepsilon$.

Существует близкое сходство между ε-БЛИЗОСТЬЮ и УНИКАЛЬНОСТЬЮ ЭЛЕМЕНТОВ, изложенной в разд. 5.2. Однако УНИКАЛЬНОСТЬ ЭЛЕМЕНТОВ не является частным случаем ε-БЛИЗОСТИ, поскольку значение $\varepsilon = 0$, необходимое для получения указанного частного случая, недопустимо. Различие этих задач выразительно проиллюстрировано на рис. 8.7 для двумерного случая: если речь идет об УНИКАЛЬНОСТИ ЭЛЕМЕНТОВ, то непересекающиеся связные компоненты множества истинности $W$ (см. разд. 1.4 и 5.2) представляют собой открытые множества, разделенные только их общей границей; если же речь идет об ε-БЛИЗОСТИ, то компоненты представляют собой замкнутые множества и расстояние между ними положительно. Если исключить это различие, то доказательство того, что множество $W$ для задачи об ε-БЛИЗОСТИ распадается на $N!$ непересекающихся компонент связности, будет аналогично доказательству, приведенному в разд. 5.2; отсюда вытекает следствие из теоремы 1.2:



**Следствие 8.1.** *Любой алгоритм, который использует модель алгебраического дерева решений, для определения существования некоторой пары чисел из N-элементного множества, удаленных друг от друга на расстояние, меньшее чем ε, затратит* $\Omega(N \log N)$ *проверок.*

Следствие 8.1 доказывает оптимальность результата Кли для задачи о вычислении меры в случае одного измерения, но оставляет открытым вопрос об оптимальной оценке при $d \geqslant 2$.

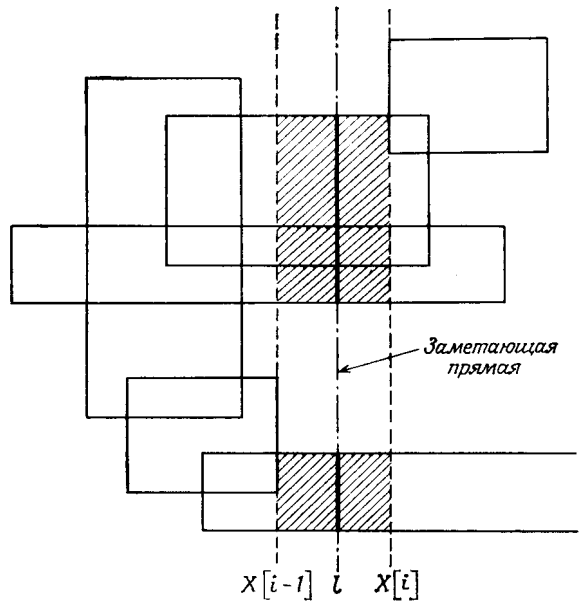



Рис. 8.8. Применение метода плоского заметания к задаче определения меры объединения прямоугольников для случая двух измерений.

Бентли [Bentley (1977)][1]) взялся за эту задачу и преуспел в создании оптимального алгоритма для вычисления меры объединения при $d = 2$. Его подход является простой и разумной модификацией одномерного метода. В частности, при одном измерении длина интервала $[X[i-1], X[i]]$ добавляется к искомой мере (строка 6 в алгоритме МЕРА-ОБЪЕДИНЕНИЯ-ИНТЕРВАЛОВ) в зависимости от того, будет ли хотя бы один отрезок накрывать его или нет, или, что то же самое, будет ли параметр $C$ ненулем или нулем соответственно. Следовательно,

[1]) Этот результат изложен также в работе [Van Leeuwen, Wood (1981)].

необходимо знать только значение $C$ (строка 7). В случае двух измерений (рис. 8.8) вертикальная полоса, заключенная между $X[i-1]$ и $X[i]$, вносит в меру объединения прямоугольников вклад, равный величине $(X[i]-X[i-1])m_i$, где $m_i$ — длина пересечения произвольной вертикальной прямой из этой полосы с объединением прямоугольников. Поэтому величина $m_i$ (являющаяся константой и равная 1 в случае одного измерения) — это ключевой параметр для двумерной техники. Если бы $m_i$ можно было вычислять и корректировать за время, не превышающее $O(\log N)$, то времена сканирования и предварительной обработки были бы равны и был бы получен оптимальный алгоритм с оценкой $\theta(N \log N)$.

Поскольку ординаты горизонтальных сторон прямоугольников известны *заранее*, то определение $m_i$ можно реализовать посредством *дерева отрезков* (см. разд. 8.3). Вспоминая общий формат элементарных операций на дереве отрезков ВСТАВИТЬ И УДАЛИТЬ (см. разд. 8.3), определим для текущего приложения следующий дополнительный параметр узла:

$m(v) :=$ вклад интервала $[B[v], E[v]]$ в величину $m_i$.

Вычисление $m[v]$ проводится следующей процедурой (вызываемой в строке 4 из процедуры ВСТАВИТЬ):

```
        procedure КОРРЕКТИРОВАТЬ(v)
begin   if (C[v] ≠ 0) then m[v] := E[v] − B[v]
        else if (v не лист) then m[v] := m[ЛСЫН[v]] +
                                         m[ПСЫН[v]]
             else m[v] := 0
end.
```

Ясно, что $m_i = m$ [корень дерева отрезков]. Как общий параметр $C[v]$, так и специальный параметр $m[v]$ можно легко корректировать (за константное время на один узел) при вставке или удалении одного отрезка (вертикальной стороны прямоугольника) из дерева отрезков. Итак, корректировку дерева отрезков и вычисление $m_i$ можно реализовать за время $O(\log N)$ для одной вертикальной стороны прямоугольника; тем самым достигается ранее поставленная цель. Теперь можно сформулировать алгоритм, где $b_i$ и $e_i$ обозначают соответственно минимальную и максимальную ординаты вертикальной стороны, имеющей абсциссу $X[i]$. (Заметим, что этот алгоритм является непосредственным обобщением одномерного алгоритма, приведенного ранее.)

```
        procedure МЕРА-ОБЪЕДИНЕНИЯ-
                  ПРЯМОУГОЛЬНИКОВ
1.  begin X[1 : 2N] := упорядоченная последовательность
          абсцисс вертикальных сторон;
```



```
2.        X[0] := X[1];
3.        m := 0;
4.        построить и инициализовать дерево отрезков T
          для ординат сторон прямоугольника;
5.        for i := 1 until 2N do
6.            begin m* := m[корень(T)];
7.                  m := m + m*(X[i] − X[i − 1]);
8.                  if (X[i] — абсцисса левой стороны) then
                        ВСТАВИТЬ(b_i, e_i; корень(T))
9.                  else УДАЛИТЬ(b_i, e_i; корень(T))
              end
  end.
```

Завершим это обсуждение следующей теоремой:

**Теорема 8.1.** *Меру объединения $N$ изотетичных прямоугольников на плоскости можно вычислить за оптимальное время* $\theta(N \log N)$.

Возвращаясь к последнему алгоритму, мы сразу же увидим, что он относится к категории плоского заметания (см. разд. 8.3), где *списком точек событий* служит массив $X[1:2N]$, а *статус заметающей прямой* задается деревом отрезков. Подобная реализация допускает непосредственное обобщение этого метода на случаи более чем двух измерений. Действительно, техника заметания позволяет преобразовать исходную $d$-мерную задачу (на $N$ гиперпрямоугольниках в $E^d$) в «последовательность», состоящую из $N$ штук $(d-1)$-мерных подзадач (каждая на не более чем $N$ гиперпрямоугольниках в $E^{d-1}$). Для $d = 3$ эти подзадачи становятся двумерными, и по теореме 8.1 каждую из них можно решить за время $O(N \log N)$; поэтому задачу о мере объединения при трех измерениях можно решить за время $O(N^2 \log N)$. Это устанавливает базис для индуктивного доказательства, приводящего к следствию 8.2.

**Следствие 8.2.** *Меру объединения $N$ изотетичных гиперпрямоугольников в пространстве $d \geqslant 2$ измерений можно вычислить за время* $O(N^{d-1} \log N)$.

Сильным недостатком описанного метода при $d \geqslant 3$ является тот факт, что при заметании мы не смогли использовать «когерентность»[1]) между двумя последовательными сечениями. (При $d = 2$ дерево отрезков позволяет воспользоваться когерентностью смежных сечений путем эффективного вычисления текущего сечения за счет простой корректировки предыдущего сечения.) Эта идея была впоследствии перенесена на случай

[1]) То есть согласованность, взаимозависимость. — *Прим. перев.*

трех измерений Ван Леювеном и Вудом [Van Leeuwen, Wood (1981)], которые предложили использовать в качестве структуры данных «квадродерево» [Finkel, Bentley (1974)]. Квадродерево можно считать двумерным обобщением дерева отрезков, но весьма курьезно, что оно было предложено раньше, чем эта более простая структура. Сейчас мы кратко опишем его организацию.

*Квадродерево* — это способ организации ортогональной сетки, состоящей из $N \times M$ ячеек, определенных $(N+1)$ горизонтальными и $(M+1)$ вертикальными прямыми. (Дерево отрезков — это соответственно способ организации $N$ смежных интервалов.) Для простоты изложения предположим, что $N = M = 2^k$. Квадродерево $T$ — это дерево со степенями исхода, кратными четырем, у которого с каждым узлом связан участок сетки размером $2^i \times 2^i$ (именуемый $2^i$-квадратом) $(i = 0, 1, \ldots, k)$. $2^i$-квадрат, связанный с заданным узлом $v$ из $T$ (при $i > 0$), разбивается на четыре $2^{i-1}$-квадрата вертикальным и горизонтальным делениями пополам (рис. 8.9); каждый из этих четырех квадратов (именуемых СЗ, СВ, ЮВ и ЮЗ квадратами) связывается с одним из четырех потомков узла $v$. Данное построение начинается со связывания всей сетки с корнем $T$, а заканчивается тогда, когда процесс четвертования дает $2^0$-квадраты (листья $T$). Поскольку $T$ обладает $N^2$ листьями, то его построение занимает $O(N^2)$ времени и использует $O(N^2)$ памяти.

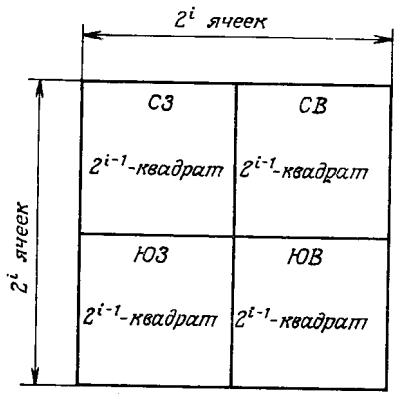



Рис. 8.9. Пример разбиения плоской сетки, осуществляемого одним из узлов квадродерева, который соответствует $2^i$-квадрату.

Теперь мы рассмотрим вопрос о хранении прямоугольника $R$ в квадродереве. Исходная сетка образована $2N$ абсциссами и $2N$ ординатами сторон заданных $N$ прямоугольников, и квадродерево $T$ строится на этой сетке. Каждый узел $v$ из $T$ обладает одним целочисленным параметром $C[v]$, который вначале равен 0 (такое квадродерево называется *скелетным*). Прямоугольник $R$ вносит 1 в $C[v]$ тогда и только тогда, когда (1) $R$ содержит квадрат, относящийся к $v$, и (2) ни один из предков $v$ в $T$ не обладает таким же свойством. Очевидно, что таким образом определяется однозначный способ разбиения $R$ на квадраты, каждый из которых связан с одним узлом $v$. В то время как в дереве отрезков каждый отрезок разбит на не более чем



$O(\log N)$ интервалов, для квадродерева можно относительно просто показать [1]), что один прямоугольник разбивается на не более чем $O(N)$ квадратов. Поэтому процедуры вставки и удаления в квадродереве, а также определение меры объединения (площади) можно реализовать каждую за $O(N)$ операций. Отсюда следует, что последовательная корректировка сечений

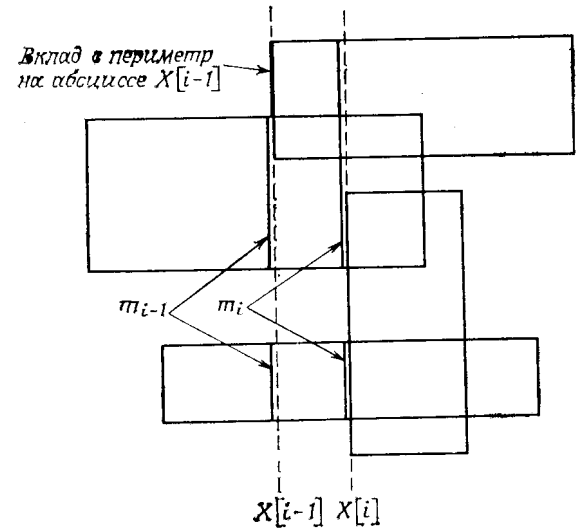



Рис. 8.10. Иллюстрация к вычислению суммарной длины вертикальных сторон.

при пространственном заметании занимает $O(N)$ времени, а поскольку существует $2N$ сечений, то решение задачи определения меры объединения в трехмерном пространстве займет в сумме $O(N^2)$ времени. Именно данный метод — а не двумерный, основанный на дереве отрезков, — можно использовать в качестве базиса индукции и доказать следующий результат:

**Теорема 8.2.** *Меру объединения $N$ изотетичных гиперпрямоугольников при $d \geqslant 3$ измерениях можно вычислить за время $O(N^{d-1})$.*

Хотя кажется, что улучшить этот результат будет весьма трудно, нет никаких данных о его возможной оптимальности.

**Замечание.** Приведенный алгоритм вычисления меры объединения прямоугольников можно модифицировать для решения следующей задачи.

[1]) Доказательство можно найти в статье [Van Leeuwen, Wood (1981)].

**Задача О.3** (ПЕРИМЕТР ОБЪЕДИНЕНИЯ ПРЯМОУГОЛЬНИКОВ). Дано объединение $F$, состоящее из $N$ изотетичных прямоугольников. Найти *периметр* $F$, т. е. длину его границы.

Метод решения основан на следующем наблюдении. Предположим, что на $i$-й итерации некий прямоугольник вставляется в дерево отрезков или удаляется из него (мы условились говорить, что прямоугольник вставляется или удаляется, если заметающая прямая касается его левой или правой стороны соответственно). Обращаясь к рис. 8.10, напомним, что $m_i$ обозначает суммарную длину вертикального сечения непосредственно слева от $X[i]$ (см. строку 6 алгоритма МЕРА-ОБЪЕДИНЕНИЯ-ПРЯМОУГОЛЬНИКОВ). Суммарная длина вертикальных кусков границы $F$ на сечении $X[i-1]$ равна разности длин вертикальных сечений $F$ непосредственно слева и справа от $X[i-1]$. В то время как первая из этих длин равна $m_{i-1}$ по определению, вторая равна $m_i$, поскольку сечение неизменно на всем интервале $[X[i-1], X[i]]$. Поэтому искомая суммарная длина равна $|m_i - m_{i-1}|$[1]). Кроме того, необходимо учесть вклад горизонтальных граничных сторон, расположенных внутри вертикальной полосы. В интервале $[X[i-1], X[i]]$ этот вклад, очевидно, равен $(X[i]-X[i-1])\times\alpha_i$, где целое число $\alpha_i$ равно количеству горизонтальных сторон границы $F$ внутри рассматриваемой вертикальной полосы. Параметр $\alpha_i$, по сути, весьма похож на $m_i$. Действительно, введем для каждого узла $v$ в дереве отрезков три новых специальных параметра, один четный целый параметр $\alpha[v]$ и два двоичных параметра ЛКР$[v]$ и ПКР$[v]$[2]). Обозначая через $\mathcal{I}$ текущее вертикальное сечение $F$ ($\mathcal{I}$ — набор не связанных между собой интервалов), определяем эти параметры следующим образом:

$\alpha[v]$ := удвоенное число компонент связности $\mathcal{I} \cap [B[v], E[v]]$;

ЛКР$[v]$ := 1 или 0 в зависимости от того, будет или нет $B[v]$ нижним концом интервала из $\mathcal{I} \cap [B[v], E[v]]$;

ПКР$[v]$ := 1 или 0 в зависимости от того, будет или нет $E[v]$ верхним концом интервала из $\mathcal{I} \cap [B[v], E[v]]$.

Эти три специальных параметра вначале равны нулю, а их вычисление выполняет нижеследующая конкретизация подпрограммы КОРРЕКТИРОВАТЬ, вызываемой в строке 4 процедуры ВСТАВИТЬ (см. разд. 8.2):

```
        procedure КОРРЕКТИРОВАТЬ(v);
1.  begin if (C[v] > 0) then
2.        begin α[v] := 2;
3.              ЛКР[v] := 1;
4.              ПКР[v] := 1
          end
5.        else begin α[v] := α[ЛСЫН[v]] + α[ПСЫН[v]] —
                                 —2 · ПКР[ЛСЫН[v]] ЛКР[ПСЫН[v]];
6.                   ЛКР[v] := ЛКР[ЛСЫН[v]];
7.                   ПКР[v] := ПКР[ПСЫН[v]]
              end
    end.
```

Корректность процедуры КОРРЕКТИРОВАТЬ доказывается легко. Если $C[v] \geqslant 1$, т. е. интервал $[B[v], E[v]]$ полностью накрыт, то в составе $\mathcal{I} \cap [B[v], E[v]]$ будет ровно один элемент, причем $\alpha[v] = 2$, ЛКР$[v]$ = ПКР$[v]$ = 1, как указано в строках 2—4. Если же $C[v] = 0$, то число элементов в $\mathcal{I} \cap [B[v], E[v]]$ равно сумме чисел таких элементов для двух потомков узла $v$, за исключением того случая, когда сечение $\mathcal{I}$ содержит один интервал, соединяющий точку $E[$ЛСЫН$[v]] = B[$ПСЫН$[v]]$; в последнем случае ПКР$[$ЛСЫН$[v]] =$ ЛКР$[$ПСЫН$[v]] = 1$. Этим доказывается корректность строк 5—7 алгоритма. Кроме того, не составит большого труда показать, что дополнительные параметры $\alpha$, ЛКР и ПКР вы-

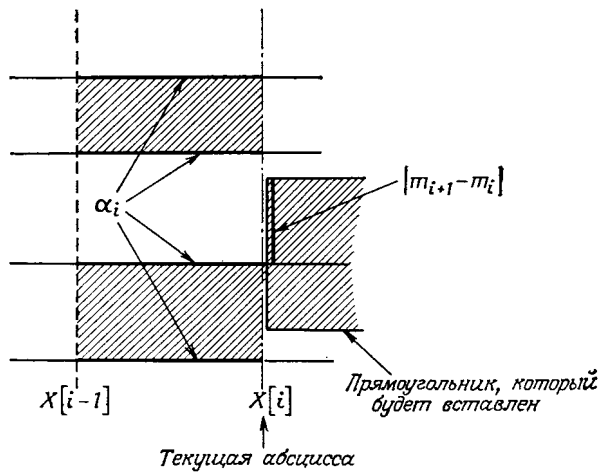



Рис. 8.11. Вклад в величину периметра на текущем шаге состоит из длин горизонтальных и вертикальных ребер.

[1]) Заметим, что возможно равенство $X[i-1]$ и $X[i]$, т. е. две вертикальные стороны имеют одинаковые абсциссы. В этом случае введем для них удобное лексикографическое упорядочение (например, по ординатам их нижних концов) и представим себе, что две такие стороны фиктивно раздвинуты на величину $\varepsilon$ в направлении оси $x$.

[2]) ЛКР и ПКР — аббревиатуры выражений «левый край» и «правый край». — *Прим. перев.*

числяются за константное время в расчете на один пройденный узел.

Теперь очевидно, что $\alpha_i = \alpha$[корень $(T)$]. Следовательно, алгоритм вычисления периметра $F$ получается путем простой модификации соответствующего алгоритма вычисления меры этого объединения. Заметим, однако, что в то время, как в алгоритме вычисления меры объединения к площади $F$ добавляется площадь только что заметенной вертикальной полосы (так что корректировка дерева отрезков следует за суммированием этой площади), в обсуждаемом алгоритме ситуация несколько иная. Обратимся к рис. 8.11; вклад текущего шага в периметр состоит из двух частей: от горизонтальных ребер в полосе $[X[i-1], X[i]]$ он равен $\alpha_i \times (X[i] - X[i-1])$ и от вертикальных ребер на абсциссе $X[i]$ он равен $|m_{i+1} - m_i|$. Поэтому величину $\alpha_i$ необходимо извлечь из дерева отрезков до его корректировки, а величину $m_{i+1}$ — после. Итак, получаем:

```
        procedure ПЕРИМЕТР-ОБЪЕДИНЕНИЯ-
                  ПРЯМОУГОЛЬНИКОВ
begin   X[1 : 2N] := упорядоченная последовательность
        абсцисс вертикальных сторон;
        X[0] := X[1];
        m_0 := 0;
        p := 0;
        построить и инициализировать дерево T, состоящее
        из ординат сторон прямоугольников;
        for i := 1 until 2N do
            begin α* := α [корень(T)];
                  if (X[i] — абсцисса левого конца)
                  then ВСТАВИТЬ(b_i, e_i, корень(T))
                  else УДАЛИТЬ(b_i, e_i; корень(T));
                  m* := m[корень(T)];
                  p := p + α* × (X[i] − X[i − 1]) + | m* − m_0 |;
                  m_0 := m*
            end
end.
```

В заключение получаем:

**Теорема 8.3.** *Периметр объединения $N$ изотетичных прямоугольников можно вычислить за время $O(N \log N)$.*

## 8.5. Контур объединения прямоугольников

Тот же самый общий подход (плоское заметание, основанное на дереве отрезков) можно с успехом применить для решения другой интересной задачи: определения контура



объединения $F$, состоящего из $N$ изотетичных прямоугольников $R_1, \ldots, R_N$. Вновь $F = R_1 \cup \ldots \cup R_N$ может состоять из одной или более не пересекающихся компонент связности, а каждая компонента может содержать или не содержать внутренние дыры (заметим, что дыра может содержать внутри себя некоторые из компонент связности $F$). Контур (граница) $F$ состоит из набора непересекающихся циклов, образованных (чередующимися) вертикальными и горизонтальными ребрами. Условимся, что любое ребро ориентировано так, что фигура расположена слева от него; другими словами, цикл ориентирован по часовой стрелке, если он ограничивает дыру, и против часовой стрелки, если он является внешней границей компоненты связности. Мы имеем:

**Задача О.4** (КОНТУР ОБЪЕДИНЕНИЯ ПРЯМОУГОЛЬНИКОВ). Дан набор из $N$ изотетичных прямоугольников. Требуется определить контур их объединения.

