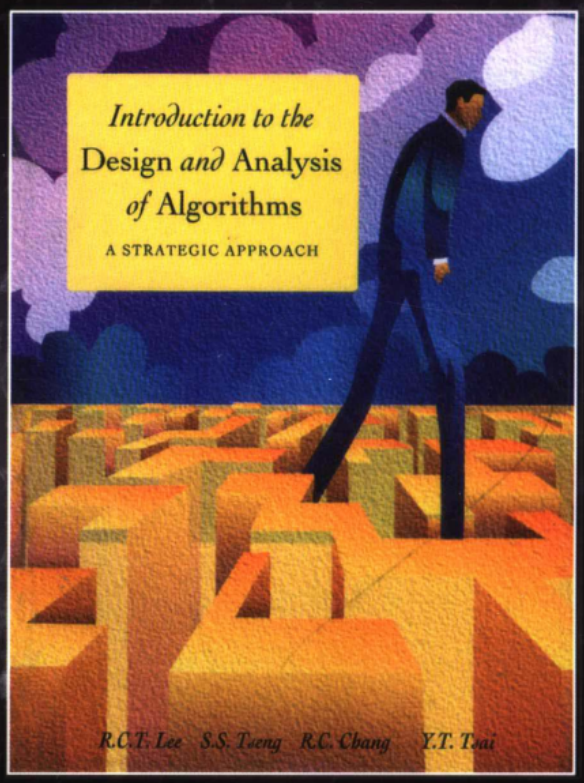
计 算 机 科 学 丛 书

算法设计与分析导论

R. C. T. Lee S. S. Tseng R. C. Chang Y. T. Tsai 著 王卫东 译



Introduction to the Design and Analysis of Algorithms

A Strategic Approach

机械工业出版社
China Machine Press

算法设计与分析导论

通信网络设计、VLSI布局和DNA序列分析是重要而具有挑战性的问题，不能用初级算法解决。因此，对于计算机科学家来说，掌握良好的算法设计和分析的知识系统是十分重要的。

本书从算法策略的角度来描述算法设计。每个策略包含许多基于此策略的算法设计。对于每个算法，用丰富的实例进行诠释。另外，每个例子都采用详细的图示。

近年来，许多近似算法相继开发出来。本书清晰地描述了其中的两个重要概念：PTAS和NPO完全性。在介绍近似算法之前，本书对NP完全性的概念进行了讨论，并通过大量的具体实例进行解释，目的是使学生对这个很抽象的概念有明确的认识。

另外，本书还介绍了在线算法的专题，每个在线算法通过先描述其内在的基本原理来展开介绍。分摊分析是算法研究的一个新领域，本书对这个不易理解的新概念也进行了详细的介绍。

本书可以作为计算机科学专业高年级本科生或硕士研究生的教材使用。

作者简介

R. C. T. Lee（李家同） 1939年生于上海，台湾大学电机系学士，美国加州伯克利大学电机博士。历任台湾清华大学工学院院长、教务长以及代校长，静宜大学校长，暨南大学校长，现任暨南大学教授。李教授是美国电机电子学会的荣誉会士，并且曾担任过11种国际学术刊物的编辑委员。其在算法和逻辑方面的著作曾被译为多种文字出版。