Во-первых, заметим, что периметр (для вычисления которого в предыдущем разделе был описан довольно простой алгоритм) является не чем иным, как длиной искомого контура. Однако мы увидим, что вычислить периметр значительно проще, чем сам контур, как набор непересекающихся циклов.

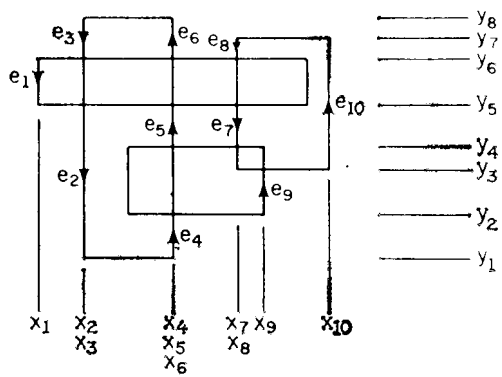



Рис. 8.12. Иллюстрация к задаче построения контура объединения прямоугольников.

Алгоритм лучше всего описать неформально, на примере. Он состоит из двух главных этапов. На первом этапе определяется множество $V$, состоящее из вертикальных ребер контура (ребра от $e_1$ до $e_{10}$ на рис. 8.12); на втором этапе эти вертикальные ребра соединяются горизонтальными ребрами для формирования ориентированных циклов контура [Lipski, Preparata (1980)].

Обозначим через $(x; b, t)$ вертикальное ребро с абсциссой $x$, для которого $b$ и $t$ $(b < t)$ являются соответственно нижней и верхней ординатами; аналогично $(y; l, r)$ обозначает горизонтальное ребро с ординатой $y$, для которого $l$ и $r$ $(l < r)$ являются соответственно левой и правой абсциссами.

Для получения множества $V$ просканируем слева направо абсциссы, соответствующие вертикальным сторонам прямо-

угольников. Для произвольной абсциссы $c$ сечение $\mathcal{I}(c)$ объединения $F$ вертикальной прямой $x = c$ является набором непересекающихся интервалов. Такое сечение не изменяется между двумя последовательными вертикальными сторонами прямоугольников, но корректируется при сканировании всякий раз, когда встречается такая сторона. Если $s$ является левой вертикальной стороной с абсциссой $c$ для какого-нибудь прямоугольника $R$, то участок контура $F$, определяемый $s$, задается $s \cap \overline{\mathcal{I}(c_-)}$, где $\mathcal{I}(c_-)$ обозначает объединение интервалов, лежащих непосредственно слева от абсциссы $c$, а $\overline{\mathcal{I}(c_-)}$ обозначает его теоретико-множественное дополнение. Аналогично, если $s$ — правая вертикальная сторона $R$ с абсциссой $c$, то вклад $s$ в контур равен $s \cap \overline{\mathcal{I}(c_+)}$; смысл обозначений очевиден. Запоминание и корректировка сечения $\mathcal{I}$ и эффективное определение $s \cap \overline{\mathcal{I}(c_-)}$ или $s \cap \overline{\mathcal{I}(c_+)}$ являются наиболее сложной частью алгоритма и будут рассмотрены позднее.

Предположим пока, что множество $V$ уже получено и в согласии с принятым условием об ориентации контура каждое ребро $V$ направлено вверх или вниз в зависимости от того, принадлежат его ординаты соответственно правой или левой стороне прямоугольника.

Заметим теперь, что если множество вертикальных ребер $V$ известно, то горизонтальные ребра можно определить прямым методом. Действительно, если $(y; x_1, x_2)$ — горизонтальное ребро (рис. 8.13), то существуют два вертикальных ребра с абсциссами $x_1$ и $x_2$ соответственно, причем у каждого из них один из концов имеет ординату $y$. Предположим далее, что для вертикального ребра $e_j = (x_j; b_j, t_j)$ из $V$ сформированы две тройки $\langle x_j, b_j; t_j\rangle$ и $\langle x_j, t_j; b_j\rangle$. Каждую из этих троек следует считать точкой (координаты которой совпадают с первым и вторым элементами тройки), в совокупности эти две точки являются концами ребра $e_j$; третий член каждой тройки является просто ссылкой на противоположный конец $e_j$. Упорядочим множество троек *лексикографически по возрастанию* (*сначала по ординате, затем по абсциссе*). Для примера на рис. 8.12 получается следующая последовательность:

$$\langle x_2, b_2; t_2\rangle\langle \boldsymbol{x_4}, b_4; t_4\rangle\langle x_4, t_4; b_4\rangle\langle x_9, b_9; t_9\rangle\langle x_9, t_9; b_9\rangle\langle x_{10}, b_{10}; t_{10}\rangle$$
$$\langle x_5, b_5; t_5\rangle\langle x_7, b_7; t_7\rangle\langle x_1, b_1; t_1\rangle\langle x_2, t_2; b_2\rangle\langle x_5, t_5; b_5\rangle\langle x_7, b_7; t_7\rangle$$
$$\langle x_1, t_1; b_1\rangle\langle x_3, b_3; t_3\rangle\langle x_6, b_6; t_6\rangle\langle x_8, b_8; t_8\rangle\langle x_8, t_8; b_8\rangle\langle x_{10}, t_{10}; b_{10}\rangle$$
$$\langle x_3, t_3; b_3\rangle\langle x_6, t_6; b_6\rangle.$$

Причина выбора такого расположения заключается в следующем. Пусть $a_1, a_2, \ldots, a_p$ ($p$ четно) представляет собой результирующую последовательность. Легко видеть, что горизонтальные ребра контура могут быть получены как отрезки, соединяющие вершины $a_{2k-1}$ и $a_{2k}$ для $k = 1, 2, \ldots, p/2$. Более точно, пусть $a_{2k-1} = \langle l, y: y_1\rangle$, $a_{2k} = \langle r, y: y_2\rangle$. Горизонтальное ребро $(y; l, r)$ ориентируется слева направо, если ребро, соответствующее тройке $\langle l, y; y_1\rangle$, ориентировано вниз и $y < y_1$ или если $\langle l, y; y_1\rangle$ ориентировано вверх[1]) и $y_1 < y$; иначе ребро $(y; l, r)$ ориентируется справа налево. Для вышеприведенного примера пара $(a_1, a_2) = (\langle x_2, b_2: t_2\rangle, \langle x_4, b_4: t_4\rangle)$ при $b_2 = b_4$ порождает горизонтальное ребро $(b_2; x_2, x_4)$, ориентированное слева направо, поскольку ребро $e_2$, соответствующее $\langle x_2, b_2; t_2\rangle$, направлено вниз и $b_2 < t_2$; напротив, пара $(a_{17}, a_{18}) = (\langle x_8, t_8; b_8\rangle, \langle x_{10}, t_{10}; b_{10}\rangle)$ при $t_8 = t_{10}$ порождает ребро $(t_8; x_8, x_{10})$, ориентированное справа налево, поскольку $e_8$ направлено вверх, а $t_8 \not< b_8$. Очевидно, что за один проход по последовательности $a_1, \ldots, a_p$ можно породить все горизонтальные ребра и дважды связать каждое из них с двумя смежными вертикальными ребрами. Результирующие списки дают в точности искомые циклы контура. Кроме того, если идентифицировать каждый цикл посредством его ребра с минимальной абсциссой, то направление такого ребра определяет, будет ли этот цикл границей дыры (тогда указанное ребро направлено вверх) или участком внешней границы (тогда указанное ребро направлено вниз).

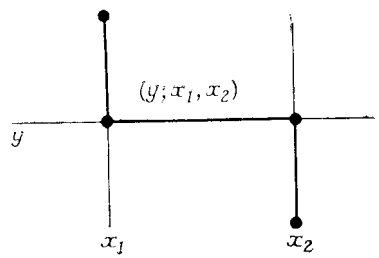



Рис. 8.13. Любое горизонтальное ребро смежно с парой вертикальных ребер.

Оценим теперь работу алгоритма. Читателя не должна беспокоить повторная операция сортировки $2p$ элементов (лексикографическое упорядочение, которое было описано ранее). Действительно, путем предварительной сортировки абсцисс и ординат сторон прямоугольников (за $O(N \log N)$ операций) можно нормализовать эти координаты и заменить их на последовательные целые числа. После этого $2p$ троек, которые нужно упорядочить лексикографически, будут состоять из целых чисел. Используя стандартный метод сортировки вычерпыванием [Knuth (1973)], получаем искомое упорядочивание за время $O(p)$; еще $O(p)$ времени используется для порождения горизонтальных ребер и контурных циклов.

[1]) Точнее сказать, вверх ориентировано ребро, соответствующее данной тройке.— *Прим. перев.*

Сосредоточим теперь внимание на эффективной реализации первого этапа алгоритма, т. е. на построении множества $V$ вертикальных ребер контура. Как и следует ожидать, предлагаемый метод является плоским заметанием, опирающимся на дерево отрезков $T$. Напоминая об общей схеме, введенной в разд. 8.3, укажем, что специальным узловым параметром, используемым в этом приложении, является СТАТУС$[v]$, дающий грубую классификацию меры $\mathscr{I} \cap [B[v], E[v]]$. А именно СТАТУС может принимать одно из следующих значений:

$$\text{СТАТУС}[v] = \begin{cases} \text{полон}, & \text{если } C[v] > 0; \\ \text{неполон}, & \text{если } C[v] = 0, \text{ но } C[u] > 0, \text{ где} \\ & u \text{ — один из потомков } v; \\ \text{пуст}, & \text{если } C[u] = 0 \text{ для любого } u \\ & \text{в поддереве с корнем в } v. \end{cases}$$

Согласно этому определению, текущее сечение $\mathscr{I}$ является объединением всех отрезков $[B[v], E[v]]$ для всех узлов дерева отрезков, чей СТАТУС полон.

Для заданного отрезка $s = (x; b, e)$ (вертикальной стороны прямоугольника) множество $s \cap \mathscr{I}$ показано на рис. 8.14; $s \cap \mathscr{I}$ является последовательностью *промежутков* между смежными элементами $\mathscr{I}$ в интервале $[b, e]$. Сторона $s$, помещенная в дерево отрезков, разбита на $O(\log N)$ фрагментов хорошо известным способом.

Было бы удобно иметь $s \cap \mathscr{I}$ в форме объединения вкладов от каждого из фрагментов. (Напомним, что каждый такой фрагмент соответствует узлу $v$ дерева $T$, для которого $b \leqslant B[v] < E[v] \leqslant e$.) Легко убедиться, что вклад узла $v$ в $s \cap \mathscr{I}$ (*вклад* $(v)$), соответствующий одному из фрагментов $s$, задается так:

$$\begin{aligned} &\textit{вклад}\ (v) = \\ &[B[v], E[v]] \cap \mathscr{I} = \end{aligned} \begin{cases} \varnothing, & \text{если СТАТУС}[v] \text{ полон} \\ & \text{или СТАТУС}[u] \\ & \text{полон для какого-} \\ & \text{нибудь предка } v \text{ в } T; \\ [B[v], E[v]], & \text{если СТАТУС}[v] \text{ пуст;} \\ \textit{вклад}\,(\text{ЛСЫН}[v]) \cup \textit{вклад}\ (\text{ПСЫН}[v]), & \\ & \text{если СТАТУС}[v] \\ & \text{неполон.} \end{cases}$$

Отсюда следует, что поиск в поддереве с корнем $v$ нужно проводить, только если СТАТУС$[v]$ неполон. Позже мы вернемся к реализации этого поиска.

Предположим временно, что последовательность (*вклад*$(v)$, где $v$ соответствует фрагменту $s$) уже получена; для того чтобы



каждое контурное ребро получалось как единый элемент, смежные интервалы необходимо объединить. Эта ситуация показана на рис. 8.15. Там отрезок $s = (x; b, e)$ разбит деревом $T$ на пять фрагментов, каждый из которых соответствует одному

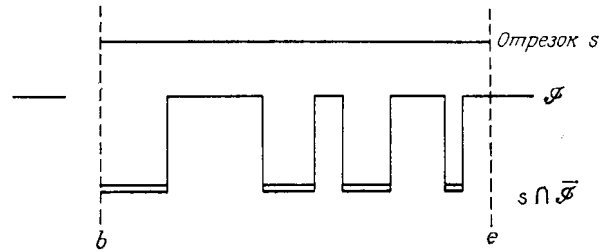



Рис. 8.14. Пример множества интервалов $s \cap \bar{\mathscr{I}}$. (Заметим, что $s$ является горизонтальным отрезком.)

из узлов $T$. Каждый из таких узлов порождает, вообще говоря, набор интервалов из $a \cap \bar{\mathscr{I}}$. Любые два смежных интервала, которые порождены разными узлами дерева отрезков, должны быть объединены. Для реализации этого интервалы из *вклад*$(v)$ помещаются в СТЕК, что соответствует проходу снизу вверх

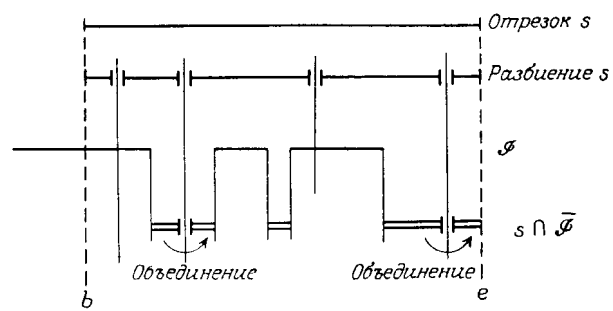



Рис. 8.15. Объединение смежных интервалов, производимое при вычислении $s \cap \bar{\mathscr{I}}$. (Вновь $s$ изображается как горизонтальный отрезок.)

по плоской фигуре. На вершине СТЕКА всегда будет располагаться верхний конец последнего из вставленных интервалов. Если нижний конец следующего интервала, который должен быть добавлен к СТЕКУ, совпадает с вершиной СТЕКА, то последняя удаляется из СТЕКА прежде, чем верхний конец следующего отрезка будет добавлен к нему, реализуя тем самым желаемое объединение.

Теперь можно описать поиск промежутков в $\mathscr{I}$ на отрезке $s = (x; b, e)$. В оригинальной работе Липски и Препараты использовалась стандартная структура данных и подпрограмма

ВКЛАД$(b, e;$ корень $(T))$, которая вычисляет множество $[b, e] \cap$ $\cap \bar{\mathscr{I}}$ на общей для них абсциссе.

```
        procedure КОНТУР-ОБЪЕДИНЕНИЯ-
                  ПРЯМОУГОЛЬНИКОВ[1])
  begin X[1 : 2N] := упорядоченная последовательность абсцисс
        вертикальных сторон; (* см. комментарий после
        процедуры *)
        𝒜 := ∅; (* 𝒜 — множество вертикальных ребер *)
        построить и инициализировать дерево отрезков T,
        состоящее из ординат сторон прямоугольников;
        for i := 1 until 2N do
            if (X[i] — абсцисса левой стороны) then
               begin 𝒜 := ВКЛАД(b_i, e_i; корень (T)) ∪ 𝒜;
                     ВСТАВИТЬ(b_i, e_i; корень(T))
               end
            else begin УДАЛИТЬ(b_i, e_i; корень(T));
                       𝒜 := ВКЛАД(b_i, e_i; корень(T)) ∪ 𝒜
                 end
  end.
```

Стоит подчеркнуть, что корректировка множества вертикальных ребер должна предшествовать вставке левой стороны, однако удаление правой стороны должно идти перед ней. Отсюда вытекает, что если какие-нибудь правая и левая стороны обладают одинаковой абсциссой, то обработка левой стороны должна предшествовать обработке правой стороны; поэтому этап сортировки в предыдущем алгоритме должен выполняться с учетом данного условия.

Осталось обсудить еще два момента. Первый связан с корректировкой специальных параметров (в данном случае СТАТУС$[v]$) в процедурах ВСТАВИТЬ и УДАЛИТЬ, второй — это подпрограмма ВКЛАД.

Что касается первого момента, то предлагается следующая процедура, которая, очевидно, выполняется за константное время в расчете на один просмотренный узел:

```
        procedure КОРРЕКТИРОВАТЬ(v)
  begin if (C[v] > 0) then СТАТУС[v] := полон
        else if (ЛСЫН[v] = Λ) then СТАТУС[v] := пуст
                                  (* v — лист *)
             else if (СТАТУС[ЛСЫН[v]] =
                      СТАТУС[ПСЫН[v]] = пуст)
                  then СТАТУС[v] := пуст
                  else СТАТУС[v] := неполон
  end.
```

[1]) Строго говоря, эта процедура реализует лишь первый этап построения контура объединения прямоугольников: она строит множество вертикальных ребер этого контура. — *Прим. перев.*



Теперь опишем подпрограмму ВКЛАД:

```
  function ВКЛАД(b, e; v)
```

(* данная подпрограмма использует внешний параметр СТЕК. В начале работы при вызове данной подпрограммы с параметрами ВКЛАД$(b, e;$ корень $(T))$ из главной процедуры СТЕК пуст. Подпрограмма заносит в СТЕК последовательность отрезков, представляющих собой $[b, e] \cap$ *вклад*$(v)$. Содержание СТЕКА возвращается при вызове ВКЛАД$(b, e;$ корень$(T))$ *)

```
    begin
1.    if СТАТУС[v] ≠ полон) then
2.    if (b ⩽ B[v]) and(E[v] ⩽ e) and(СТАТУС[v] = пуст) then
        (* вклад отрезка [B[v], E[v]] уже учтен *)
3.      begin if (B[v] = вершина (СТЕКА)) then (* объединение
                  смежных отрезков *)
4.              удалить вершину (СТЕКА)
5.            else СТЕК ⇐ B[v] (* начало ребра *);
6.            СТЕК ⇐ E[v] (* текущий конец ребра *)
        end;
7.    else begin if (b < ⌊(B[v] + E[v])/2⌋) then
8.              ВКЛАД(b, e; ЛСЫН[v]);
9.            if (⌊(B[v] + E[v])/2⌋ < e) then
10.             ВКЛАД(b, e; ПСЫН[v])
        end
    end.
```

Маршрут, пройденный процедурой ВКЛАД в дереве отрезков (т. е. последовательность обработанных ею узлов), существенно пересекается с аналогичной последовательностью, соответствующей процедуре ВСТАВИТЬ (или УДАЛИТЬ), однако есть два важных отличия, которые сейчас будут обсуждены на примере процедуры ВСТАВИТЬ (без потери общности). Хорошо известно, что маршрут процедуры ВСТАВИТЬ имеет типичную структуру, которая уже изучалась в разд. 1.2.3.1 и воспроизведена на рис. 8.16 для удобства читателей). Начальный путь (возможно, пустой) $P_{\text{нач}}$ приводит к узлу, именуемому *развилкой*, из которого выходят два (возможно, пустых) пути $P_{\text{л}}$ и $P_{\text{п}}$; изображено также множество *узлов отнесения*, определяющих сегментацию вставленного интервала. Подпрограмма ВКЛАД тоже проходит $P_{\text{нач}}$, $P_{\text{л}}$ и $P_{\text{п}}$; однако если какой-нибудь узел на любом из этих путей имеет полный СТАТУС, то проход по этому пути прерывается (строка 1 в подпрограмме ВКЛАД). Это происходит потому, что $y$-интервал отрезка $s$ (или участок этого интервала) «перекрывается» $\mathscr{I}$ и его вклад в $s \cap \bar{\mathscr{I}}$ пуст. Когда СТАТУС любого назначенного узла $v$, которого достигла подпрограмма ВКЛАД, не является полным, тогда, если он пуст,

вклад всего отрезка $[B[v], E[v]]$ уже учтен; в противном случае ВКЛАД начинает поиск в поддереве с корнем $v$ (строки 7—10) (это происходит, конечно, только если СТАТУС$[v]$ неполон). Этот поиск является наиболее времяемкой частью всей задачи, как будет сейчас показано. На рис. 8.16(b) видно, что для каждого отрезка $u$ из $\bar{\mathcal{T}} \cap [B[v], E[v]]$ существуют два уникальных узла в поддереве с корнем $v$, которые соответствуют самому левому и самому правому участкам $u$. Простой

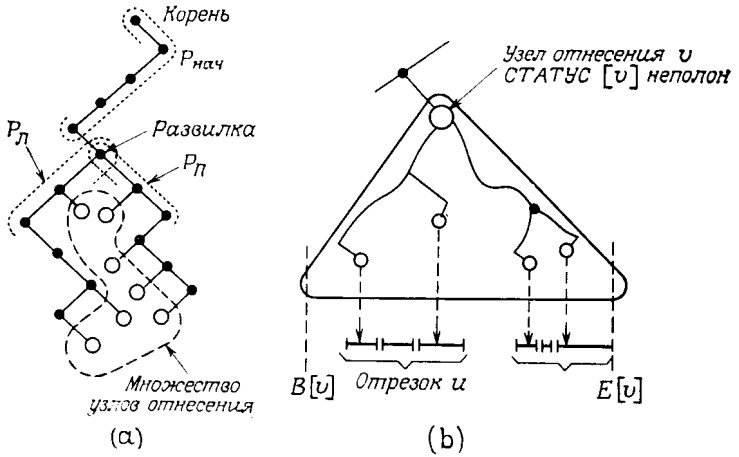



Рис. 8.16. (a) — типичная подструктура в $T$, пройденная процедурой ВСТАВИТЬ или УДАЛИТЬ; (b) — для каждого узла отнесения с неполным СТАТУСОМ, достигнутого подпрограммой ВКЛАД, производится последующий проход по всем путям, ведущим к узлам, которые соответствуют концам отрезков из $\bar{\mathcal{T}} \cap [B[v], E[v]]$.

анализ показывает, что подпрограмма ВКЛАД выполняет прохождение в прямом порядке путей, ведущих из $v$ к указанным узлам, при этом затрачивается фиксированное количество работы на каждый пройденный узел и, возможно, на его братьев по дереву. Поэтому суммарная работа пропорциональна сумме длин этих путей. Согласно общему свойству двоичных деревьев [Lipski, Preparata (1980)], если в поддереве существует $\nu$ концевых узлов, то искомая суммарная длина путей ограничена величиной $\nu \log(16N/\nu)$. Обозначим через $n_i$ число отдельных кусков в $s_i \cap \bar{\mathcal{T}}$, где $s_i$ — это $i$-я сторона прямоугольника, которую обрабатывает данный алгоритм. Тогда общая работа подпрограммы ВКЛАД ограничена величиной

$$C \sum_{i=1}^{2N} n_i \log \frac{16N}{n_i} \leqslant Cp \log \frac{32N^2}{p},$$



где $p$ обозначает суммарное число ребер контура, а $C$ — константа. Объединяя последнюю оценку с результатами анализа процедур ВСТАВИТЬ и УДАЛИТЬ, а также с оценкой работы второго этапа данного метода (на котором соединяются горизонтальные и вертикальные ребра), получаем следующую теорему:

**Теорема 8.4.** *Контур объединения $N$ изотетичных прямоугольников, состоящий из $p$ ребер, можно получить за время $O(N \log N + p \log(N^2/p))$.*

Если поставить вопрос об оптимальном алгоритме, то заметим, что известна нижняя оценка для этой задачи, равная $\Omega(N \log N + p)$. Действительно, $O(p)$ — тривиальная оценка, следующая из размера контура, а $\Omega(N \log N)$ — это нижняя оценка, полученная преобразованием СОРТИРОВКИ к построению контура (свободного от дыр) объединения прямоугольников, как вытекает из следующей теоремы:

**Теорема 8.5.** *Сложность построения контура для $F = R_1 \cup R_2 \cup \ldots \cup R_N$, где $F$ не содержит дыр, равна $\Omega(N \log N)$ при работе с моделью дерева решений.*

*Доказательство.* Покажем, что задача сортировки $N$ чисел $x_1, \ldots, x_N$ преобразуется за время $O(N)$ в нашу задачу. Действительно, пусть для заданного $x_i$ $R_i$ будет декартовым произведением $x$-интервала $[0, x_i]$ и $y$-интервала $[0, M - x_i]$, где

$$M = \max_{1 \leqslant i \leqslant N} x_i + 1$$

(рис. 8.17; без потери общности предположим, что $x_1, \ldots, x_N > 0$). Ясно, что $F = R_1 \cup \ldots \cup R_N$ не имеет дыр и что по контуру $F$ можно получить упорядоченную последовательность $\{x_i\}$ за время $O(N)$.

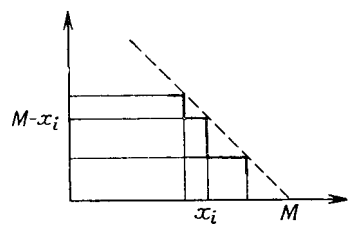



Рис. 8.17. Преобразование процедуры СОРТИРОВКА к задаче поиска контура объединения прямоугольников, которое не имеет дыр.

Для предыдущего алгоритма не удалось добиться верхней оценки, которая бы совпадала с нижней. Действительно, нетрудно обнаружить некоторую неэффективность поиска в поддеревьях для узлов отнесения с неполным статусом, поскольку затраты на прохождение по длинным путям приписываются только их конечным узлам. Это ограничение было преодолено Гютингом [Güting (1984)], построившим изумительную структуру данных для перечисления дыр в каждом поддереве за время, пропорциональное их числу, и тем самым создавшим

асимптотически оптимальный алгоритм. Как и следовало предполагать, структура данных Гютинга чрезвычайно сложна в работе. За описанием метода Гютинга читатель может обратиться к его статье.

## 8.6. Замыкание объединения прямоугольников

Начнем с того, что дадим строгое определение замыканию объединения прямоугольников. Говорят, что две точки $p_1 = (x_1, y_1)$ и $p_2 = (x_2, y_2)$ на плоскости *несравнимы*, если они не связаны отношением «доминирования» (см. разд. 4.1.3), т. е. если $(x_1 - x_2)(y_1 - y_2) < 0$. Если предположить без потери общности, что $x_2 > x_1$, то, следуя удобной терминологии [Soisalon-Soininen, Wood (1982)], назовем ЮЗ- и СВ-*сопряжениями* для $p_1$ и $p_2$ две точки $q_1 = (x_1, y_2)$ и $q_2 = (x_2, y_1)$ соответственно (рис. 8.18). Если задана плоская область $R$ (не обязательно связная), то говорят, что две точки *связаны* в $R$, если существует кривая, полностью расположенная в одной из компонент связности $R$ и соединяющая эти две точки. Введем теперь следующее определение:

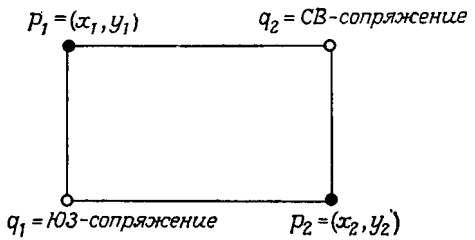



Рис. 8.18. Две точки $p_1$ и $p_2$, несравнимые по доминированию, и сопряженные им точки $q_1$ и $q_2$.

**Определение 8.2.** Плоская область $R$ называется СВ-*замкнутой*, если для любых двух несравнимых точек $p_1$ и $p_2$, которые связаны в $R$, их СВ-сопряжения тоже расположены в $R$. Аналогично определяется и ЮЗ-*замкнутая* плоская область.

**Определение 8.3.** СВ-*замыканием* плоской области $S$ называется наименьшая из СВ-замкнутых областей, содержащих $S$; она обозначается СВ($S$). Аналогично определяется и ЮЗ-*замыкание* $S$, обозначаемое ЮЗ($S$). СВЮЗ-*замыканием* (или просто *замыканием* для краткости) называется наименьшая область, содержащая $S$ и такая, что она СВ-замкнута и ЮЗ-замкнута одновременно; она обозначается СВЮЗ($S$).

Теория замыкания была развита в работах [Yannakakis, Papadimitriou, Kung (1979); Lipski, Papadimitriou (1981); Soisalon-Soininen, Wood (1982)]. Для дальнейшего изложения нам необходимо подчеркнуть некоторые важные свойства изучаемых объектов.



Говорят, что плоская кривая Г является *x-монотонной*, если для любых двух точек $(x_1, y_1)$ и $(x_2, y_2)$, лежащих на Г, справедливо условие: $x_2 > x_1 \Rightarrow y_2 \geqslant y_1$. Если на плоскости задана запретная область $D$, то две плоские кривые $\Gamma_1$ и $\Gamma_2$ называются *гомотопными*, если их можно совместить непрерывной деформацией, не пересекая $D$.

Если обозначить через $R(F)$ минимальный изотетичный прямоугольник, охватывающий заданную (не обязательно связную) плоскую область $F$, то будут справедливы следующие очевидные соотношения:

$$F \subseteq \text{СВЮЗ}(F) \subseteq R(F).$$

Если СВЮЗ($F$) связно, то легко видеть, что СВ- и ЮЗ-угловые точки $R(F)$ (рис. 8.19(a)) принадлежат этому замыканию $F$. Кроме того, для любой связной компоненты $G$ из СВЮЗ($F$)

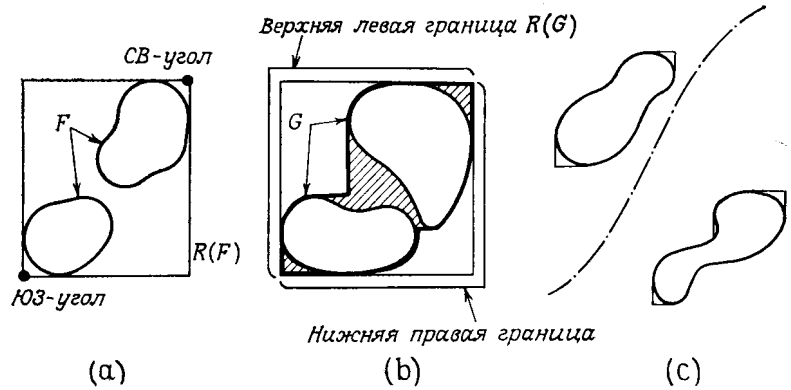



Рис. 8.19. (a) — плоская фигура $F$ и минимальный охватывающий ее (координатно-ориентированный. — *Перев.*) прямоугольник $R(F)$; (b) — связная компонента $G$ СВЮЗ-замыкания плоской фигуры $F$; (c) — разделимость двух компонент связности СВЮЗ($F$).

(рис. 8.19(b)) граница СВЮЗ($F$) состоит из двух $x$-монотонных кривых, гомотопных соответственно верхнему левому и нижнему правому участкам границы $R(G)$. Каждая из указанных двух $x$-монотонных кривых является экстремальной в собственном классе гомотопных кривых, не пересекающих $G$. Поэтому эти кривые можно называть *верхним* и *нижним* контурами замкнутой области СВЮЗ($F$). В общем случае для произвольной области $F$ замыкание СВЮЗ($F$) состоит из одной или более компонент; если число таких компонент больше единицы, то любые две из них можно разделить $x$-монотонной кривой (иначе будет нарушено условие замкнутости) (рис. 8.19(c)).

Сойсалон-Сойнинен и Вуд установили также следующее важное свойство, справедливое для любой плоской фигуры $F$:

$$\text{СВЮЗ}(F) = \text{СВ}(\text{ЮЗ}(F)) = \text{ЮЗ}(\text{СВ}(F)). \tag{8.1}$$

Для практических приложений наиболее интересен случай, когда $F$ является объединением изотетичных прямоугольников:

**Задача О.5** (ЗАМЫКАНИЕ ОБЪЕДИНЕНИЯ ПРЯМОУГОЛЬНИКОВ). Дан набор из $N$ изотетичных прямоугольников. Надо построить замыкание их объединения.

Свойство (8.1) позволяет предложить два эквивалентных алгоритма для вычисления замыкания заданной фигуры $F$. Например, равенство $\text{СВЮЗ}(F) = \text{ЮЗ}(\text{СВ}(F))$ позволяет вычислить сначала СВ-замыкание $F$, а затем завершить решение этой задачи путем вычисления ЮЗ-замыкания для $\text{СВ}(F)$. Эти две задачи сильно упрощаются, когда $F$ является объединением изотетичных прямоугольников, поскольку верхний и нижний контуры любой компоненты $\text{СВЮЗ}(F)$ являются при этом $x$-монотонными ступенчатыми кривыми. Данный алгоритм будет работать следующим образом. Он состоит из двух плоских заметаний в противоположных направлениях. Первое из них, проводимое слева направо, строит СВ-замыкание для фигуры $F$ (объединения изотетичных прямоугольников), т.е. оно определяет СВ-замкнутые компоненты связности. Второе плоское заметание (справа налево) строит ЮЗ-замыкание для $\text{СВ}(F)$, т. е., начиная со связных компонент $\text{СВ}(F)$, оно определяет в результате компоненты $\text{ЮЗ}(\text{СВ}(F))$ (и тем самым $\text{СВЮЗ}(F)$).

Компонента $\text{СВ}(F)$ называется *активной* (при плоском заметании), если ее пересекает заметающая прямая; очевидно, что компонента $\text{СВ}(F)$ активна тогда и только тогда, когда она содержит по крайней мере один прямоугольник из $F$, пересекаемый заметающей прямой (он называется *активным прямоугольником*).

Во-первых, исследуем те две основные ситуации, которые встречаются при плоских заметаниях (для конкретности: речь будет идти о заметании слева направо, которое строит $\text{СВ}(F)$). Очевидно, что точками событий являются абсциссы вертикальных сторон прямоугольников. Если текущим событием является встреча с левой стороной, то производятся следующие действия: (1) инициализируется активная компонента, или (2) расширяется активная компонента, или (3) объединяются две или более активные компоненты; с другой стороны, завершение работы с активной компонентой (и ее деактивация) может произойти только при встрече с правой стороной какого-нибудь прямоугольника. Активная компонента характеризуется *активным интервалом* (вдоль заметающей прямой), который является плоской полосой, заключенной между максимальной ординатой для *всех* просмотренных к данному моменту прямоугольников, принадлежащих указанной компоненте, и минимальной ординатой среди *активных* в этот момент прямоугольников этой компоненты (типичная ситуация показана на рис. 8.20). Активный интервал играет важнейшую роль при определении типа одной из трех возможных ситуаций, которые возникают при



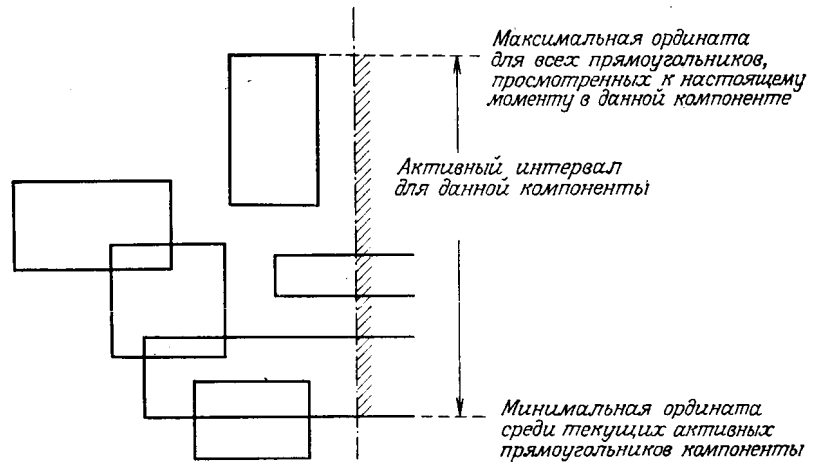



Рис. 8.20. Активная компонента и ее активный интервал. (Заштрихованная полоса будет использоваться здесь и далее для обозначения активных интервалов.)

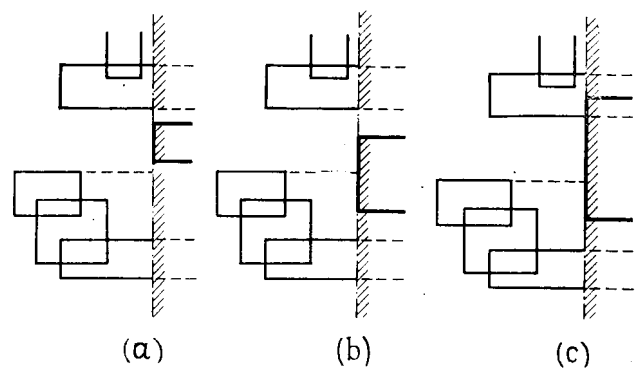



Рис. 8.21. Ситуации, которые могут возникнуть при встрече заметающей прямой с левой стороной. (a) — начинается новая компонента; (b) — расширяется одна из уже существующих компонент; (c) — две (или больше) компоненты соединяются. (Показаны активные интервалы *после* операции вставки.)

встрече заметающей прямой с левой стороной прямоугольника, как это показано на рис. 8.21.

Когда текущее событие является правой стороной какого-нибудь прямоугольника $R$, могут возникнуть также три ситуации: (1) $R$ перестает быть активным прямоугольником, не влияя на величину активного интервала (рис. 8.22(a)); или (2) $R$ перестает быть активным прямоугольником, а активный интервал поджимается кверху (рис. 8.22(b): это происходит, только если $R$ является *самым нижним* из активных прямоугольников данной

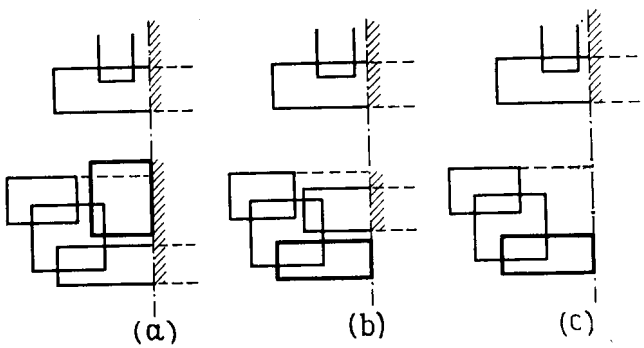



Рис. 8.22. Ситуации, которые могут возникнуть при встрече заметающей прямой с правой стороной прямоугольника. (a) — прямоугольник деактивируется, не влияя на величину активного интервала; (b) — прямоугольник деактивируется, приводя к сужению активного интервала; (c) — компонента завершается. (Вновь активные интервалы показаны *после* операции удаления.)

компоненты); или (3) компонента завершается (рис. 8.22(c): это происходит, только если $R$ оставался *единственным* активным прямоугольником в данной компоненте).

Для проведения вышеописанных действий статус заметающей прямой реализуется на несколько усложненной структуре данных. Первичная структура организована в форме сбалансированного по высоте дерева $T$, листья которого содержат концы активных интервалов. В каждом листе $T$, кроме нижнего конца активного интервала, хранится указатель на вторичную структуру, являющуюся также сбалансированным по высоте деревом, в котором хранятся все активные прямоугольники в данной компоненте. Вставки (левых сторон прямоугольников) и удаления (их правых сторон) выполняются следующим образом.

Когда нужно обработать левую сторону $[y_1, y_2]$, то сначала $y_1$ локализуется в $T$, а затем просматривается последовательность листьев до тех пор, пока не найдется также $y_2$. Эта операция исполняется за время $O(\log N + h)$, где $h$ — число объ-



единенных активных интервалов (рис. 8.21(c)); $y_2$ вставляется также и во вторичную структуру, и если $h \geqslant 2$, то соответствующие вторичные структуры сцепляются (как сцепляемые очереди). Затраты на конкатенацию вторичных структур за время работы алгоритма легко оценить, если $O(\log N)$ операций, затраченных на сцепление двух подобных структур $A_1$ и $A_2$, отнести на счет крайнего правого активного прямоугольника $A_1$; а поскольку на каждый прямоугольник такие затраты могут относиться не более одного раза, то верхняя оценка затрат на сцепление равна $O(N \log N)$.

Когда нужно обработать правую сторону $[y_1, y_2]$ вновь, сначала находим $y_1$ в активном интервале (в первичной структуре $T$). Затем элемент $y_1$ находится также в соответствующей вторичной структуре и удаляется из нее, а если он совпадает с минимальным элементом этой вторичной структуры, то корректируется и значение нижнего конца активного интервала (рис. 8.22(b)). Наконец, если $y_1$ является единственным элементом вторичной структуры, то соответствующий активный интервал удаляется также из первичной структуры. Очевидно, что обработку любой правой стороны можно выполнить за время $O(\log N)$.

Плоское заметание справа налево, при котором строится ЮЗ(СВ($F$)), проводится аналогично, поэтому справедлива

**Теорема 8.6.** СВЮЗ-*замыкание объединения $N$ изотетичных прямоугольников можно построить, затратив оптимальные время* $\theta(N \log N)$ *и память* $\theta(N)$.

*Доказательство* (набросок). Действительно, после предварительной сортировки абсцисс сторон прямоугольников обработка этих $2N$ сторон займет также время $O(N \log N)$, как показано выше. Структуры данных для этой работы занимают $O(N)$ памяти. Эти оценки оптимальны, поскольку задачу об уникальности элементов можно легко преобразовать в задачу о замыкании объединения прямоугольников.

**Замечание.** Вышеизложенный метод легко приспособить для вычисления связных компонент множества прямоугольников с сохранением таких же оценок по времени и памяти[1]). Действительно, легко убедиться, что все отличие состоит в замене (единственного) активного интервала для каждой компоненты (как описано выше) набором непересекающихся интервалов, представляющих собой сечения, образованные заметающей прямой и активными прямоугольниками из этой компоненты.

[1]) См. также [Edelsbrunner, Van Leeuwen, Ottman, Wood (1981)].

## 8.7. Внешний контур объединения прямоугольников

В предыдущем разделе было показано, как один основной метод — плоское заметание — можно приспособить для решения множества разных задач, связанных с объединением изотетичных прямоугольников, таких как вычисление их площади, периметра, замыкания и контура (задачи О.1, О.3, О.4 и О.5). Как указывалось раньше (см. разд. 8.5), контур объединения $N$ прямоугольников может иметь $\theta(N^2)$ ребер, однако это неверно для подмножества контура, задаваемого следующим определением:

**Определение 8.4.** *Внешним контуром* объединения изотетичных прямоугольников $F$ называется граница между $F$ и бесконечной областью плоскости.

Покажем теперь, что внешний контур имеет $O(N)$ ребер. Этот результат устанавливается легко, если использовать два интересных супермножества внешнего контура, первое из которых вводится следующим определением:

**Определение 8.5.** *Нетривиальным контуром* объединения изотетичных прямоугольников $F$ называется множество таких контурных циклов, каждый из которых содержит по меньшей мере одну вершину какого-нибудь из исходных прямоугольников.

Примеры этих трех объектов — контура, нетривиального контура и внешнего контура — даны на рис. 8.23. Из этого рисунка непосредственно видно, что внешний контур является подмножеством нетривиального контура.

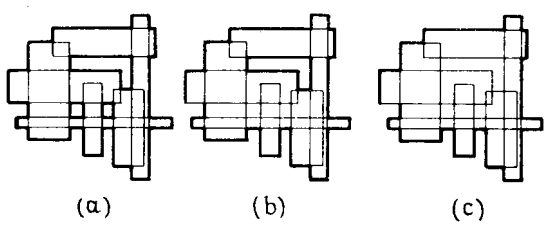



Рис. 8.23. Примеры: (a) — контура, (b) — нетривиального контура и (c) — внешнего контура для объединения прямоугольников.

Для дальнейшего изложения будет удобно считать каждое ребро состоящим из двух дуг, ориентированных в противоположных направлениях и лежащих по разные стороны от исходного ребра, как показано на рис. 8.24. Ребра прямоугольников $R_1, \ldots, R_N$ разбивают плоскость на области, одна из которых бесконечна. После введения вспомогательных дуг границы всех



областей этого разбиения порождают набор *ориентированных цепей*, состоящих из дуг; подобная цепь называется *внешней* или *внутренней* в зависимости от того, будет ли она ориентирована по часовой стрелке или против. Дуга в цепи называется

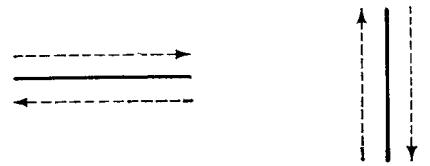


Рис. 8.24. Соглашения относительно направлений объединяющих дуг для заданного ребра.