S. S. Tseng（曾宪雄）和 R. C. Chang（张瑞川） 二人都获得了台湾交通大学博士学位，现任台湾交通大学计算机与信息科学系教授。

Y. T. Tsai（蔡英德） 台湾交通大学信息工程系硕士，台湾清华大学信息科学系博士。现任台湾静宜大学信息传播工程学系教授兼系主任。

Mc Graw Hill Education

上架指导：计算机/算法设计

ISBN 978-7-111-22504-1

9 787111 225041

ISBN 978-7-111-22504-1

定价：49.00 元

投稿热线：(010) 88379604

购书热线：(010) 68995259, 68995264

读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

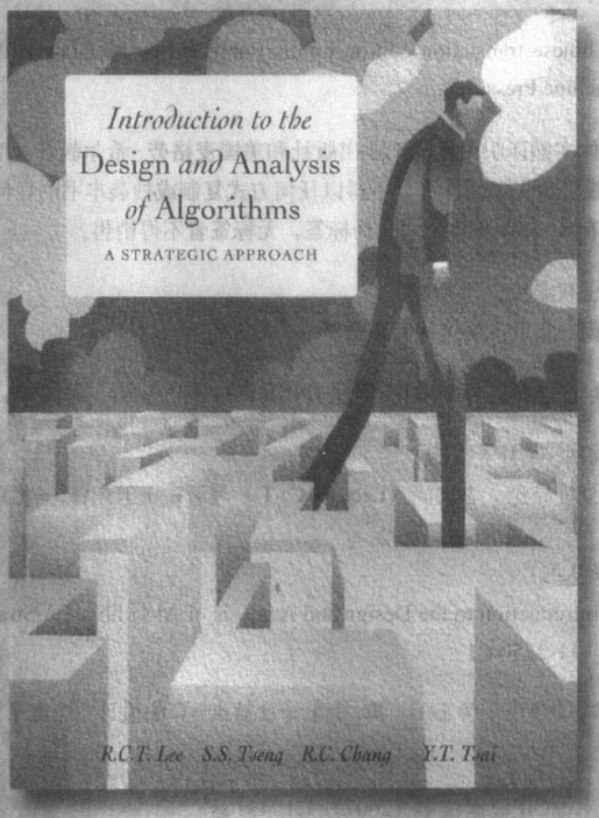
网上购书：www.china-pub.com

封面设计：锡林

计 算 机 科 学 丛 书

算法设计与分析导论

R. C. T. Lee S. S. Tseng R. C. Chang Y. T. Tsai 著 王卫东 译



Introduction to the Design and Analysis of Algorithms

A Strategic Approach

机械工业出版社
China Machine Press

本书在介绍算法时，重点介绍用于设计算法的策略，非常与众不同。书中介绍了剪枝搜索、分摊分析、随机算法、在线算法以及多项式近似方案等相对较新的思想和众多基于分摊分析新开发的算法，每个算法都与实例一起加以介绍，而且每个例子都利用图进行详细解释。此外，本书还提供了超过400幅图来帮助初学者理解。

本书适合作为高等院校算法设计与分析课程的高年级本科生和低年级研究生的教材，也可供相关科技人员和专业人士参考使用。

R. C. T. Lee, S. S. Tseng, R. C. Chang, Y. T. Tsai: Introduction to the Design and Analysis of Algorithms: A Strategic Approach (ISBN 10: 0-07-124346-1).

本书版权登记号：图字：01-2007-0469

图书在版编目（CIP）数据

算法设计与分析导论 / 李家同（Lee, R. C. T.）等著；王卫东译. -北京：机械工业出版社，2007.10

（计算机科学丛书）

书名原文：Introduction to the Design and Analysis of Algorithms: A Strategic Approach

ISBN 978-7-111-22504-1

Ⅰ. 算…　Ⅱ. ①李…　②王…　Ⅲ. ①电子计算机-算法设计　②电子计算机-算法分析　Ⅳ. TP301.6

中国版本图书馆CIP数据核字（2007）第154316号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：王　玉

北京瑞德印刷有限公司印刷 · 新华书店北京发行所发行

2008年1月第1版第1次印刷

184mm × 260mm · 24.25印张

定价：49.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近260个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”。为了保证这两套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这两套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、

编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@hzbook.com
联系电话：(010) 68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

译 者 序

本书是算法设计与分析方面的优秀著作。它从策略的角度详细地阐述了关于算法设计的多种设计策略。设计算法的基本原则，也就是策略，显然比算法本身更重要。本书利用算法设计策略作为指导进行算法设计，通过大量实例介绍了设计策略的基本原理，同时强调了对每种算法详细的复杂性分析。

本书的内容具有相当的广度和深度，这包括对各种策略的基本机制、基础算法、基本算法设计技术、算法的复杂度分析等，具体包括算法复杂度与问题的下界、贪心法、分治策略、树搜索策略、剪枝搜索、动态规划、NP完全性理论、近似算法、分摊分析、随机算法和在线算法等。

本书的组织方式简明扼要、循序渐进，涵盖了大多数算法设计中的一般技术，同时包含了剪枝搜索、分摊分析、在线算法以及多项式近似方案等相对较新的思想，这些都是近20年来算法研究中发展迅猛的领域。另外，对于两个重要概念：PTAS和NPO完全性做了清晰的解释。对于在线算法、近似算法、随机算法等结合实例做了详尽的介绍。对于较难理解的分摊分析给出了理论证明。

算法设计一直是备受关注的研究领域之一。因此，本书还介绍了算法设计与分析领域的最新进展、最新成果和特别声明，使读者能够容易地找到进一步研究的方向。

我们相信本书中译本的出版将对国内高校计算机专业算法课程的教学起到积极的推动作用。

由于译者水平有限，翻译中难免有错误和不妥之处，恳请读者批评指正。

王卫东

西安电子科技大学计算机学院

2007年7月

前　言

研究算法可能有许多原因，其中的主要原因是能够使我们更高效地使用计算机。实践证明，一位不具备很好算法知识的新手，对于其老板及工作单位来说都是一个大麻烦。假如某人要找一棵最小生成树，设想现在或者将来都不存在计算机，那么他通过检查所有可能的生成树来确定最小生成树的做法，对他来说也许足够好了。但是，如果他了解Prim算法，那么有一台IBM个人计算机就足以解决问题。另一个例子是某人要解决语音识别问题，对他来说如何开始是相当困难的。但是，如果他了解能够使用动态规划方法解决的最长公共子序列问题，那么他将会明白该问题是相当容易解决的。算法的研究不仅对于计算机科学家非常重要，而且通信工程师在编码时也要用到动态规划或A^*算法。有许多其他行业的科学家从研究算法中获益匪浅，其中相当一部分是从事分子生物学研究的科学家。当他们要比较两个DNA序列、两个蛋白质或者三维结构的RNA时，需要了解非常复杂的算法。