*концевой*, если она содержит концевую точку того исходного отрезка (стороны прямоугольника), которому она принадлежит. Цепь называется *нетривиальной* или *тривиальной* в зависимости от того, содержит она или нет концевую дугу. Множество

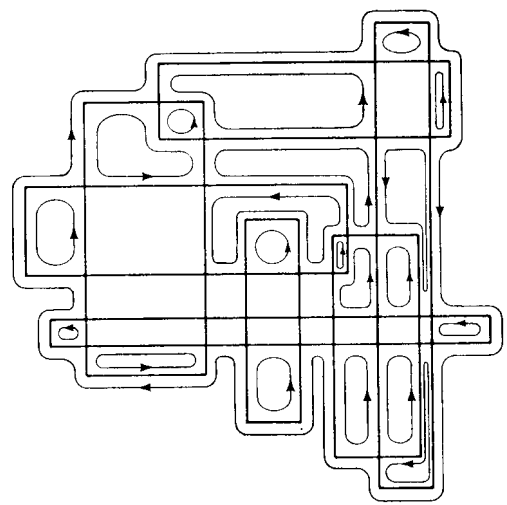


Рис. 8.25. Нетривиальные цепи для объединения прямоугольников, изображенного на рис. 8.23.

нетривиальных цепей для примера с рис. 8.23 проиллюстрировано на рис. 8.25. Теперь можно поставить следующие задачи:

**Задача О.6** (НЕТРИВИАЛЬНЫЙ КОНТУР ОБЪЕДИНЕНИЯ ПРЯМОУГОЛЬНИКОВ). Дан набор из $N$ изотетичных прямоугольников. Надо найти нетривиальный контур их объединения.

**Задача О.7** (ВНЕШНИЙ КОНТУР ОБЪЕДИНЕНИЯ ПРЯМОУГОЛЬНИКОВ). Дан набор из $N$ изотетичных прямоугольников. Надо построить внешний контур их объединения.

Непосредственно видна справедливость следующей цепочки включений:

Внешний контур $\subseteq$ Нетривиальный контур $\subseteq$ Нетривиальные цепи.

Важное свойство этих множеств состоит в том, что число дуг в нетривиальных цепях равно $O(N)$, как это следует из теоремы 8.7:

**Теорема 8.7.** *Общее число дуг в нетривиальных цепях объединения $N$ изотетичных прямоугольников равно $O(N)$.*

*Доказательство.* Присвоим вершине тип $i$ ($i = 1, 3$), если угол поворота по часовой стрелке между дугами, инцидентными этой вершине (и ориентированными согласно направлению

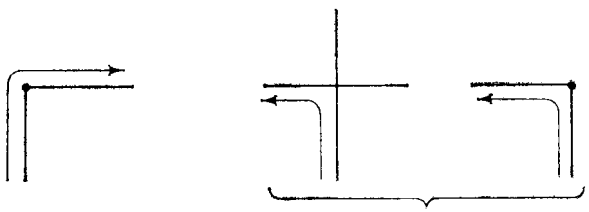



Рис. 8.26. Типы вершин.

цепи), равен $i\pi/2$ (рис. 8.26). Обозначим также через $v_i$ число вершин типа $i$ в заданной цепи. Тогда для любой цепи справедливо следующее соотношение:

$$v_3 - v_1 = \begin{cases} 4 & \text{для внутренней (ориентированной против часовой стрелки) цепи,} \\ -4 & \text{для внешней (ориентированной по часовой стрелке) цепи.} \end{cases}$$

Следовательно, общее число $v$ вершин в цепи (т. е. число дуг) выражается так:

$$v = v_1 + v_3 = 2v_1 \pm 4 \leqslant 2v_1 + 4.$$

Суммируя $v$ по всем нетривиальным цепям, получаем оценку для общего числа дуг $t$:

$$t \leqslant 2 \sum v_1 + 4 \sum 1 \leqslant 2 \cdot 4N + 4 \cdot 4N = 24N.$$



Данная теорема показывает, что если (общий) контур состоит из $O(N^2)$ дуг, то бо́льшая их часть принадлежит тривиальным цепям. Поэтому, если необходимо построить именно нетривиальный или внешний контуры, то придется разработать специальные методы, позволяющие избавиться от накладных расходов, создаваемых тривиальными цепями. Однако, прежде чем приступить к решению последней задачи, отметим, что нижней оценкой ее трудоемкости будет $\Omega(N \log N)$, поскольку в данной ситуации верны те же самые рассуждения, которые были использованы при доказательстве теоремы 8.5 (в разд. 8.5); действительно, в том доказательстве понятия общего, нетривиального и внешнего контуров совпадали.

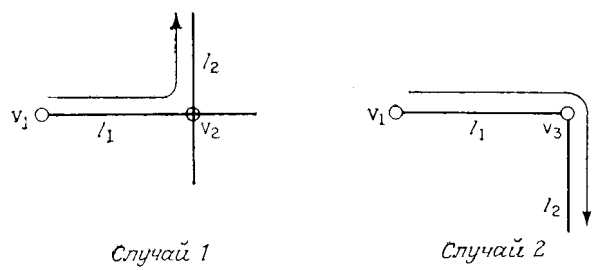



Рис. 8.27. Ситуации, возникающие в алгоритме на этапе движения.

Данную задачу вряд ли возможно решить методом плоского заметания, поскольку определить факт тривиальности цепи можно только при окончании процесса заметания (а не в его начале). Недавно был предложен метод [Lipski, Preparata (1981)], осуществляющий «обход» цепей того контура, который следует построить, добавляя по одной дуге на каждом шаге. Ясно, что если обработка каждой дуги потребует $O(\log N)$ времени, то, согласно теоремам 8.5 и 8.7, будет получен оптимальный алгоритм. Приступим к описанию этого метода.

Основной частью процедуры обхода является *механизм движения*, согласно которому мы отправляемся из *текущей вершины* $v_1$ (рис. 8.27) и продвигаемся по *текущему отрезку* $l_1$ в заданном направлении, причем может возникнуть одна из следующих двух ситуаций.

1. Существует такой отрезок $l_2$, *ближайший* к $v_1$, который пересекает $l_1$ и заходит внутрь области, лежащей слева от $l_1$. В этом случае делаем левый поворот, т. е. точка пересечения $v_2$ становится текущей вершиной, а $l_2$ — текущим отрезком.

2. Достигнута концевая точка $v_3$ отрезка $l_1$, совпадающая с концевой точкой отрезка $l_2$ (очевидно, $l_2$ не заходит внутрь области, лежащей слева от $l_1$). В этом случае делаем правый

поворот, т. е. $v_3$ становится текущей вершиной, а $l_2$ — текущим отрезком.

Вышеописанный шаг движения можно реализовать путем поиска на двух подходящих геометрических структурах, именуемых *горизонтальной и вертикальной картами смежности*, которые будут сейчас изложены. Ограничимся горизонтальной картой смежности (ГКС), поскольку все рассуждения в полной мере применимы к вертикальной карте смежности (ВКС).

Рассмотрим множество $V$, состоящее из вертикальных отрезков, которые являются вертикальными сторонами исходных прямоугольников (рис. 8.28). Через каждую концевую точку $p$

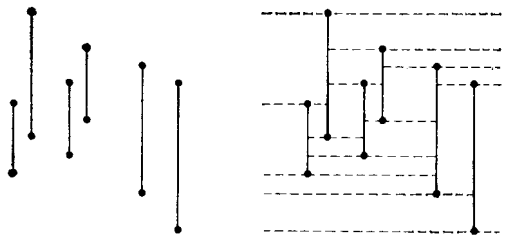



Рис. 8.28. Множество вертикальных отрезков $V$ и результирующая горизонтальная карта смежности.

каждого отрезка из $V$ проведем пару горизонтальных лучей: один вправо и один влево; каждый из этих лучей или заканчивается на ближайшем к $p$ вертикальном отрезке, или если подобного пересечения не существует, то луч продолжается в бесконечность. Таким образом плоскость разбивается на области, две из которых являются полуплоскостями, а все прочие — прямоугольниками, возможно бесконечными в одном или обоих горизонтальных направлениях. Эти области будем называть «обобщенными прямоугольниками». Каждый обобщенный прямоугольник является классом эквивалентности для точек на данной плоскости по отношению к их горизонтальной смежности вертикальным отрезкам (отсюда и название «горизонтальная карта смежности»). Можно простым индуктивным доказательством проверить то, что общее число областей в ГКС не превосходит $3|V|+1$.

Горизонтальная карта смежности является просто частным случаем разбиения плоскости, порожденного укладкой планарного графа. Если предположить, что текущий отрезок $l_1$ горизонтален (рис. 8.27), то локализация текущей вершины $v_1$ в одной из областей исходной карты (т. е. определение того обобщенного прямоугольника, который содержит $v_1$) означает определение вертикальных сторон этой области. Легко убедиться, что это все, что требуется для реализации шага движения. Действительно, предположим, что $v_1=(x_1, y_1)$, а $l_1$ лежит на прямой $y=y_1$ в интервале $[x_1, x_3]$. Локализуем точку $(x_1, y_1)$ в ГКС и получаем абсциссу $x_2$ для ближайшего вертикального отрезка, лежащего справа от $(x_1, y_1)$. Если $x_2 \leqslant x_3$, то надо сделать левый поворот (случай 1); если $x_2 > x_3$, то надо сделать правый поворот (случай 2). Три оставшихся случая расположения текущего отрезка (слева, сверху и снизу от текущей вершины) обрабатываются совершенно аналогичным образом. Поэтому добавка одной дуги к нетривиальной цепи занимает столько же времени, сколько одно обращение к любой из карт смежности.



Для реализации процедуры обращения к карте смежности (т. е. для локализации точки на плоскости) можно прибегнуть к одному из многих методов поиска, описанных в разд. 2.2, и с оценкой по времени $O(\log N)$. Особенно удобен для данной ситуации так называемый метод трапеций (на основе медиан) (разд. 2.2.2.4), причем благодаря природе исходной плоской карты (определенной множеством параллельных отрезков) указанные трапеции становятся прямоугольниками. Для $O(N)$ точек (как в нашем случае, поскольку число вершин исходных изотетичных прямоугольников равно $4N$) данный метод тратит на поиск $O(\log N)$ времени, используя структуру данных (двоичное дерево поиска), которую можно построить за время $O(N \log N)$. Нежелательным аспектом этого, в остальном весьма простого метода является то, что в худшем случае он требует $O(N \log N)$ памяти вместо $\theta(N)$. Однако вероятностный анализ [Bilardi, Preparata (1982)], надежно базирующийся на большом экспериментальном материале, показал, что для широкого класса гипотез относительно статистического распределения отрезков средний объем необходимой памяти равен $\theta(N)$ с очень малой, практически приемлемой мультипликативной константой (равной примерно 6).

Нетривиальный контур (и, следовательно, внешний контур) можно получить, систематически построив множество всех нетривиальных цепей, а затем удалив нежелательные его элементы. Порождение нетривиальных цепей можно осуществить в результате применения вышеописанного шага движения к вершине прямоугольника и той дуге, которая выходит из нее и направлена вовне прямоугольника. Тем самым каждая вершина прямоугольника становится «ростком цепи». Вначале все вершины занесены в память (массив) и помечены. Они извлекаются поочередно из указанного массива для генерации всех нетривиальных цепей, а этот процесс прекращается, когда массив становится пустым. Поскольку в нетривиальных цепях

имеется $O(N)$ дуг, а построение одной такой дуги можно проделать за время $O(\log N)$, то все множество нетривиальных цепей можно построить за время $O(N \log N)$, включая сюда и время предварительной обработки для построения деревьев поиска на картах смежности.

Для завершения построения внешнего контура необходимо удалить те цепи, которые не отделяют $F$ от его внешности. Чтобы отличить нетривиальные цепи, которые принадлежат внешнему контуру, от тех, которые не принадлежат ему, можно создать процедуру плоского заметания, требующую $O(N \log N)$ времени (см. упр. 1 в конце данной главы).

**Теорема 8.8.** *Нетривиальный и внешний контуры объединения из $N$ изотетичных прямоугольников можно построить за оптимальное время $\theta(N \log N)$, затратив в худшем случае $O(N \log N)$ памяти.*

## 8.8. Пересечения прямоугольников и связанные с этим задачи

В предыдущих разделах обсуждались методы вычисления некоторых глобальных свойств множества изотетичных прямоугольников, таких как мера, периметр и т. п. В настоящем разделе будет, напротив, рассматриваться задача вычисления двух отношений, которые естественным образом определены на множестве прямоугольников. Эти два отношения «пересечение» и «включение» (или, как иногда говорят, «охват») рассматриваются по отдельности в следующих разделах.

### 8.8.1. Пересечения прямоугольников

Два изотетичных прямоугольника пересекаются тогда и только тогда, когда у них есть по крайней мере одна общая точка. Простейшим примером является случай одного измерения, где «прямоугольники» — это интервалы на прямой линии. Поскольку $d$-мерный прямоугольник является декартовым произведением $d$ интервалов, каждый из которых определен на соответствующей координатной оси, то два $d$-мерных прямоугольника пересекаются тогда и только тогда, когда пересекаются их проекции (два интервала) на ось $x_j$ для всех $j = 1, 2, \ldots, d$. Итак, очевидно, что одномерный случай играет фундаментальную роль, поэтому он будет изучен прежде всего.

Пусть заданы два интервала: $R' = [x_1', x_2']$ и $R'' = [x_1'', x_2'']$; условие $R' \cap R'' \neq \varnothing$ эквивалентно одному из следующих взаимно исключающих условий:

$$x_1' \leqslant x_1'' \leqslant x_2'; \tag{8.2}$$

$$x_1'' \leqslant x_1' \leqslant x_2''. \tag{8.3}$$



Четыре возможных ситуации взаимного расположения концов интервалов, соответствующих условию $R' \cap R'' \neq \varnothing$, фактически охватываются соотношениями (8.2) и (8.3), что может быть проверено непосредственным образом. Поэтому для проверки пересечения $R'$ и $R''$ достаточно проверить факт попадания или левого конца $R'$ внутрь $R''$, или левого конца $R''$ внутрь $R'$.

Указанное свойство одномерной задачи играет важнейную роль при решении следующей задачи:

**Задача О.8** (ОТЧЕТ О ПЕРЕСЕЧЕНИИ ПРЯМОУГОЛЬНИКОВ). Дан набор из $N$ изотетичных прямоугольников. Надо перечислить все их пересекающиеся пары.

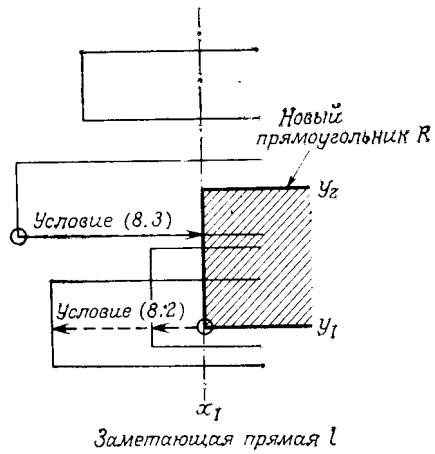



Рис. 8.29. Ситуация, возникающая при достижении заметающей прямой левой стороны прямоугольника $R$. Показаны только активные прямоугольники.

Для решения этой задачи представляется естественным обращение к методу плоского заметания, где точками событий являются, как обычно, абсциссы вертикальных сторон прямоугольников. Статус заметающей прямой задается пересечениями исходных прямоугольников с этой заметающей прямой; для упрощения формулировок назовем вновь прямоугольники, пересеченные заметающей прямой, *активными* (см. разд. 8.6). Предположим, что в процессе плоского заметания слева направо текущей точкой события стала левая сторона (вновь встреченного) прямоугольника $R = [x_1, x_2] \times [y_1, y_2]$. Очевидно, что абсцисса заметающей прямой $x_1$ (которая совпадает с левым концом $x$-интервала для $R$) принадлежит $x$-интервалам для каждого из активных прямоугольников; поэтому все, что необходимо сделать, состоит в определении того, для каких из активных прямоугольников выполнены условия или (8.2), или (8.3) на их $y$-интервалах (рис. 8.29). Кажется, что необходим двукратный поиск: один раз для проверки условия (8.2), а второй для проверки условия (8.3). Двойственная природа этих

двух проверок (одна для определения всех интервалов, содержащих некую точку; другая для определения всех точек, попавших в некий интервал) может навести на мысль об использовании двух разных структур данных и о разработке несколько более сложных алгоритмов [Bentley, Wood (1980); Six, Wood (1980)].

Превосходное решение было получено в результате введения [McCreight (1981); Edelsbrunner (1980)] новой структуры данных, названной Эдельсбруннером *деревом интервалов*. Хотя возможна и более совершенная динамическая версия интервального дерева, мы ограничимся простым статическим вариантом, пригодным для наших целей.

Если $[b^{(i)}, e^{(i)}]$ обозначает $y$-интервал для прямоугольника $R^{(i)}$, то пусть $(y_1, y_2, \ldots, y_{2N})$ обозначает упорядоченную последовательность концов $N$ штук $y$-интервалов. (*Статическое*) *дерево интервалов* имеет скелетную структуру (именуемую *первичной* структурой), которая статически зависит от заданной последовательности точек (в нашем случае от последовательности $(y_1, \ldots, y_{2N})$, хотя в ней можно хранить произвольное подмножество (активное подмножество) интервалов, чьи концы принадлежат множеству $\{y_1, y_2, \ldots, y_{2N}\}$. Тогда *дерево интервалов* $T$ на последовательности $(y_1, y_2, \ldots, y_{2N})$ и множестве интервалов $I \subseteq \{[b^{(i)}, e^{(i)}]: i = 1, \ldots, N\}$, определяется следующим образом:

1. Корень $w$ дерева $T$ имеет дискриминант $\delta(w) = (y_N + y_{N-1})/2$ и два указателя на (вторичные) списки $\mathscr{L}(w)$ и $\mathscr{R}(w)$. Списки $\mathscr{L}(w)$ и $\mathscr{R}(w)$ содержат упорядоченные списки соответственно левых и правых концевых точек тех интервалов из $I$, которые содержат $\delta(w)$. ($\mathscr{L}(w)$ и $\mathscr{R}(w)$ упорядочены по возрастанию и убыванию соответственно.)

2. Левым поддеревом для $w$ является дерево интервалов на последовательности $(y_1, \ldots, y_N)$ и подмножестве интервалов $I\mathscr{L} \subseteq I$, ординаты правых концов которых не превосходят $\delta(w)$. Аналогично определяется и правое поддерево для $w$.

3. Каждый первичный узел в $T$ считается или «активным», или «неактивным». Узел активен, если его вторичные списки непусты или если в обоих его поддеревьях есть активные узлы.

Статическое дерево интервалов обладает рядом интересных свойств, следующих прямо из вышеизложенного определения:

(1) Первичная структура $T$ является сбалансированным двоичным деревом, листья которого связаны с величинами $y_1, \ldots, y_{2N}$, и прохождение по $T$ во внутреннем порядке дает упорядоченную последовательность $(y_1, \ldots, y_{2N})$.

(2) На вторичных списках $\mathscr{L}(v)$ и $\mathscr{R}(v)$ для нелистового первичного узла $v$ должны быть определены операции вставки и удаления элементов. Эти списки удобно реализовать как сбалансированные по высоте или весу деревья.



(3) Активные узлы можно однозначно связать в структуру, представляющую собой двоичное дерево $\mathscr{T}$, каждая дуга которого соответствует пути (или его части), исходящему из корня $T$. Поэтому каждому первичному узлу $v$ припишем два указателя: ЛУ (левый указатель) и ПУ (правый указатель), используемые для реализации дерева $\mathscr{T}$. Если узел $v$ неактивен, то $\text{ЛУ}[v] = \Lambda$ и $\text{ПУ}[v] = \Lambda$; однако если $v$ активен, то $\text{ЛУ}[v] \neq \Lambda$ только тогда, когда существуют активные узлы в левом поддереве узла $v$, и аналогично $\text{ПУ}[v] \neq \Lambda$ только тогда, когда существуют активные узлы в правом поддереве $v$. Заметим, что больше половины активных узлов имеют непустые вторичные списки.

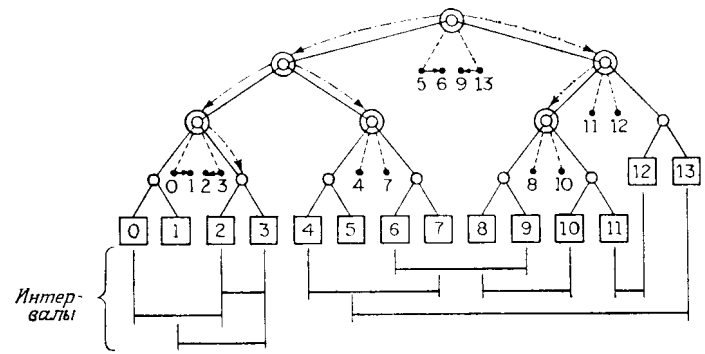



Рис. 8.30. Пример отнесения множества интервалов в статическом дереве интервалов. Дуги, обозначенные сплошными линиями, образуют первичную структуру; пунктирные дуги образуют вторичные структуры; штрих-пунктирные линии образуют особую сверхструктуру $\mathscr{T}$.

На рис. 8.30 приведен пример отнесения интервалов в статическом дереве интервалов.

Во-первых, рассмотрим вопрос об управлении вставками и удалениями в статическом дереве интервалов. Заметим, что как первичная структура, так и каждый из вторичных списков реализуются посредством двоичных деревьев глубины $O(\log N)$. Вставка интервала $[b, e]$ заключается в проходе пути в $T$ от корня до первого узла $v^*$ такого, что $b \leqslant \delta(v^*) \leqslant e$ (развилки); в этот момент ордината $b$ вставляется в $\mathscr{L}(v^*)$, а ордината $e$ — в $\mathscr{R}(v^*)$. Проверка того, что при этом левые и правые указатели можно поддерживать за константное время в расчете на один узел, является достаточно простым упражнением. По аналогии с незначительными изменениями можно выполнить и удаление. Каждая из этих операций требует время $O(\log N)$.

Теперь проанализируем поиск на интервальном дереве, т. е. определение пересекающихся пар интервалов при вставке интервала $[b, e]$ в $T$. Опять пройдем по некоему (возможно, пустому) пути от корня к первичному узлу $v^*$ (развилке) такому, что $b \leqslant \delta(v^*) \leqslant e$. Затем, начиная от $v^*$, пройдем по двум разным путям к двум листьям в поддереве с корнем в $v^*$, которые связаны с величинами $b$ и $e$ соответственно (рис. 8.31). Пусть

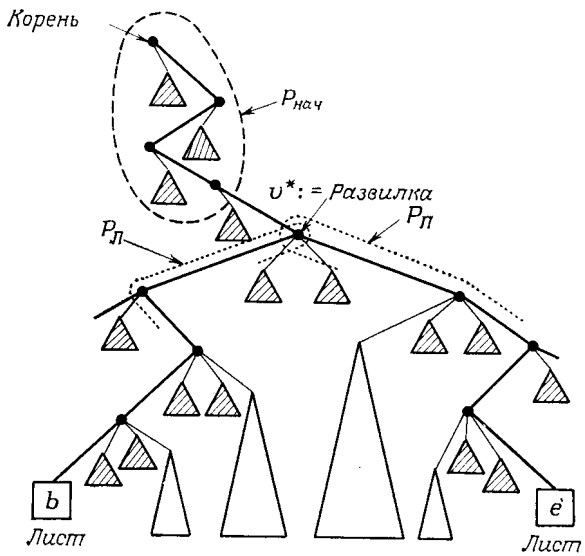



Рис. 8.31. Поиск в интервальном дереве. Закрашенные треугольники обозначают вторичные списки, которые следует обойти. Поименованные поддеревья также необходимо обойти (с помощью ЛУ и ПУ).

$P_{нач}$ обозначает последовательность узлов от корня до узла, предшествующего $v^*$; кроме того, пусть $P_л$ — последовательность узлов от $v^*$ до листа $b$, а $P_п$ — последовательность узлов от $v^*$ до $e$. Рассмотрим два основных случая:

1. $v \in P_{нач}$. В этом случае $[b, e]$ лежит слева или справа от $\delta(v)$. Предположим без потери общности, что $e \leqslant \delta(v)$. Поэтому прежде всего надо проверить возможность пересечения $[b, e]$ с какими-нибудь из активных интервалов, содержащих $\delta(v)$. Для любого такого активного интервала $[b^{(i)}, e^{(i)}]$ эта проверка производится путем выяснения справедливости условий $b \leqslant b^{(i)} \leqslant e$ или $b^{(i)} \leqslant b \leqslant e^{(i)}$. Но поскольку $b < e \leqslant \delta(v) < e^{(i)}$, то характеристическим условием пересечения интервалов $[b^{(i)}, e^{(i)}]$ и $[b, e]$ является только $b^{(i)} \leqslant e$; оно проверяется путем сканирования $\mathscr{L}(v)$ в порядке возрастания и перечисления всех тех интервалов, для которых это условие выполнено. Делается эта работа эффективно за время, *пропорциональное* числу найденных пересечений (без лишнего поиска).

2. $v \in P_л$ (и аналогично с соответствующими изменениями $v \in P_п$). Если $\delta(v) \leqslant b$, то рассуждая так же, как при $v \in P_{нач}$, необходимо просмотреть правый список узла $v$ — $\mathscr{R}(v)$ в убывающем порядке (опять без лишней работы). Если же $b < \delta(v)$, то известно, что $[b, e]$ пересекает не только *все* активные интервалы, отнесенные к $v$, но также и *все* интервалы, отнесенные к узлам из правого поддерева $v$. Первое из этих множеств получается простым перечислением тех интервалов, чьи правые концы находятся в $\mathscr{R}(v)$. Второе множество получается в результате обхода активных узлов правого поддерева. Для эффективного решения последней задачи будут использованы дуги, определяемые значениями ЛУ и ПУ. Согласно предыдущим рассуждениям (менее половины активных узлов в этом поддереве имеют пустые вторичные списки), число пройденных узлов по порядку будет равно числу перечисленных интервалов.



Анализ трудоемкости вышеописанного метода не составляет труда. Прежде всего статическое дерево интервалов для набора из $N$ интервалов требует $O(N)$ памяти, поскольку в нем будет $4N - 1$ первичных узлов, и еще не более $2N$ элементов необходимо запомнить во вторичных списках. Скелетная первичная структура строится за время $O(N \log N)$, поскольку необходима предварительная сортировка абсцисс. Каждый интервал вставляется или удаляется за время $O(\log N)$, как упоминалось выше. Поиск в дереве требует $O(\log N)$ времени для прослеживания поисковых путей (см. опять рис. 8.31), хотя на каждый обнаруженный интервал тратится константное время; единственные накладные расходы, относящиеся к соответствующему первичному узлу, пропорциональны числу вторичных списков, обрабатываемых в процессе данной работы. Заметим, кроме того, что если поиск в дереве осуществляется только в связи со вставкой прямоугольника (т. е. при вставке его левой стороны), то каждая пересекающаяся пара будет обнаружена ровно один раз. В заключение подведем итог предыдущему обсуждению следующей теоремой:

**Теорема 8.9.** *Обнаружить $s$ пересекающихся пар в наборе из $N$ изотетичных прямоугольников можно за оптимальное время $\theta(N \log N + s)$, затратив $O(N \log N)$ времени на предварительную обработку и используя оптимальную память $\theta(N)$.*

*Доказательство.* Верхние оценки по времени и памяти были установлены ранее. Поэтому ограничимся доказательством оп-

тимальности. Оптимальность по памяти очевидна. Что же касается оптимальности по времени, заметим, что задачу об уникальности элементов, для которой справедлива нижняя оценка $\Omega(N \log N)$ на модели дерева решений (см. гл. 5), можно тривиально преобразовать в нашу задачу. На самом деле пусть даны $N$ действительных чисел $\{z_1, \ldots, z_N\}$ и выбран интервал $[b, e]$; по $z_i$ построим прямоугольник $[b, e] \times [z_i, z_i]$ и применим к нему алгоритм поиска пересечений прямоугольников. Если пересечений не обнаружено, то все числа различны.

### 8.8.2. Другой подход к задаче о пересечении прямоугольников

Как отмечено в начале разд. 8.8.1, два интервала $R' = [x_1', x_2']$ и $R'' = [x_1'', x_2'']$ пересекаются, если выполнено любое из следующих условий:

$$(1) \quad x_1' \leqslant x_1'' \leqslant x_2'; \tag{8.2}$$

$$(2) \quad x_1'' \leqslant x_1' \leqslant x_2''. \tag{8.3}$$

Легко видеть, что дизъюнкция двух предыдущих условий эквивалентна конъюнкции двух следующих условий [1]):

$$x_1'' \leqslant x_2', \quad x_1' \leqslant x_2'',$$

которые можно тривиально преобразовать так:

$$-x_2' \leqslant -x_1'', \quad x_1' \leqslant x_2''. \tag{8.4}$$

Последнее условие выражает отношение «доминирования» $\prec$ на плоскости (см. разд. 4.1.3) для точек $(-x_1'', x_2'')$ и $(-x_2', x_1')$. В такой интерпретации задача определения всех пересекающихся пар интервалов из набора, в котором их $N$ штук, преобразуется следующим образом. Пусть задано множество $\mathscr{R} = \{R^{(j)} = [x_1^{(j)}, x_2^{(j)}]: j = 1, \ldots, N\}$, состоящее из $N$ интервалов; определим, во-первых, функцию $\sigma_0$: $\mathscr{R} \to E^2$, отображающую интервал $[x_1, x_2]$ в точку $(x_1, x_2)$ на плоскости (рис. 8.32; заметим, что точка $(x_1, x_2)$ лежит выше биссектрисы $x_1 = x_2$ первого квадранта). Теперь определим отображение $\sigma'$: $E^2 \to E^2$, которое является преобразованием симметрии плоскости относительно оси $x_2$ (т. е. точка $(x_1, x_2)$ отображается в точку $(-x_1, x_2)$). Наконец, определим отображение $\sigma''$: $E^2 \to E^2$, которое является преобразованием симметрии плоскости относительно биссектрисы $x_2 = -x_1$ второго квадранта (т. е. точка $(-x_1, x_2)$ отображается в точку $(-x_2, x_1)$). Для упрощения

[1]) Действительно, пусть $P_1 := (x_1' \leqslant x_1'')$, $P_2 := (x_1'' \leqslant x_2')$, $P_3 := (x_1'' \leqslant x_1')$ и $P_4 := (x_1' \leqslant x_2'')$. В символической записи: $P_1P_2 \vee P_3P_4 = T \Leftrightarrow (P_1 \vee P_3) \cdot (P_2 \vee P_3)(P_1 \vee P_4)(P_2 \vee P_4) = T$. Поскольку $\overline{P}_1 = P_3$, $\overline{P}_1 \vee P_4 = T$, $\overline{P}_3 \vee P_2 = T$, то имеем: $P_3P_4(P_2 \vee P_4) = P_3P_4 = T$.



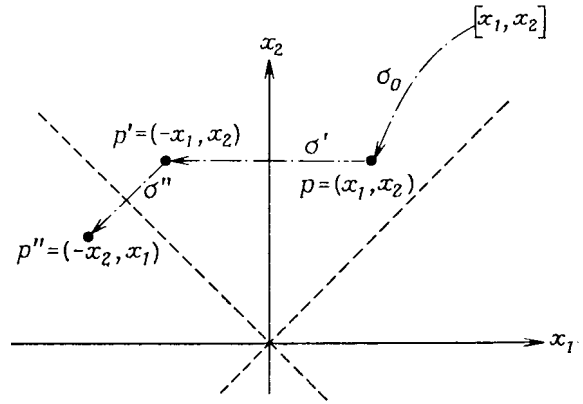



Рис. 8.32. Отображение интервала в две точки $p'$ и $p''$ на плоскости.

выражений обозначим через $p'$ и $p''$ композиции отображений $\sigma'\sigma_0$ и $\sigma''\sigma'\sigma_0$ соответственно. Убедимся в эквивалентности следующих соотношений, которая непосредственно следует из определений $p'$, $p''$ и (8.4):

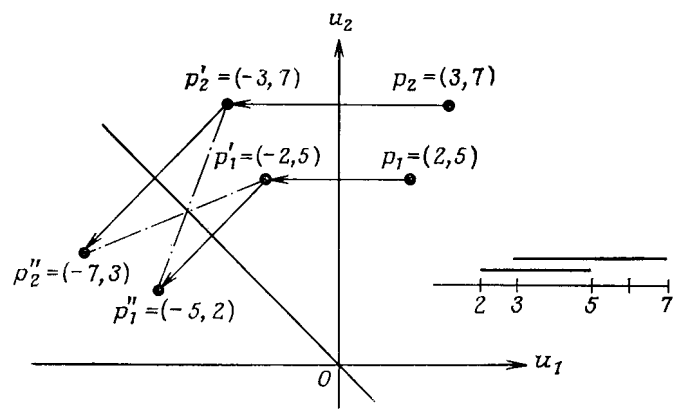



Рис. 8.33. Пары точек, связанных отношением доминирования; каждая пара эквивалентна пересечению интервалов.

$$R^{(i)} \cap R^{(j)} \neq \varnothing \Leftrightarrow p'(R^{(i)}) \succ p''(R^{(j)}) \Leftrightarrow p'(R^{(j)}) \succ p''(R^{(i)}).$$

Эта ситуация иллюстрируется на примере (рис. 8.33). Заметим, что каждая пара пересекающихся интервалов порождает *две* пары отношений доминирования.

Теперь обратим внимание на задачу О.8 (перечисление всех пересекающихся пар из множества прямоугольников $\mathscr{R}$ — двумерный случай). Пусть дан произвольный прямоугольник $R=[x_1, x_2]\times[y_1, y_2]$; сначала отобразим его в пару точек $p'(R)=(-x_1, x_2, -y_1, y_2)$ и $p''(R)=(-x_2, x_1, -y_2, y_1)$, лежащих в пространстве $E^4$ с координатными осями $u_1$ $u_2$, $u_3$, $u_4$. Затем сформируем два множества точек в $E^4$:

$$S'=\{p'(R): R\in\mathscr{R}\},$$
$$S''=\{p''(R): R\in\mathscr{R}\}.$$

Вышеизложенные соображения свидетельствуют о том, что два прямоугольника $R_1$ и $R_2$ из $\mathscr{R}$ пересекаются тогда и только тогда, когда $p'(R_1)\succ p''(R_2)$ (или, что то же самое, $p'(R_2)\succ p''(R_1)$). Поэтому задача О.8 о перечислении всех пересекающихся пар является частным случаем следующей общей задачи о доминировании [1]):

**Задача О.9** (СЛИЯНИЕ ДОМИНИРОВАНИЙ). Даны два множества точек $S_1$ и $S_2$ в $E^d$. Надо найти все пары $p_1\in S_1$ и $p_2\in S_2$ такие, что $p_1\succ p_2$.

О том, что наш случай является частным случаем этой общей задачи, свидетельствует тот факт, что у нас $S'$ и $S''$ разделены двумя гиперплоскостями $u_1+u_2=0$ и $u_3+u_4=0$ в четырехмерном пространстве. Применяя алгоритм, который будет описан в следующем разделе, можно решить задачу о перечислении всех пересекающихся пар за время $O(N\log^2 N+s)$ вместо оптимального времени $\theta(N\log N+s)$, затрачиваемого алгоритмом из разд. 8.8.1. Разрыв между этими двумя оценками, вероятно, следует скорее из неспособности использовать специфику нашей постановки этой общей задачи, а не из неадекватности общего алгоритма решения задачи о слиянии доминирований.

### 8.8.3. Охват прямоугольников

Задача об охвате (или о принадлежности или включении, как часто говорят) прямоугольников является в некотором смысле частным случаем задачи о пересечении и формулируется следующим образом:

**Задача О.10** (ОХВАТ ПРЯМОУГОЛЬНИКА). Дано множество $S$, состоящее из $N$ изотетичных прямоугольников на плоскости. Надо перечислить все такие упорядоченные пары элементов $S$, что первый элемент охватывает второй.

[1]) Заметим, что имеет место некоторая избыточность получающегося решения, и не только потому, что каждая искомая пара будет перечислена дважды, но еще и потому, что все тривиальные пары типа $(R, R)$ также будут перечислены.



(Заметим, что, в то время как пересечение является симметричным отношением, охват является отношением частичного порядка.)

Опять введем обозначение $\mathscr{R}=\{R^{(1)}, \ldots, R^{(N)}\}$. В раннем решении задачи об охватах в качестве структур данных использовались деревья отрезков и регионов [Vaishnavi, Wood (1980)]; соответствующий алгоритм тратит время $O(N\log^2 N+s)$ и использует $O(N\log N)$ памяти. Теперь покажем, что преобразование задачи об охвате прямоугольника к задаче о доминировании точек позволяет достигнуть оптимальной оценки по памяти $\theta(N)$, сохранив прежнюю оценку по времени [Lee, Preparata (1982)].

Обозначая, как обычно, $R^{(i)}=[x_1^{(i)}, x_2^{(i)}]\times[y_1^{(i)}, y_2^{(i)}]$, имеем, что $R^{(j)}\subseteq R^{(i)}$ тогда и только тогда, когда выполнены одновременно следующие четыре условия:

$$x_1^{(i)}\leqslant x_1^{(j)},\ x_2^{(j)}\leqslant x_2^{(i)},\ y_1^{(i)}\leqslant y_1^{(j)},\ y_2^{(j)}\leqslant y_2^{(i)}. \tag{8.5}$$

Очевидно, что эти условия эквивалентны следующим:

$$-x_2^{(j)}\leqslant -x_1^{(i)},\quad x_2^{(j)}\leqslant x_2^{(i)},\quad -y_1^{(j)}\leqslant -y_1^{(i)},\quad y_2^{(j)}\leqslant y_2^{(i)}, \tag{8.6}$$

которые выражают хорошо известное отношение «$\prec$» доминирования между двумя четырехмерными точками, т. е.

$$(-x_1^{(j)}, x_2^{(j)}, -y_1^{(j)}, y_2^{(j)})\prec(-x_1^{(i)}, x_2^{(i)}, -y_1^{(i)}, y_2^{(i)}). \tag{8.7}$$

Поэтому после отображения каждого прямоугольника $R^{(i)}\in\mathscr{R}$ в соответствующую ему четырехмерную точку задача об охвате прямоугольника становится задачей о доминировании точек в четырехмерном пространстве [1]). А именно:

**Задача О.11** (ДОМИНИРОВАНИЕ). Дано множество точек $S=\{p_1, \ldots, p_N\}$ в $d$-мерном пространстве. Надо найти для каждой точки $p_i\in S$ такое подмножество $D_i\subseteq S$, что $D_i=\{p: p\in S, p\prec p_i\}$.

Метод решения задачи О.11, который будет описан, сильно напоминает (что не удивительно) метод, который был предложен ранее для решения задачи поиска максимумов на множестве векторов (в разд. 4.1.3). По-прежнему при $d=4$ обозначим через $u_1$, $u_2$, $u_3$ и $u_4$ координаты в $E^4$. Первый предварительный шаг состоит в преобразовании каждого $R^{(i)}\in\mathscr{R}$ в точку $p(R^{(i)})$ из $E^4$, где функция $p(\ )$ описывается приведенными выше соот-

[1]) Это соответствие было отмечено также в работе [Edelsbrunner, Overmars (1982)].

ношениями (8.6) и (8.7). Итак, получено множество $S = \{p(R^{(i)}): R^{(i)} \in \mathscr{R}\} = \{p_1, \ldots, p_N\}$, индексы элементов которого следует изменить так, чтобы $(i < j) \Rightarrow (u_1(p_i) \leqslant u_1(p_j))$. Теперь имеем

**procedure** ДОМИНИРОВАНИЕ

Д1. (Разделение) Разбить $S$ на $(S_1, S_2)$, где $S_1 = \{p_1, \ldots, p_{\lfloor N/2 \rfloor}\}$, а $S_2 = \{p_{\lfloor N/2 \rfloor + 1}, \ldots, p_N\}$.

Д2. (Рекурсия) Решить задачу о доминировании точек на множествах $S_1$ и $S_2$ независимо.

Д3. (Слияние) Найти все такие пары $p_i \prec p_j$, у которых $p_i \in S_1$, а $p_j \in S_2$.

Теперь обсудим реализацию шага Д3. Заметим, что на этом шаге решается задача О.9 о слиянии доминирований. Для пары $p_i \in S_1$ и $p_j \in S_2$, поскольку по построению $u_1(p_i) \leqslant u_1(p_j)$, то $p_i \prec p_j$ тогда и только тогда, когда $u_l(p_i) \leqslant u_l(p_j)$ для $l = 2$, 3, 4. Поэтому шаг Д3 фактически является *трехмерной задачей*. Опять воспользуемся методом «разделяй и властвуй» и обозначим через $\bar{u}_2$ медиану $\{u_2(p_i): p_i \in S_2\}$.

**procedure** СЛИЯНИЕ-ДОМИНИРОВАНИЙ

С1. (Разделение) Разбить $S_1$ на $(S_{11}, S_{12})$, а $S_2$ на $(S_{21}, S_{22})$ так, чтобы $S_{11} = \{p: p \in S_1, u_2(p) \leqslant \bar{u}_2\}$, $S_{21} = \{p: p \in S_2, u_2(p) \leqslant u\}$, а $S_{12} = S_1 - S_{11}$, $S_{22} = S_2 - S_{21}$.

С2. (Рекурсия) Решить задачу слияния на парах множеств $(S_{11}, S_{21})$ и $(S_{12}, S_{22})$.

С3. (Объединение) Найти все такие пары $p_i \prec p_j$, у которых $p_i \in S_{11}$, а $p_j = S_{22}$.

Чтобы установить корректность описанного выше метода, заметим, во-первых, что $S$ разбито на множества $S_{11}$, $S_{12}$, $S_{21}$ и $S_{22}$. Обращаясь к рис. 8.34 (где каждая подзадача обозначена дугой), видим, что на каждом из указанных четырех подмножеств решается задача о доминировании точек на шаге Д2; остается рассмотреть вопрос о дугах, соединяющих пары этих подмножеств. Две из шести пар — $(S_{11}, S_{12})$ и $(S_{21}, S_{22})$ — также обрабатываются на шаге Д2; пары $(S_{11}, S_{21})$ и $(S_{12}, S_{22})$ — на шаге С2; пара $(S_{11}, S_{22})$ — на шаге С3, а пару $(S_{12}, S_{21})$ не надо рассматривать, поскольку для каждых $p_i \in S_{12}$ и $p_j \in S_{21}$ верно, что $u_1(p_i) \leqslant u_1(p_j)$ и $u_2(p_i) > u_2(p_j)$. Отметим также, что шаг С3 (Объединение) является двумерной задачей (на координатах $u_3$ и $u_4$).

Очевидно, что ключевой операцией всей задачи является реализация шага С3 — *двумерное* слияние (или объединение). Действительно, все вычисление сводится к аккуратному выполнению последовательности шагов, подобных шагу С3; поэтому



в последующем надо сосредоточить внимание на разработке эффективной реализации шага «Объединение». Будет показано, что «Объединение» можно выполнить за время, линейно зависящее от размера входных данных, с затратой $O(N \log N)$ времени на предварительную сортировку, которую можно отнести на счет общей процедуры. Позднее мы увидим, как этот результат влияет на оценку всей задачи.

Множества $S_{11}$ и $S_{12}$, возникающие на шаге С3, являются наборами двумерных точек на плоскости $(u_3, u_4)$. Создадим для

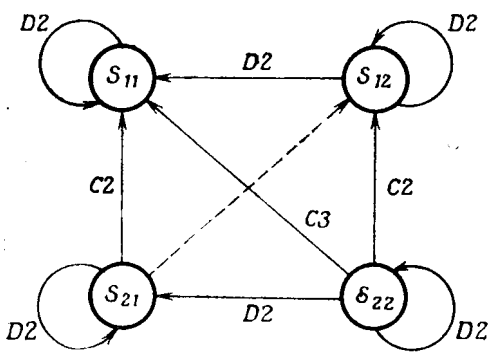



Рис. 8.34. Представление алгоритма для решения задачи СЛИЯНИЯ-ДОМИНИРОВАНИЙ методом «разделяй и властвуй» в качестве ориентированного графа. Каждое подмножество является узлом, а каждая подзадача — дугой.

каждого из этих множеств дважды прошитый двусвязный список следующим образом: каждой точке $p$ сопоставим узел, содержащий информацию $(u_3(p), u_4(p))$; кроме того, имеется два указателя СЛЕД3 и СЛЕД4, описывающие упорядочение по осям $u_3$ и $u_4$ соответственно. Связи в обратном направлении устанавливаются двумя дополнительными указателями ПРЕД3 и ПРЕД4. Временно пренебрежем затратами на построение дважды прошитых списков.

Обозначим через НАЧ31 и НАЧ32 указатели на начальные (первые) элементы по координате $u_3$ в списках $S_{11}$ и $S_{22}$ соответственно. Предлагаемый алгоритм имеет общую форму плоского заметания, точками событий которого являются координаты $u_3$ точек множества $S_{11} \cup S_{22}$, а статус заметающей прямой представлен списком $\mathscr{L}$. Этот список содержит упорядоченную по возрастанию координаты $u_4$ последовательность точек из $S_{11}$ (а именно $u_4$-координаты тех точек из $S_{11}$, чьи координаты по оси $u_3$ не превосходят текущего значения в процессе заметания). Временно будем использовать обозначения СЛЕД$\mathscr{L}$ и НАЧ$\mathscr{L}$ для прямого и начального указателей в списке $\mathscr{L}$, хотя,

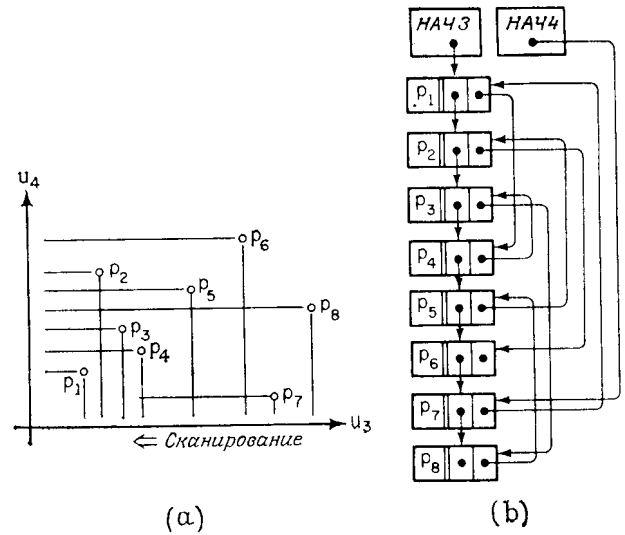
как будет показано ниже, СЛЕД$\mathscr{L}$ можно заменить на СЛЕД4. Предлагается следующий алгоритм:

```
        procedure ОБЪЕДИНЕНИЕ
 1  begin j1 := НАЧ31; j2 := НАЧ32;
 2        while (j2 ≠ Λ) do
 3              begin if (u3[j1] ⩽ u3[j2]) then
 4                    begin вставить u4[j1] в ℒ
                           с сохранением упорядоченности;
 5                        j1 := СЛЕД3[j1];
                      end
 6                  else begin l := НАЧℒ;
 7                        while (l ≠ Λ) and (u4[j2] ⩾ u4[l]) do
 8                              begin печать (j2, l);
 9                                    l := СЛЕДℒ[l];
                                end;
10                          j2 := СЛЕД3[j2];
                        end
              end
    end.
```

Такой алгоритм, очевидно, по структуре относится к типу алгоритмов слияния. В строке 3 производится проверка возможности продвижения по $S_{11}$ или по $S_{22}$. В первом случае надо вставить $u_4[j_1]$ в $\mathscr{L}$ (строка 4). Во втором случае (строки 6—9) надо просмотреть список $\mathscr{L}$, начиная с самого малого из его элементов, и определить при этом все такие точки, над которыми доминирует $p_{j2}$; эта часть алгоритма работает последовательно и затребует времени, пропорционального числу выданных на печать пар $(j_2, l)$.

Самая важная часть процедуры находится в строке 4: «Вставить $u_4[j]$ в $\mathscr{L}$ с сохранением упорядоченности». Действительно, на первый взгляд кажется, что на это понадобятся затраты времени, пропорциональные $|S_{11}|^2$, поскольку каждая вставка может потребовать полного обхода $\mathscr{L}$; однако более совершенная структура данных для $\mathscr{L}$ — перестраиваемое дерево (т. е. дерево, сбалансированное по высоте или по весу) — сократила бы затраты общего времени до $O(|S_{11}|\log|S_{11}|)$. Вместе с тем существует один интересный способ организации решения данной задачи, такой, что его суммарные затраты времени равны $O(|S_{11}|)$. Наша цель состоит в том, чтобы создать расписание вставок в список $\mathscr{L}$ элементов из $u_4$-списка для $S_{11}$. Обращаясь к рис. 8.35, реализуем эту цель следующим образом. Ясно, что координату по оси $u_4$ крайнего справа элемента $S_{11}$, т. е. $p_8$, надо вставить в список $\mathscr{L}$ (т. е. в упорядоченную по координате $u_4$ последовательность точек, лежащих слева от $p_8$) между $u_4[3]$ и $u_4[5]$. Но $u_4[3]$ — это не что иное, как ПРЕД4[8]; следовательно, ПРЕД4[8] и указывает ту позицию, в которой надо вставить $u_4[8]$ в $\mathscr{L}$. Теперь, удалив $p_8$ из $u_4$-списка и повторяя процедуру для остатка этого списка, получим позицию для вставки нового элемента $u_4[7]$. Сканируя подобным образом






(a) (b)

Рис. 8.35. Пример множества $S_{11} = \{p_1, \ldots, p_8\}$ и соответствующего ему дважды прошитого списка. Приведены связи для указателей СЛЕД3 и СЛЕД4.

$u_3$-список в порядке убывания, получаем расписание вставок всех элементов из $u_4$-списка; формальной записью этой идеи является следующий алгоритм:

```
begin l := ПОСЛЕДНИЙ ЭЛЕМЕНТ (u3-списка);
      while (ПРЕД3[l] ≠ НАЧ3) do
            begin СЛЕД4[ПРЕД4[l]] := СЛЕД4[l];
                  ПРЕД4[СЛЕД4[l]] := ПРЕД4[l];
                  l := ПРЕД3[l]
            end
end.
```

**Пример.** Для множества $S_{11}$, изображенного на рис. 8.35(a), на рис. 8.35(b) показана начальная конфигурация $u_3$- и $u_4$-списков. Начальная конфигурация массива ПРЕД4 такова:

| j: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| ПРЕД4: | 7 | 5 | 4 | 1 | 8 | 2 | НАЧ | 3 |

Эволюция этого массива в процессе сканирования компактно изображена в следующей таблице (корректируемые элементы обведены кружками):

*Таблица 1*

| j: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | После сканирования |
|---|---|---|---|---|---|---|---|---|---|
| Начальный ПРЕД4 | 7 | 5 | 4 | 1 | 8 | 2 | НАЧ | 3 | — |
| | 7 | 5 | 4 | 1 | ③ | 2 | НАЧ | 3 | $p_8$ |
| | (НАЧ) | 5 | 4 | 1 | 3 | 2 | НАЧ | 3 | $p_7$ |
| | НАЧ | 5 | 4 | 1 | 3 | 2 | НАЧ | 3 | $p_6$ |
| | НАЧ | ③ | 4 | 1 | 3 | 2 | НАЧ | 3 | $p_5$ |
| | НАЧ | 3 | ① | 1 | 3 | 2 | НАЧ | 3 | $p_4$ |
| | НАЧ | ① | 1 | 1 | 3 | 2 | НАЧ | 3 | $p_3$ |
| Финал (расписание вставок) | НАЧ | 1 | 1 | 1 | 3 | 2 | НАЧ | 3 | $p_2$ |

Следовательно, окончательная конфигурация массива ПРЕД4 полностью определяет расписание вставок в список $\mathscr{L}$ (который становится $u_4$-списком после окончания сканирования), и строку 4 процедуры ОБЪЕДИНЕНИЕ можно выполнить за константное время. Это доказывает тот факт, что вся процедура ОБЪЕДИНЕНИЕ затрачивает $|S_{11}|+|S_{22}|$ времени плюс еще столько времени, каково число найденных пар доминирований точек. Имея теперь в своем распоряжении процедуру ОБЪЕДИНЕНИЕ, мы можем заняться организацией всей процедуры ДОМИНИРОВАНИЯ.

Сначала займемся структурой данных. После сортировки множества $S$ по каждой из четырех координат получаем четырежды прошитый список (ЧПС). Как и в упомянутом ранее дважды прошитом списке, здесь все связи будут двусторонними, а указатели СЛЕД$j$ и ПРЕД$j$ будут использоваться для $u_j$-списка. Очевидно, что построение ЧПС для $S$ потребует $O(N\log N)$ времени. ЧПС весьма естественно подходит для реализации за линейное время операций по разбиению множеств, определяемых на шагах Д1 и С1 в ранее изложенных алгоритмах. Действительно, предположим, что надо разбить $S$ на пару $(S_1, S_2)$ и пометить элементы $S_1$. Тогда, если пройти по ЧПС с помощью любого выбранного указателя СЛЕД$i$ при $i=1, 2, 3, 4$, то список, соответствующий этому указателю, можно разбить на два списка, которые соответствуют двум подмножествам разбиения: $S_1$ и $S_2$. Аналогично, если заданы $S_1$ и $S_2$, то можно слить два соответствующих им списка, используя «естественное слияние» [Knuth (1973)], за линейное время.



Заметим, что операции разбиения и слияния заключаются просто в модификации указателей и использовании дополнительной памяти для промежуточных данных. Чтобы проанализировать работу предложенного метода, заметим:

(1) Вся работа ведется на одной и той же памяти с использованием массивов ЧПС и сводится лишь к преобразованиям указателей. Поэтому используемая память равна $\theta(N)$.

(2) Что касается времени работы, то каждая пара доминирования (т. е. каждая пара прямоугольников, один из которых охватывает другой) обнаруживается только один раз и за константное время в цикле типа *while* (строки 7—9) процедуры ОБЪЕДИНЕНИЕ. Поэтому, если $S$ — число таких пар, то на данную работу уйдет оптимальное время $\theta(S)$. Остальное затраченное время зависит только от $N$ — мощности множества $S$; обозначим его через $D(N)$. Обозначим через $M_d(r, s)$ время работы алгоритма СЛИЯНИЯ ДОМИНИРОВАНИЙ на паре множеств, состоящих из $r$ и $s$ $d$-мерных точек, где $d=2, 3$. Предполагая для простоты, что $N$ четно, имеем

$$D(N)=2D(N/2)+M_3(N/2,\ N/2)+O(N), \tag{8.8}$$

где $O(N)$ времени займет шаг Д1 — «разделение». Аналогично имеем (полагая, что $|S_{21}|=m$, а $r$ — четно)

$$M_3(r,\ s)=M_3(r/2,\ m)+M_3(r/2,\ s-m)+$$
$$+M_2(r/2,\ \max(m,\ s-m))+O(r+s), \tag{8.9}$$

где $O(r+s)$ — это опять время, нужное для реализации разбиения множества. Верхняя оценка для $M_3(r,s)$ получается в результате максимизации правой части (8.9) относительно параметра $m$. Поскольку верхней оценкой для $M_2(r', s')$ является $O(r'+s')$, то повторяя рассуждения из работы [Kung, Luccio, Preparata (1975)], получаем $M_3(r, s)=O((r+s)\log(r+s))$, и, следовательно, $D(N)=O(N\log^2 N)$.

Тем самым доказана следующая теорема:

**Теорема 8.10.** *Задача об охватах, поставленная на $N$ прямоугольниках (и эквивалентная ей задача* о ДОМИНИРОВАНИИ *в четырехмерном пространстве), может быть решена за время $O(N\log^2 N+s)$ с оптимальной затратой памяти $\theta(N)$, где $s$ — это число пар объектов, связанных отношением доминирования.*

Безусловно, остается открытым вопрос о существовании лучшего алгоритма для решения двух вышеупомянутых эквивалентных задач. Вместе с тем установление для них в качестве нижней оценки по времени $\Omega(N\log^2 N)$ является весьма маловероятной перспективой.

## 8.9. Замечания и комментарии

Некоторые из задач, обсуждавшихся в данной главе, допускают интересные обобщения. Например, задачу о перечислении пересечений $N$ прямоугольников на плоскости можно обобщить на произвольное число измерений. Обозначим через $K$ число пересекающихся пар гиперпрямоугольников, а через $d$ число измерений. Сикс и Вуд [Six, Wood (1982)] построили алгоритм с оценками $O(N\log^{d-1}N+K)$ — по времени и $O(N\log^{d-1}N)$ — по памяти; оценка этого алгоритма по времени позднее была улучшена Эдельсбруннером [Edelsbrunner (1983)] до $O(N\log^{d-2}N)$. Чазелле и Инчерпи [Chazelle, Incerpi (1983)], а также Эдельсбруннер и Овермарс [Edelsbrunner, Overmars (1985)] снизили оценку по памяти до $\theta(N)$, которая, очевидно, является оптимальной, сохранив прежнюю оценку по времени. Соответствующую задачу подсчета в отличие от случая пересечения отрезков (см. разд. 7.2.3) можно решить за время $O(N\log^{d-1}N)$ с затратой $O(N\log^{d-2}N)$ памяти, слегка модифицировав алгоритм перечисления пересечений, принадлежащий Сиксу и Вуду, в то время как алгоритм Чазелле и Инчерпи нельзя адаптировать для эффективного решения данной задачи.

Интересным обобщением задачи о пересечении прямоугольников является задача, обычно называемая ПОИСК ПЕРЕСЕЧЕНИЙ С ОРТОГОНАЛЬНЫМИ ОБЪЕКТАМИ, в которой среди заданных $N$ изотетичных объектов надо найти все объекты, которые пересекают заданный, изотетичный им запросный объект. (Разумеется, что канонической формой изотетичного объекта в $E^d$ является декартово произведение $d$ интервалов, каждый из которых может вырождаться в единственное значение. Например, точка, изотетичный прямоугольник, а также горизонтальный и вертикальный отрезки прямых являются ортогональными объектами.) Примером подобной задачи, обсуждавшимся в гл. 2, является многомерный региональный поиск в обеих своих формах: «отчета» и «подсчета». ОБРАТНАЯ ЗАДАЧА РЕГИОНАЛЬНОГО ПОИСКА или ОХВАТА ТОЧКИ также относится к этому классу; она ставится так: среди заданных $N$ изотетичных прямоугольников надо найти такие, которые охватывают некую запросную точку. Вайшнави [Vaishnavi (1982)] предложил структуру данных, которая обеспечивает поиск с затратами $O(\log N+K)$[1]) времени на один запрос и $O(N\log N)$ памяти. Чазелле [Chazelle (1983c)] создал оптимальный алгоритм для этой задачи, т. е. его оценки равны $\theta(N)$ — по памяти и $\theta(\log N+K)$ — по времени запроса. Еще одним примером такой задачи является задача о ПОИСКЕ ПЕРЕСЕЧЕНИЙ С ОРТОГОНАЛЬНЫМИ ОТРЕЗКАМИ, в которой требуется найти среди $N$ заданных горизонтальных и вертикальных прямолинейных отрезков все пересекающие заданный ортогональный запросный отрезок. Эту задачу исследовали Вайшнави и Вуд [Vaishnavi, Wood (1982)], предложившие алгоритм, который обладает следующими оценками: по времени запроса — $O(\log N+K)$, а по времени предварительной обработки и по памяти — $O(N\log N)$. Оценку по памяти можно снизить до $\theta(N)$ [Chazelle (1983c)]. Для случая, когда в данной задаче разрешены вставки и удаления отрезков, Маккрейт предложил динамическую структуру данных [McCreight (1981)], которая требует $O(N)$ памяти и $O(\log^2 N+K)$ времени, где $O(N)$ выражает мощность множества различных значений координат. Для этого же случая Липски и Пападимитриу [Lipski, Papadimitriou (1981)] предложили алгоритм с оценками по времени — $O(\log N\cdot\log\log N+K)$, а по памяти — $O(N\log N)$. Недавно Эдельсбруннер и Маурер [Edelsbrunner, Maurer (1981)] унифицировали некоторые из ранее упомянутых методов решения данного класса задач о поиске пересечений и получили следующие результаты. Для задач статического типа существует структура данных, обеспечивающая следующие оценки эффективности: для времени запроса — $O(\log^{d-1}N+K)$, для времени предварительной обработки и памяти — по $O(N\log^d N)$. Оценка по памяти позднее была доведена до $O(N\log^{d-1}N\cdot\log\log N)$ [Ghazelle, Guibas (1984)]. Что же касается динамического случая, то существует динамическая структура данных, обеспечивающая следующие оценки эффективности: для времени запроса — $O(\log^d N+K)$, для памяти — $O(N\log^d N)$, для времени корректировки — $O(\log^d N)$ [Edelsbrunner (1980)].

[1]) Здесь, как обычно, $K$ обозначает мощность результата. — *Прим. перев.*



## 8.10. Упражнения

1. Дано множество нетривиальных цепей для объединения, состоящего из $N$ изотетичных прямоугольников. Построить алгоритм, который выбирает среди них цепи, формирующие нетривиальный контур. (Цель состоит в получении алгоритма с оценкой по времени $O(N\log N)$.)

2. *ТРЕХМЕРНОЕ ДОМИНИРОВАНИЕ.* Дано множество из $N$ точек в $E^3$. Создать алгоритм, находящий $s$ пар доминирования за время $O(N\log N+s)$ с затратой $\theta(N)$ памяти.

3. *ТРЕХМЕРНОЕ ДОМИНИРОВАНИЕ.* Доказать, что нижней оценкой времени, необходимого для поиска пар доминирования среди $N$ точек в $E^3$, является $\Omega(N\log N+s)$.

4. *Охваты квадратов.* Задачу об ОХВАТАХ КВАДРАТОВ можно получить заменой слова «прямоугольник» словом «квадрат» в формулировке за-

дачи О.10. Показать, что задача об ОХВАТАХ КВАДРАТОВ эквивалентна задаче о ТРЕХМЕРНОМ ДОМИНИРОВАНИИ.

5. Показать, что задача о СЛИЯНИИ ДОМИНИРОВАНИЙ в $E^4$, определенная на множестве, состоящем из $N$ точек, может быть решена за время $O(N \log^2 N + s)$, где $s$ — число обнаруженных пар.

6. *Маршрут в лабиринте.* Лабиринт образован двумя наборами отрезков (в сумме их $N$ штук), каждый из которых содержит параллельные отрезки (для простоты будем считать, что направления отрезков из разных наборов ортогональны). Лабиринт разбивает плоскость на связные области. Для двух заданных на плоскости точек $s$ и $t$ маршрутом называется соединяющая их кривая, не пересекающая ни одного заданного отрезка.

(a) Построить алгоритм, проверяющий факт существования маршрута, соединяющего $s$ и $t$.

(b) Построить алгоритм, соединяющий $s$ и $t$ маршрутом (если таковой существует).

# Литература[1]


[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[29] S. B. Akers, Routing, in *Design Automation of Digital Systems*, M. Breuer, ed., Prentice-Hall, Englewood Cliffs, NJ, 1972.

S. G. Akl, Two remarks on a convex hull algorithm, *Info. Proc. Lett.* **8**, 108–109 (1979).

S. G. Akl and G. T. Toussaint, Efficient convex hull algorithms for pattern recognition applications, *Proc. 4th Int'l Joint Conf. on Pattern Recognition*, Kyoto, Japan, pp. 483–487 (1978).

A. M. Andrew, Another efficient algorithm for convex hulls in two dimensions, *Info. Proc. Lett.* **9**, 216–219 (1979).

H. C. Andrews, *Introduction to Mathematical Techniques in Pattern Recognition*, Wiley-Interscience, New York, 1972.

M. J. Atallah, A linear time algorithm for the Hausdorff distance between convex polygons, *Info. Proc. Lett.* **8**, 207–209 (Nov. 1983).

D. Avis, On the complexity of finding the convex hull of a set of points, McGill University, School of Computer Science; Report SOCS 79.2, 1979

D. Avis and B. K. Bhattacharya, Algorithms for computing *d*-dimensional Voronoi diagrams and their duals, in *Advances in Computing Research*. Edited by F. P. Preparata. **1**, JAI Press, 159–180 (1983).

H. S. Baird, Fast algorithms for LSI artwork analysis, *Design Automation and Fault-Tolerant Computing*, **2**, 179–209 (1978).

R. E. Barlow, D. J. Bartholomew, J. M. Bremner and H. D. Brunk, *Statistical Inference under Order Restrictions*, Wiley, New York, 1972.

M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th ACM Annual Symp. on Theory of Comput.*, pp. 80–86 (May 1983).

R. V. Benson, *Euclidean Geometry and Convexity*, McGraw-Hill, New York, 1966.

J. L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM* **18**, 509–517 (1975).

J. L. Bentley, Algorithms for Klee's rectangle problems, Carnegie-Mellon University, Pittsburgh, Penn., Department of Computer Science, unpublished notes, 1977.

J. L. Bentley, Decomposable searching problems, *Info. Proc. Lett.* **8**, 244–251 (1979).


[1]) Цифровыми индексами отмечены работы, вышедшие на русском языке или в русском переводе, которые входят под соответствующим номером в отдельный список, приведенный далее. Звездочкой отмечены работы, уточненные выходные данные которых приведены в списке дополнительной литературы к русскому изданию.

J. L. Bentley, Multidimensional divide and conquer, *Comm. ACM* **23**(4), 214–229 (1980).
J. L. Bentley, G. M. Faust and F. P. Preparata, Approximation algorithms for convex hulls, *Comm. ACM* **25**, 64–68 (1982).
J. L. Bentley, H. T. Kung, M. Schkolnick and C. D. Thompson, On the average number of maxima in a set of vectors, *J. ACM* **25**, 536–543 (1978).
J. L. Bentley and H. A. Maurer, A note on Euclidean near neighbor searching in the plane, *Inform. Processing Lett.* **8**, 133–136 (1979).
J. L. Bentley and H. A. Maurer, Efficient worst-case data structures for range searching, *Acta Informatica* **13**, 155–168 (1980).
J. L. Bentley and T. A. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers* **28**, 643–647 (1979).
J. L. Bentley and M. I. Shamos, Divide-and-Conquer in Multidimensional Space, *Proc. Eighth ACM Annual Symp. on Theory of Comput.*, pp. 220–230 (May 1976).
J. L. Bentley and M. I. Shamos, A problem in multivariate statistics: Algorithms, data structure, and applications, *Proceedings of the 15th Annual Allerton Conference on Communication, Control, and Computing*, pp. 193–201 (1977).
J. L. Bentley and M. I. Shamos, Divide and conquer for linear expected time, *Info. Proc. Lett.* **7**, 87–91 (1978).
J. L. Bentley and D. F. Stanat, Analysis of range searches in quad trees, *Info. Proc. Lett.* **3**, 170–173 (1975).
J. L. Bentley and D. Wood, An optimal worst case algorithm for reporting intersection of rectangles, *IEEE Transactions on Computers* **29**, 571–577 (1980).
P. Bézier, *Numerical Control—Mathematics and Applications*. Translated by A. R. Forrest. Wiley, New York, 1972.
B. Bhattacharya, Worst-case analysis of a convex hull algorithm, unpublished manuscript, Dept. of Computer Science, Simon Fraser University, February 1982.
G. Bilardi and F. P. Preparata, Probabilistic analysis of a new geometric searching technique, unpublished manuscript, 1981.
G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, McMillan, New York, 1965.
M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest and R. E. Tarjan, Time bounds for selection, *Jour. Compt. Sys. Sci.* **7**, 448–461 (1973).
B. Bollobás, *Graph Theory, An Introductory Course*, Springer-Verlag, New York, 1979.
K. Q. Brown, Fast intersection of half spaces, Carnegie-Mellon University, Pittsburgh, Pennsylvania, Department of Computer Science; Report CMU-CS-78-129, 1978.
K. Q. Brown, Geometric transformations for fast geometric algorithms, Ph. D. thesis, Dept. of Computer Science, Carnegie Mellon Univ., Dec. 1979a.
K. Q. Brown, Voronoi diagrams from convex hulls, *Info. Proc. Lett.* **9**, 223–228 (1979b).
W. A. Burkhard and R. M. Keller, Some approaches to best match file searching, *Comm. ACM* **16**, 230–236 (1973).
A. Bykat, Convex hull of a finite set of points in two dimensions, *Info. Proc. Lett.* **7**, 296–298 (1978).
J. C. Cavendish, Automatic triangulation of arbitrary planar domains for the finite element method, *Int'l J. Numerical Methods in Engineering* **8**, 679–696 (1974).
D. R. Chand and S. S. Kapur, An algorithm for convex polytopes, *JACM* **17**(1), 78–86 (Jan. 1970).
B. M. Chazelle, Reporting and counting arbitrary planar intersections, Rep. CS-83-16, Dept. of Comp. Sci., Brown University, Providence, RI, 1983a.
B. M. Chazelle, Optimal algorithms for computing depths and layers, *Proc. 21st Allerton Conference on Comm., Control and Comput.*, pp. 427–436 (Oct. 1983b).
B. M. Chazelle, Filtering search: A new approach to query-answering, *Proc. 24th IEEE Symp. on Foundations of Comp. Sci.*, Tucson, AZ, 122–132 Nov. 1983c.
*B. M. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap, New upper bounds for neighbor searching, submitted for publication (1984).
B. M. Chazelle and D. P. Dobkin, Detection is easier than computation, *Proc. 12th ACM Annual Symp. on Theory of Comput.*, pp. 146–153, (May 1980).
*B. M. Chazelle and H. Edelsbrunner, Optimal solutions for a class of point retrieval problems, Tech. Rep. CS-84-16, Dept. of Computer Science, Brown University, July 1984.
B. M. Chazelle and L. J. Guibas, Fractional cascading: A data structuring technique with geometric applications, manuscript, 1984.
B. M. Chazelle, L. J. Guibas and D. T. Lee, The power of geometric duality, *Proc. 24th IEEE Annual Symp. on Foundations of Comput. Sci.*, pp. 217–225 (Nov. 1983).
B. M. Chazelle and J. Incerpi, Triangulating a polygon by divide-and-conquer, *Proc. 21st Allerton Conference on Comm. Control and Comput.*, pp. 447–456 (Oct. 1983).
D. Cheriton and R. E. Tarjan, Finding minimum spanning trees, *SIAM J. Comput.* **5**(4), 724–742 (Dec. 1976).
F. Chin and C. A. Wang, Optimal algorithms for the intersection and the minimum distance problems between planar polygons, *IEEE Trans. Comput.* **C-32**(12), 1203–1207 (1983).
F. Chin and C. A. Wang, Minimum vertex distance between separable convex polygons, *Info. Proc. Lett.* **18**, 41–45 (1984).
N. Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Symposium on Algorithms and Complexity. Department of Computer Science, Carnegie-Mellon University, Apr. 1976.
V. Chvátal, A combinatorial theorem in plane geometry, *J. Comb. Theory B.* **18**, 39–41 (1975).
J. L. Coolidge, *A Treatise on the Circle and the Sphere*, Oxford University Press, Oxford, England, 1916. Reprinted 1971 by Chelsea.
B. Dasarathy and L. J. White, On some maximin location and classifier problems, Computer Science Conference, Washington, D.C., 1975 (Unpublished lecture).
P. J. Davis, *Interpolation and Approximation*, Blaisdell, NY (1963). Reprinted 1975 by Dover, New York.
[5]B. Delaunay, Sur la sphère vide, *Bull. Acad. Sci. USSR(VII)*, Classe Sci. Mat. Nat., 793–800 (1934).
R. B. Desens, Computer processing for display of three-dimensional structures, Technical Report CFSTI AD-7006010, Naval Postgraduate School, Oct. 1969.
E. W. Dijkstra, A note on two problems in connection with graphs, *Numer. Math.* **1**(*5*), 269–271 (Oct. 1959).
D. P. Dobkin and D. G. Kirkpatrick, Fast algorithms for preprocessed polyhedral intersection detection, submitted for publication (1984).
David Dobkin and Richard Lipton, Multidimensional searching problems, *SIAM J. Comput.* **5**(*2*), 181–186 (June 1976).
David Dobkin and Richard Lipton, On the complexity of computations under varying set of primitives, *Journal of Computer and Systems Sciences* **18**, 86–91 (1979).
R. L. Drysdale, III, Generalized Voronoi diagrams and geometric searching, Ph.D. Thesis, Dept. Comp. Sci., Stanford University, Tech. Rep. STAN-CS-79-705 (1979).
[6]R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience, New York, 1973.
R. D. Düppe and H. J. Gottschalk, Automatische Interpolation von Isolinien bei willkürlich verteilten Stützpunkten, *Allgemeine Vermessungsnachrichten* **77**, 423–426 (1970).
M. E. Dyer, A simplified O(nlogn) algorithm for the intersection of 3-polyhedra,

ematics and Statistics; Report TPMR 80-5, 1980.
M. E. Dyer, Linear time algorithms for two- and three-variable linear programs, *SIAM J. Comp.* **13**(1), 31–45 (Feb. 1984).
*M. I. Edahiro, I Kokubo and T. Asano, A new point-location algorithm and its practical efficiency—comparison with existing algorithms, Res. Memo. RMI 83-04, Dept. Math. Eng. B Instrumentation Physics, Univ. of Tokyo, Oct. 1983.
W. Eddy, A new convex hull algorithm for planar sets, *ACM Trans. Math. Software* **3**(4), 398–403 (1977).
H. Edelsbrunner, Dynamic data structures for orthogonal intersection queries, Rep. F59, Tech. Univ. Graz, Institute für Informationsverarbeitung 1980.
H. Edelsbrunner, Intersection problems in computational geometry, Ph. D. Thesis, Rep. 93, IIG, Technische Universität Graz, Austria, 1982.
H. Edelsbrunner, A new approach to rectangle intersections, Part II, *Int'l. J. Comput. Math.* **13**, 221–229 (1983).
*E. Edelsbrunner, L. J. Guibas and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM J. Comput.*, to appear (1985).
*H. Edelsbrunner, J. van Leeuwen, T. A. Ottman, and D. Wood, Connected components of orthogonal geometric objects, McMaster University, Hamilton, Ontario, Unit for Computer Science, Report 81-CS-04, 1981.
H. Edelsbrunner and H. A. Maurer, On the intersection of orthogonal objects, *Info. Proc. Lett.* **13**, 177–181 (1981).
H. Edelsbrunner and M. H. Overmars, On the equivalence of some rectangle problems. *Information Processing Letters* **14**(3), 124–127 (May 1982).
*H. Edelsbrunner and M. H. Overmars, Batched dynamic solutions to decomposable searching problems, *J. Algorithms*, to appear (1985).
H. Edelsbrunner, M. H. Overmars and D. Wood, Graphics in flatland: A case study, in *Advances in Computing Research*, Edited by F. P. Preparata, Vol. 1, JAI Press, 35–59 (1983).
*H. Edelsbrunner and R. Seidel, Arrangements of planes and Voronoi diagrams, in preparation (1985).
*H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and O(n** 0.695) query time, Rep. 111, IIG, Technische Universität Graz, Austria, 1983.
J. Edmonds, Maximum matching and a polyhedron with 0,1 vertices, *J. Res. NBS*, 69B, 125–130 (Apr.–June 1965).
B. Efron, The convex hull of a random set of points, *Biometrika* **52**, 331–343 (1965).
H. El Gindy and D. Avis, A linear algorithm for computing the visibility polygon from a point, *J. Algorithms* **2**(2), 186–197 (1981).
D. J. Elzinga and D. W. Hearn, Geometrical solutions for some minimax location problems, *Transportation Science* **6**, 379–394 (1972a).
D. J. Elzinga and D. W. Hearn, The minimum covering sphere problem, *Mgmt. Sci.* **19**(1), 96–104 (Sept. 1972b).
P. Erdös, On sets of distances of $n$ points, *Amer. Math. Monthly* **53**, 248–250 (1946).
P. Erdös, On sets of distances of $n$ points in Euclidean space, *Magy. Tud. Akad. Mat. Kut. Int. Kozl.* **5**, 165–169 (1960).
K. P. Eswaran, J. N. Gray, R. A. Lorie and I. L. Traiger, The notions of consistency, and predicate locks in a database system, *Comm. ACM* **19**, 624–633 (1976).
S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.
H. Eves, *A Survey of Geometry*, Allyn and Bacon, Newton, Mass., 1972.
G. Ewald, *Geometry: An Introduction*, Wadsworth, Belmont, Calif., 1971.
I. Fáry, On straight-line representation of planar graphs, *Acta Sci. Math. Szeged*, **11**, 229–233 (1948).
R. A. Finkel and J. L. Bentley, Quad-trees: a data structure for retrieval on composite keys, *Acta Inform.* **4**, 1–9 (1974).



187–195 (1971).
R. L. Francis and J. A. White, *Facility Layout and Location: An Analytical Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
M. L. Fredman, A lower bound of the complexity of orthogonal range queries, *J. ACM* **28**, 696–705 (1981).
M. L. Fredman and B. Weide, The complexity of computing the measure of $\cup[a_i, b_i]$, *Comm. ACM* **21**(7), 540–544 (July 1978).
H. Freeman, Computer processing of line-drawing images, *Comput. Surveys* **6**, 57–97 (1974).
H. Freeman and P. P. Loutrel, An algorithm for the solution of the two-dimensional hidden-line problem, *IEEE Trans. Elec. Comp.* **EC-16**(6), 784–790 (1967).
H. Freeman and R. Shapira, Determining the minimum-area encasing rectangle for an arbitrary closed curve, *Comm. ACM.* **18**(7), 409–413 (1975).
J. H. Friedman, J. L. Bentley and R. A. Finkel, An algorithm for finding best match in logarithmic expected time, *ACM Trans. Math. Software* **3**(3), 209–226 (1977).
H. Gabow, An efficient implementation of Edmond's maximum matching algorithm, Technical Report 31, Computer Science Department, Stanford Univ., 1972.
K. R. Gabriel and R. R. Sokal, A new statistical approach to geographic variation analysis, *Systematic Zoology* **18**, 259–278 (1969).
R. Galimberti and U. Montanari, An algorithm for hidden-line elimination, *CACM* **12**(4), 206–211 (1969).
M. R. Garey, R. L. Graham and D. S. Johnson, Some NP-complete geometric problems, *Eighth Annual Symp. on Theory of Comput.*, pp. 10–22 (May 1976).
[4]M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.
M. R. Garey, D. S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theor. Comp. Sci.* **1**, 237–267 (1976).
M. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, Triangulating a simple polygon, *Info. Proc. Lett.* **7**(4), 175–180 (1978).
S. I. Gass, *Linear Programming*, McGraw-Hill, New York, 1969.
J. Gastwirth, On robust procedures, *J. Amer. Stat. Assn.* **65**, 929–948 (1966).
J. A. George, Computer implementation of the finite element method, Technical Report STAN-CS-71-208, Computer Science Department, Stanford University, 1971.
P. N. Gilbert, New results on planar triangulations, Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979.
T. Gonzalez, Algorithms on sets and related problems, Technical Report, Department of Computer Science, University of Oklahoma, 1975.
J. C. Gower and G. J. S. Ross, Minimum spanning trees and single linkage cluster analysis, *Appl. Stat.* **18**(1), 54–64 (1969).
R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, *Info. Proc. Lett.* **1**, 132–133 (1972).
R. L. Graham and F. F. Yao, Finding the convex hull of a simple polygon, Tech. Rep. No. STAN-CS-81-887, Stanford University, 1981; also *J. Algorithms* **4**(4), 324–331 (1983).
P. J. Green and B. W. Silverman, Constructing the convex hull of a set of points in the plane, *Computer Journal* **22**, 262–266 (1979).
B. Grünbaum, A proof of Vazsonyi's conjecture, *Bull. Res. Council Israel* **6**(*A*), 77–78 (1956).
B. Grünbaum, *Convex Polytopes*, Wiley-Interscience, New York, 1967.
R. H. Güting, An optimal contour algorithm for isooriented rectangles, *Jour. of Algorithms* **5**(3), 303–326 (Sept. 1984).
[27]H. Hadwiger and H. Debrunner, *Combinatorial Geometry in the Plane*, Holt, Rinebart

[28] M. Hanan and J. M. Kurtzberg, Placement techniques, in *Design Automation of Digital Systems*. Edited by M. Breuer. Prentice-Hall, Englewood Cliffs, NJ, 1972.
M. Hanan, Layout, interconnection, and placement, *Networks* **5**, 85–88 (1975).
J. A. Hartigan, *Clustering Algorithms*, Wiley, New York, 1975.
T. L. Heath, *A History of Greek Mathematics*, Oxford University Press, Oxford, England, 1921.
*S. Hertel, K. Mehlhorn, M. Mäntylä and J. Nievergelt, Space sweep solves intersection of two convex polyhedron elegantly, *Acta Informatica* (to appear) (1985).
[3] D. Hilbert, *Foundations of Geometry*, (1899). Repr. 1971 by Open Court, La Salle, IL.
J. G. Hocking and G. S. Young, *Topology*, Addison-Wesley, Reading, MA, 1961.
C. A. R. Hoare, Quicksort, *Computer Journal* **5**, 10–15 (1962).
I. Hodder and C. Orton, *Spatial Analysis in Archaeology*, Cambridge University Press, Cambridge, England, 1976.
P. G. Hoel, *Introduction to Mathematical Statistics*, Wiley, New York, 1971.
P. J. Huber, Robust statistics: A review, *Ann. Math. Stat.* **43**(3), 1041–1067 (1972).
F. K. Hwang, An $O(n \log n)$ algorithm for rectilinear minimal spanning tree, *J. ACM* **26**, 177–182 (1979).
J. W. Jaromczyk, Lower bounds for polygon simplicity testing and other problems, *Proc. MFCS-84*, Prague (Springer-Verlag), 339–347 (1984).
R. A. Jarvis, On the identification of the convex hull of a finite set of points in the plane, *Info. Proc. Lett.* **2**, 18–21 (1973).
S. C. Johnson, Hierarchical clustering schemes, *Psychometricka* **32**, 241–254 (1967).
M. Kallay, Convex hull algorithms in higher dimensions, unpublished manuscript, Dept. Mathematics, Univ. of Oklahoma, Norman, Oklahoma, 1981.
[8] R. M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computations*. Edited by R. E. Miller and J. W. Thatcher. Plenum Press, New York, 1972.
J. M. Keil and J. R. Sack, Minimum decomposition of polygonal objects, in *Computational Geometry*. Edited by G. T. Toussaint. North-Holland, Amsterdam, The Netherlands (1985).
[9] M. G. Kendall and P. A. P. Moran, *Geometrical Probability*, Hafner, New York, 1963.
D. G. Kirkpatrick, Efficient computation of continuous skeletons, *Proc. 20th IEEE Annual Symp. on Foundations of Comput. Sci.*, pp. 18–27 (Oct. 1979).
D. G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Comput.* **12**(1), 28–35 (1983).
*D. G. Kirkpatrick and R. Seidel, The ultimate planar convex hull algorithm? Tech. Rep. 83–577, Dept. Comput. Sci., Cornell Univ., Oct. 1983.
V. Klee, Convex polytopes and linear programming, *Proc. IBM Sci. Comput. Symp. Combinatiorial Problems*, IBM, White Plains, NY, pp. 123–158 (1966).
V. Klee, Can the measure of $\cup[a_i, b_i]$ be computed in less than $O(n \log n)$ steps?, *Amer. Math. Monthly*, **84**(4), 284–285 (April 1977).
V. Klee, On the complexity of $d$-dimensional Voronoi diagrams, *Archiv der Mathematik*, **34**, 75–80 (1980).
[10] F. Klein, *Das Erlangen Programm: vergleichende Betrachtungen über neuere geometrisches Forschungen*, 1872. Reprinted by Akademische VerlagsGesellschaft Greest und Portig, Leipzig, 1974.
[11] D. E. Knuth, *The Art of Computer Programming. Volume I: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
[12] D. E. Knuth, *The Art of Computer Programming. Volume III: Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.
D. E. Knuth, Big omicron and big omega and big theta, *SIGACT News* **8**(2), 18–24 (April–June 1976).
J. F. Kolars and J. D. Nystuen, *Human Geography*, McGraw-Hill, New York, 1974.



problem, *Proc. AMS* **7**, 48–50 (1956).
H. T. Kung, F. Luccio and F. P. Preparata, On finding the maxima of a set of vectors, *JACM* **22**(4), 469–476 (Oct. 1975).
U. Lauther, 4-dimensional binary search trees as a means to speed up associative searches in the design verification of integrated circuits, *Jour. of Design Automation and Fault Tolerant Computing*, **2**(3), 241–247 (July 1978).
C. L. Lawson, The smallest covering cone or sphere, *SIAM Rev.* **7**(3), 415–417 (1965).
C. L. Lawson, Software for $C^1$ surface interpolation, JPL Publication, pp. 77–30 (Aug. 1977).
D. T. Lee, On finding k nearest neighbors in the plane, Technical Report. Department of Computer Science, University of Illinois, 1976.
D. T. Lee, Proximity and reachability in the plane, Tech. Rep. No. R-831, Coordinated Sci. Lab., Univ. of Illinois at Urbana, IL, 1978.
D. T. Lee, On k-nearest neighbor Voronoi diagrams in the plane, *IEEE Trans. Comput.* **C-31**(6), 478–487 (June 1982).
D. T. Lee, Two dimensional Voronoi diagram in the $L_p$-metric, *J. ACM* **27**, 604–618 (1980a).
D. T. Lee, Farthest neighbor Voronoi diagrams and applications, Tech. Rep. No. 80-11-FC-04, Dpt. EE/CS, Northwestern Univ., 1980b.
D. T. Lee, On finding the convex hull of a simple polygon, Tech. Rep. No. 80-03-FC-01, EE/CS Dept., Northwestern Univ., 1980. See also *Int'l J. Comput. and Infor. Sci.* **12**(2), 87–98 (1983a).
D. T. Lee, Visibility of a simple polygon, *Computer Vision, Graphics and Image Processing* **22**, 207–221 (1983b).
D. T. Lee and I. M. Chen, Display of visible edges of a set of convex polygons, in *Computational Geometry*. Edited by G. T. Toussaint. North Holland, Amsterdam, to appear (1985).
D. T. Lee and R. L. Drysdale III, Generalized Voronoi diagrams in the plane, *SIAM J. Comput.* **10**, 1, 73–87 (Feb. 1981).
D. T. Lee and F. P. Preparata, Location of a point in a planar subdivision and its applications, *SIAM Journal on Computing* **6**(3), 594–606 (Sept. 1977).
D. T. Lee and F. P. Preparata, The all nearest neighbor problem for convex polygons, *Info. Proc. Lett.* **7**, 189–192 (1978).
D. T. Lee and F. P. Preparata, An optimal algorithm for finding the kernel of a polygon, *Journal of the ACM* **26**, 415–421 (1979).
D. T. Lee and F. P. Preparata, An improved algorithm for the rectangle enclosure problem, *Journal of Algorithms*, **3**(3), 218–224 (1982).
D. T. Lee and B. Schachter, Two algorithms for constructing Delaunay triangulations, *Int'l J. Comput. and Info. Sci.* **9**(3), 219–242 (1980).
D. T. Lee and C. K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees, *Acta Informatica* **9**, 23–29 (1977).
D. T. Lee and C. K. Wong, Voronoi diagrams in $L_1$-($L_\infty$-) metrics with 2-dimensional storage applications, *SIAM J. Comput.* **9**(1), 200–211 (1980).
J. van Leeuwen and D. Wood, Dynamization of decomposable searching problems, *Info. Proc. Lett.* **10**, 51–56 (1980).
J. van Leeuwen and D. Wood, The measure problem for rectangular ranges in $d$-space, *J. Algorithms* **2**, 282–300 (1981).
E. Lemoine, *Géométrographie*, C. Naud, Paris, 1902.
W. Lipski, Jr. and C. H. Papadimitriou, A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems, *Journal of Algorithms* **2**, 211–226 (1981).
W. Lipski, Jr. and F. P. Preparata, Finding the contour of a union of iso-oriented rectangles, *Journal of Algorithms* **1**, 235–246 (1980).

W. Lipski, Jr. and F. P. Preparata, Segments, rectangles, contours, *Journal of Algorithms* **2**, 63–76 (1981).

R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs, *Conference on Theoretical Computer Science*, Waterloo, Ont., pp. 1–10 (August 1977a).

R. J. Lipton and R. E. Tarjan, Applications of a planar separator theorem, *Proc. 18th IEEE Annual Symp. on Found. of Comp. Sci.*, Providence, RI, pp. 162–170 (October 1977b).

R. J. Lipton and R. E. Tarjan, Applications of a planar separator theorem, *SIAM J. Comput.* **9**(3), 615–627 (1980).

C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, New York, 1968.

E. L. Lloyd, On triangulations of a set of points in the plane, *Proc. 18th IEEE Annual Symp. on Foundations of Comput. Sci.*, pp. 228–240 (Oct. 1977).

H. Loberman and A. Weinberger, Formal procedures for connecting terminals with a minimum total wire length, *JACM* **4**, 428–437 (1957).

D. O. Loftsgaarden and C. P. Queensberry, A nonparametric density function, *Ann. Math. Stat.* **36**, 1049–1051 (1965).

P. P. Loutrel, A solution to the hidden-line problem for computer-drawn polyhedra, *IEEE Trans. Comp.* **C-19**(3), 205–215 (March 1970).

G. S. Lueker, A data structure for orthogonal range queries, *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pp. 28–34 (1978).

* U. Manber and M. Tompa, Probabilistic, nondeterministic and alternating decision trees, Tech. Rep. No. 82-03-01, Univ. Washington, 1982.

* H. G. Mairson and J. Stolfi, Reporting line segment intersections in the plane, Tech. Rep., Dept. of Computer Sci., Stanford University, 1983.

Y. Matsushita, A solution to the hidden line problem, Technical Report 335, Department of Computer Science, University of Illinois, 1969.

D. W. Matula and R. R. Sokal, Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane, *Geographical Analysis* **12**, 205–222 (July 1980).

D. McCallum and D. Avis, A linear algorithm for finding the convex hull of a simple polygon, *Info. Proc. Lett.* **9**, 201–206 (1979).

E. M. McCreight, Priority search trees, Tech. Rep. Xerox PARC CSL-81-5, 1981.

P. McMullen and G. C. Shephard, *Convex Polytopes and the Upper Bound Conjecture*, Cambridge University Press, Cambridge, England, 1971.

C. A. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, Mass., 1979.

N. Megiddo, Linear time algorithm for linear programming in $R^3$ and related problems, *SIAM J. Comput.* **12**(4), 759–776 (Nov. 1983).

W. S. Meisel, *Computer-Oriented Approaches to Pattern Recognition*, Academic Press, New York, 1972.

Z. A. Melzak, *Companion to Concrete Mathematics*, Wiley-Interscience, New York, 1973.

J. Milnor, On the Betti numbers of real algebraic varieties, *Proc. AMS* **15**, 2745–280 (1964).

[14] J. Milnor, *Singular Points of Complex Hypersurfaces*, Princeton Univ. Press, Princeton, NJ, 1968.

[15] M. I. Minski and S. Papert, *Perceptrons*, MIT Press, Amherst, Mass., 1969.

J. W. Moon, Various proofs of Cayley's formula for counting trees, in *A Seminar on Graph Theory*. Edited by F. Harary. Holt, Rinehart and Winston, New York, 1976.

[13] D. E. Muller and F. P. Preparata, Finding the intersection of two convex polyhedra, *Theoretical Computer Science* **7**(2), 217–236 (Oct. 1978).

K. P. K. Nair and R. Chandrasekaran, Optimal location of a single service center of



[16] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, 1973.

J. Nievergelt and F. Preparata, Plane-sweep algorithms for intersecting geometric figures, *Comm. ACM* **25**(10), 739–747 (1982).

A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms*, Academic Press, New York, 1975.

J. O'Rourke, C.-B. Chien, T. Olson and D. Naddor, A new linear algorithm for intersecting convex polygons, *Computer Graphics and Image Processing* **19**, 384–391 (1982).

R. E. Osteen and P. P. Lin, Picture skeletons based on eccentricities of points of minimum spanning trees, *SIAM J. Comput.* **3**(1), 23–40 (March 1974).

M. H. Overmars, The design of dynamic data structures, Ph. D. Thesis, University of Utrecht, The Netherlands (1983).

M. H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Comput. and Syst. Sci.* **23**, 166–204 (1981).

C. H. Papadimitriou, The Euclidean traveling salesman problem is NP-complete, *Theoret. Comput. Sci.* **4**, 237–244 (1977).

C. H. Papadimitriou and K. Steiglitz, Some complexity results for the traveling salesman problem, *Eighth ACM Annual Symp. on Theory of Comput.*, pp. 1–9 (May 1976).

E. Pielou, *Mathematical Ecology*, Wiley-Interscience, New York, 1977.

F. P. Preparata, Steps into computational geometry, Technical Report. Coordinated Science Laboratory, University of Illinois, 1977.

F. P. Preparata, An optimal real time algorithm for planar convex hulls, *Comm. ACM* **22**, 402–405 (1979).

F. P. Preparata, A new approach to planar point location, *SIAM J. Comput.* **10**(3), 473–482 (1981).

F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Comm. ACM* **2**(20), 87–93 (Feb. 1977).

[17] F. P. Preparata and D. E. Muller, Finding the intersection of $n$ half-spaces in time $O(n \log n)$, *Theoretical Computer Science* **8**(1), 45–55 (Jan. 1979).

F. P. Preparata and K. J. Supowit, Testing a simple polygon for monotonicity, *Info. Proc. Lett.* **12**(4), 161–164 (Aug. 1981).

[18] R. C. Prim, Shortest connecting networks and some generalizations, *BSTJ* **36**, 1389–1401 (1957).

M. O. Rabin, Proving simultaneous positivity of linear forms, *Journal of Computer and System Sciences* **6**, 639–650 (1972).

[19] H. Rademacher and O. Toeplitz, *The Enjoyment of Mathematics*, Princeton University Press, Princeton, NJ, 1957.

H. Raynaud, Sur l'enveloppe convexe des nuages des points aléatoires dans $R^n$, I, *J. Appl. Prob.* **7**, 35–48 (1970).

E. M. Reingold, On the optimality of some set algorithms, *Journal of the ACM* **19**, 649–659 (1972).

[20] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

A. Rényi and R. Sulanke, Ueber die konvexe Hulle von $n$ zufallig gewahlten Punkten, I, *Z. Wahrschein.* **2**, 75–84 (1963).

R. Riesenfeld, Applications of b-spline approximation to geometric problems of computer-aided design, Technical Report UTEC-CSc-73-126. Department of Computer Science, University of Utah, 1973.

[23] R. T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, NJ 1970.

[21] C. A. Rogers, *Packing and Covering*, Cambridge University Press, Cambridge, England, 1964.

[22] [illegible] *Pictures Processing by Computers*, Academic Press, New York, 1969.

D. J. Rosenkrantz, R. E. Stearns and P. M. Lewis, Approximate algorithms for the traveling salesperson problem, *Fifteenth Annual IEEE Symposium on Switching and Automata Theory*, pp. 33–42 (May 1974).

[24] T. L. Saaty, *Optimization in Integers and Related Extremal Problems*, McGraw-Hill, New York, 1970.

[25] L. A. Santaló, *Integral Geometry and Geometric Probability*, Encyclopedia of Mathematics and Its Applications, Vol. 1. Addison-Wesley, Reading, Mass., 1976.

J. B. Saxe, On the number of range queries in k-space, *Discrete Applied Mathematics* **1**, 217–225 (1979).

J. B. Saxe and J. L. Bentley, Transforming static data structures into dynamic structures, *Proc. 20th IEEE Annual Symp. on Foundations of Comput. Sci.*, pp. 148–168 (1979).

A. M. Schönhage, M. Paterson and N. Pippenger, Finding the median, *J. Comput. and Syst. Sci.* **13**, 184–199 (1976).

J. T. Schwartz, Finding the minimum distance between two convex polygons, *Info. Proc. Lett.* **13**(4), 168–170 (1981).

D. H. Schwartzmann and J. J. Vidal, An algorithm for determining the topological dimensionality of point clusters, *IEEE Trans. Comp.* **C-24**(12), 1175–1182 (Dec. 1975).

R. Seidel, A convex hull algorithm optimal for points in even dimensions, M.S. thesis, Tech. Rep. 81-14, Dept. of Comput. Sci., Univ. of British Columbia, Vancouver, Canada, 1981.

R. Seidel, The complexity of Voronoi diagrams in higher dimensions, *Proc. 20th Allerton Conference on Comm., Control and Comput.*, pp. 94–95 (1982). See also Tech. Rep. No. F94, Technische Universität Graz, Austria, 1982.

M. I. Shamos, Problems in computational geometry, unpublished manuscript, 1975a.

M. I. Shamos, Geometric Complexity, *Seventh ACM Annual Symp. on Theory of Comput.*, pp. 224–233 (May 1975b).

M. I. Shamos, Geometry and statistics: Problems at the interface, in *Recent Results and New Directions in Algorithms and Complexity*. Edited by J. F. Traub. Academic Press (1976).

M. I. Shamos, Computational geometry, Ph.D. thesis, Dept. of Comput. Sci., Yale Univ., 1978.

M. I. Shamos and D. Hoey, Closest-point problems, *Sixteenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 151–162 (Oct. 1975).

M. I. Shamos and D. Hoey, Geometric intersection problems, *Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 208–215 (Oct. 1976).

H. W. Six and D. Wood, The rectangle intersection problem revisited, *BIT* **20**, 426–433 (1980).

H. W. Six and D. Wood, Counting and reporting intersections of *d*-ranges, *IEEE Trans. Comput.* **C-31**, 181–187 (1982).

J. Sklansky, Measuring concavity on a rectangular mosaic, *IEEE Trans. Comp.* **C-21**, 1355–1364 (1972).

E. Soisalon-Soininen and D. Wood, An optimal algorithm for testing for safety and detecting deadlock in locked transaction systems, *Proc. of ACM Symposium on Principles of Data Bases*, Los Angeles, March 1982; pp. 108–116.

J. M. Steele and A. C. Yao, Lower bounds for algebraic decision trees, *J. Algorithms* **3**, 1–8 (1982).

J. Stoer and C. Witzgall, *Convexity and Optimization in Finite Dimensions I*, Springer-Verlag, New York, 1970.

[26] G. Strang and G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.

K. J. Supowit, The relative neighborhood graph with an application to minim-



I. E. Sutherland, Computer graphics: ten unsolved problems, *Datamation* **12**(5), 22–27 (May 1966).

J. J. Sylvester, On Poncelet's approximate linear valuation of surd forms, *Phil. Mag.* Ser. 4, **20**, 203–222 (1860).

R. Thom, Sur l'homologie des variétés algébraïques réelles, *Differential and Combinatorial Topology*, Edited by S. S. Cairns. Princeton Univ. Press, Princeton, NJ, 1965.

C. Toregas, R. Swain, C. Revelle and L. Bergman, The location of emergency service facilities, *Operations Research* **19**, 1363–1373 (1971).

G. T. Toussaint, Pattern recognition and geometrical complexity, *Proc. 5th Int'l Conference on Pattern Recognition*, pp. 1324–1347 (Dec. 1980a).

G. T. Toussaint, The relative neighborhood graph of a finite planar set, *Pattern Recognition* **12**(4), 261–268 (1980b).

G. T. Toussaint, An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons, *Proc. 21st Allerton Conference on Comm., Control and Comput.*, pp. 457–458 (1983a).

G. T. Toussaint, Computing largest empty circles with location constraints, *Int'l J. Computer and Info. Sci.* **12**(5), 347–358 (1983b).

V. Vaishnavi, Computing point enclosures, *Pattern Recog.* **15**, 22–29 (1982).

V. Vaishnavi and D. Wood, Data structures for the rectangle containment and enclosure problems, *Computer Graphics and Image Processing* **13**, 372–384 (1980).

V. Vaishnavi and D. Wood, Rectilinear line segment intersection, layered segment trees, and dynamization, *J. Algorithms* **3**, 160–176 (1982).

F. A. Valentine, *Convex Sets*, McGraw-Hill, New York, 1964.

G. Voronoi, Nouvelles applications des parametres continus à la theorie des formes quadratiques. Deuxième Mémorie: Recherches sur les paralléloèdres primitifs, *J. reine angew. Math.* **134**, 198–287 (1908).

J. E. Warnock, A hidden-surface algorithm for computer generated halftone pictures, Technical Report TR 4-15. Computer Science Department, University of Utah, 1969.

G. S. Watkins, A real-time visible surface algorithm, Technical Report UTECH-CSc-70-101. Computer Science Department, University of Utah, June 1970.

D. E. Willard, Predicate-oriented database search algorithms, Harvard University, Cambridge, MA, Aiken Computation Laboratory, Ph.D. Thesis, Report TR-20-78, 1978.

[2] D. E. Willard, Polygon retrieval, *SIAM J. Comput.* **11**, 149–165 (1982).

N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[30] I. M. Yaglom and V. G. Boltyanskii, *Convex Figures*, Holt, Rinehart and Winston, New York, 1961.

A. C. Yao, An $O(E \log\log V)$ algorithm for finding minimum spanning trees, *Info. Proc. Lett* **4**, 21–23 (1975).

A. C. Yao, A lower bound to finding convex hulls, *J. ACM* **28**, 780–787 (1981).

A. C. Yao, On contructing minimum spanning trees in k-dimensional space and related problems, *SIAM J. Comput.* **11**(4), 721–736 (1982).

M. Z. Yannakakis, C. H. Papadimitriou and H. T. Kung, Locking policies: safety and freedom for deadlock, *Proceedings 20th Annual Symposium on Foundations of Computer Science*, pp. 286–297 (1979).

C. T. Zahn, Graph-theoretical methods for detecting and describing gestalt clusters, *IEEE Trans. Comp.* **C-20**(1), 68–86 (Jan. 1971).

[7] S. I. Zhukhovitsky and L. I. Avdeyeva, *Linear and Convex Programming*, Saunders, Philadelphia, 1966.

## Работы вышедшие на русском языке или в русском переводе


1. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.
2. Вирт Н. Алгоритмы + структуры данных = программы. — М.: Мир, 1985.
3. Гильберт Д. Основания геометрии. — М. — Л.: Гостехиздат, 1948.
4. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. — М.: Мир, 1982.
5. Делоне Б. Н. О пустом шаре. — Известия АН СССР, 1934, VII серия, 6, с. 793—800.
6. Дуда Р., Харт П. Распознавание образов и анализ сцен. — М.: Мир, 1976.
7. Зуховицкий С. И., Авдеева Л. И. Линейное и выпуклое программирование. — М.: Наука, 1964.
8. Карп Р. М. Сводимость комбинаторных проблем. — Кибернетический сборник, вып. 12. — М.: Мир, 1975, с. 16—38.
9. Кендэл М., Моран П. Геометрические вероятности. — М.: Наука, 1972.
10. Клейн Ф. Сравнительное обозрение новейших геометрических исследований. (Программа при вступлении в университет в Эрлангене). — Казань, Имп. университет, 1896.
11. Кнут Д. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы. — М.: Мир, 1976.
12. Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. — М.: Мир, 1978.
13. Маллер Д., Препарата Ф. Нахождение пересечения двух выпуклых многогранников. — Кибернетический сборник, вып. 20. — М.: Мир, 1983, с. 5—29.
14. Милнор Дж. Особые точки комплексных гиперповерхностей. — М.: Мир, 1974.
15. Минский М., Пейперт С. Персептроны. — М.: Мир, 1971. — 261 с.
16. Ньюмен У., Спрулл Р. Основы интерактивной машинной графики. — М.: Мир, 1976.
17. Препарата Ф., Маллер Д. Нахождение пересечения $n$ полупространств за время $O(n \log n)$. — Кибернетический сборник, вып. 21. — М.: Мир, 1984, с. 55—68.
18. Прим Р. К. Кратчайшие связывающие сети и некоторые обобщения. — Кибернетический сборник. — М.: Иностранная литература, 1961, с. 95—107.
19. Радемахер Г., Теплиц О. Числа и фигуры. Опыты матем. мышления. — М.: Физматгиз, 1962.
20. Рейнгольд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы: теория и практика. — М.: Мир, 1980.
21. Роджерс К. А. Укладки и покрытия. — М.: Мир, 1968.
22. Розенфельд А. Распознавание и обработка изображений с помощью вычислительных машин. — М.: Мир, 1972.
23. Рокафеллар Р. Т. Выпуклый анализ. — М.: Мир, 1973.
24. Саати Т. Целочисленные методы оптимизации и связанные с ними экстремальные проблемы. — М.: Мир, 1973.
25. Сантало Л. А. Интегральная геометрия и геометрические вероятности. — М.: Наука, 1983.
26. Стренг Г., Фикс Дж. Теория метода конечных элементов. — М.: Мир, 1977.
27. Хадвигер Г., Дебруннер Г. Комбинаторная геометрия плоскости. — М.: Наука, 1965.
28. Ханан М., Курцберг Дж. Методы размещения. — В кн.: Теория и методы автоматизации проектирования вычислительных систем. — М.: Мир, 1977, с. 147—225.





29. Эйкерс Ш. В. Трассировка. — В кн.: Теория и методы автоматизации проектирования вычислительных систем. — М.: Мир, 1977, с. 226—295.
30. Яглом И. М. и Болтянский В. Г. Выпуклые фигуры. М. — Л.: Гостехиздат, 1951.


## Список дополнительной литературы к русскому изданию [1])


1. Aggarwal A., Guibas L., Saxe J., Shor P. W. A linear time algorithm for computing the Voronoi diagram of a convex polygon, Proc. 19th ACM Annual Symp. on Theory of Comput. pp. 39—45, May 1987.
*2. Chazelle B. M., Cole R., Preparata F. P., Yap C. K. New upper bounds for neighbor searching, Information and Control, 68(1—3), p. 105—124 (1986).
*3. Chazelle B. M., Edelsbrunner H. Optimal solutions for a class of point retrieval problems, J. Symbol Comput. 1, p. 47—56, 1985.
4. Chazelle B. M., Edelsbrunner, An optimal algorithm for intersecting line segments, manuscript, 1988.
*5. Edahiro M. I., Kokubo I., Asano T. A new point-location algorithm and its practical efficiency — comparison with existing algorithms, ACM Trans. on Graphics 3(2), p. 86—109, 1984.
*6. Edelsbrunner E., Guibas L. J., Stolfi J. Optimal point location in monotone subdivision, SIAM J. Comput., 15(2), p. 317—340, 1986.
*7. Edelsbrunner H., Overmars M. H. Batched dynamic solutions to decomposable searching problems, J. Algorithms, 6, p. 515—542, 1985.
*8. Edelsbrunner H., Seidel R. Voronoi diagrams and arrangements, Discrete Comput. Geom. 1, p. 25—44, 1986.
*9. Edelsbrunner H., van Leeuwen T. A., Ottman T. A., Wood D. Computing the connected components of simple rectilinear geometric objects in $d$-space, RAIRO Inf. Th. 18, p. 171—183, 1984.
*10. Edelsbrunner H., Welzl E. Halfplanar range search in linear space and $O(n^{0.695})$ query time, Info. Proc. Lett., 23(6), p. 289—293, 1986.
11. Griffiths J. G. Bibliography of hidden-line and hidden-surface algorithms, Computer Aided Design, 10(3), p. 203—206, 1978.
12. Güting R. H., Ottmann T. New algorithms for special cases of hidden-line elimination problems, Lecture Notes Comput. Sci. 182, p. 161—172, 1985.
*13. Hertel S., Mehlhorn K., Mäntylä M., Nievergelt J. Space sweep solves intersection of two convex polyhedron elegantly, Acta Informatica, 21, p. 501—519, 1984.
*14. Jaromczyk J. W. Lower bounds for polygon simplicity testing and other problems. Lecture Notes Comput. Sci., 176, p. 339—347, 1984.
*15. Kirkpatrick D. G. Seidel R. The ultimate planar convex hull algorithm? SIAM J. Comput., 15(1), p. 287—299, 1986.
16. Lee D. T., Wu Y. F., Geometric complexity of some location problems, Algorithmica, 1, p. 193—211, 1986.
*17. Manber U., Tompa M. The complexity of problems on probabilistic, nondeterministic and alternating decision trees, J. ACM, 32(3), p. 720—732, 1985.


[1]) Работы с номерами 11, 12, 21 добавлены переводчиком.

*18. Mairson H. G., Stolfi J. Reporting and counting intersections between two sets of line segments, Proc. NATO Adv. Study Inst. on Theoret. Found. Comput. Graphics and CAD, Sprihger-Verlag, 1987.
*19. McCreight E. M., Priority Search Trees, SIAM J. Comput., 14(2), p. 257—276, 1985.
20. Tarjan R. E., van Wyk C. J. An $O(n \log\log n)$-time algorithm for triangulating a simple polygon, SIAM J. Comput., 17, p. 143—178, 1988.
21. Фокс А., Пратт М. Вычислительная геометрия. Применение в проектировании и на производстве. — М.: Мир, 1982.

# Именной указатель

# Предметный указатель

# Оглавление

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И — 110, ГСП, 1-й Рижский пер., д. 2, издательство «Мир».

Научное издание

Франко Препарата
Майкл Шеймос

ВЫЧИСЛИТЕЛЬНАЯ ГЕОМЕТРИЯ:
ВВЕДЕНИЕ

Зав. редакцией чл.-корр. АН СССР В. И. Арнольд
Зам. зав. редакцией А. С. Попов
Научн. редактор М. В. Хатунцева
Мл. научн. редактор Л. А. Королева
Художник Г. М. Чеховский
Художественный редактор В. И. Шаповалов
Технический редактор З. И. Резник
Корректор М. А. Смирнов

ИБ № 6477

Сдано в набор 17.05.88. Подписано к печати 27.12.88. Формат 60×90 1/16. Бумага кн.-журн. Гарнитура латинская. Печать высокая. Объем 15 бум. л. Усл. печ. л. 30. Усл. кр.-отт. 30. Уч.-изд. л. 30,42. Изд. № 1/5697. Тираж 10 000 экз. Заказ 471. Цена 2 р. 80 к.
Издательство «Мир»
Москва, 1-й Рижский пер., 2.

Отпечатано с набора Ленинградской типографии № 2 головного предприятия ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 198052, г. Ленинград, Л-52, Измайловский пр., 29 в Ленинградской типографии № 4 ордена Трудового Красного Знамени Ленинградского объединения «Техническая книга» им. Евгении Соколовой Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 191126, Ленинград, Социалистическая ул., 14.