此外，研究算法也有很多乐趣。由于我们研究算法已经很长时间，因此在任何时候看到新的设计良好的算法时，或者想出关于设计和分析算法的一些新颖想法时，都会异常兴奋。因此，我们也感到有责任让更多的人共同分享这种乐趣和兴奋。许多看似困难的问题实际上可以通过多项式时间算法解决，而一些看似普通的问题却被证实是NP完全的。最小生成树问题对许多初学者来说似乎是相当困难的，但它却有多项式时间的解决算法。如果稍微对它改动一点就成为旅行商问题，那么它就成为一个NP难问题。另一个例子是3可满足性问题，它是一个NP完全问题。通过降低维数，2可满足性问题将会成为P问题。发现这些事实总是令人着迷的。

在本书中，我们将采取一种与众不同的方法来介绍算法。实际上，我们并不是在介绍算法，而是在介绍用于设计算法的策略。原因很简单：明白用于设计算法的基本原则（也就是策略）显然比算法本身更重要。尽管并非每个算法都基于本书所介绍的某个策略，但大部分是这样的。

剪枝搜索、分摊分析、在线算法以及多项式近似方案都是相对较新的思想，而且是十分重要的思想。众多新开发的算法都是基于分摊分析的，正如我们在关于该主题的章末将看到的那样。

我们将从用来设计多项式时间算法的策略开始介绍。在必须解决看似困难但目前的确还没有多项式时间算法的问题时，我们会介绍NP完全性的概念。尽管经常很难把握它的物理意义，但应用NP完全性的概念还是比较容易的。实际上，其核心思想是为什么每个NP问题实例都关联一个布尔公式集，并且当且仅当该公式集可满足时，该问题实例的回答是“yes”。一旦读者明白了这个思想，就会懂得NP完全性的重要性。我们相信用于解释这个思想而提供的实例将会帮助大多数学生很容易地理解NP完全性的概念。

本书可用作高年级本科生和低年级研究生的教材。依据我们的经验，如果在一个学期（约50学时）内讲授本书，那么无法涵盖书中全部的内容。因此，如果只有一个学期，那么我们建议所有章都要讲，但不必都完全讲透。不要忽略任何章！关于NP完全性的章是非常重要的，应该让学生理解清楚这章的内容。最难理解的章是第10章（分摊分析），其中涉及许多数学知识。教师在授课时应当非常注意讲解分摊分析的基本思想，而不应过多地讲述相关的数学证明。换句话说，学生应该能理解为什么与好算法相结合的特定数据结构在分摊的意义上执行得非常好。另一个相对困难的章可能是第12章（在线算法）。

许多算法并不容易理解，我们将尽量清晰地介绍这些算法。介绍每个算法时都辅以实例，而每个实例都利用图进行解释。在本书中，提供了超过400幅图来帮助初学者理解。

我们还引用了关于算法设计与分析的大量书籍和论文；并特别声明最新的成果，使读者能够容易地找到进一步研究的方向。“参考文献”中列出了825本书籍和论文，共涉及1 095位作者。

我们还会在适当的时候提供实验的结果。但作为教师，还是应当鼓励学生通过实现算法来验证结论。在每一章的结尾有进一步阅读的论文列表，鼓励学生阅读这些论文以提升他们对算法的理解是非常重要的。希望学生能领会到作者不遗余力地解释大量难以理解的论文是多么辛苦！书中还提供一些用Java编写的程序，便于学生实践。

我们在此不可能列举出在准备本书的过程中帮助过我们的所有人的名字，因为实在是太多了。但他们都属于一个班，是我们的学生或者同事（许多学生后来成为同事）。在每周五晚上的讨论会上，我们总是积极活跃地讨论。这些讨论指出了算法研究中的新方向，也有助于我们决定本书应该包含哪些资料。研究生们一直在追踪大约20种学术期刊，并确保关于算法的每篇重要论文都存储在数据库中，且都关联上关键字。这个数据库对于写作本书是非常有价值的。最后，他们还阅读了本书的手稿，提出了批评意见，并完成了实验。如果没有来自同事和学生们的帮助，我们是不可能完成本书的。在这里，我们对他们所有人表示无限的感谢。

目　录

第1章 绪　论

本书介绍算法的设计与分析，首先讨论下面这个非常重要的问题或许是有意义的：为什么要研究算法？通常认为是为了获得高速的计算，但这只要一台高速计算机就够了。这显然并不完全正确。我们通过一个实验来说明，实验结果清楚地表明：在低速计算机上执行好算法比在高速计算机上执行差算法完成得更好。考虑下面简单描述的两个排序算法：插入排序（insertion sort）和快速排序（quick sort）。

插入排序是从左到右一个接一个地考查数据元素序列。在考查完每个元素之后，将元素插入已有序序列的一个适当位置。例如，假设将要排序的数据元素序列是

11，7，14，1，5，9，10

对上面的数据元素序列进行的插入排序如下：

有序序列	无序序列
11	7，14，1，5，9，10
7，11	14，1，5，9，10
7，11，14	1，5，9，10
1，7，11，14	5，9，10
1，5，7，11，14	9，10
1，5，7，9，11，14	10
1，5，7，9，10，11，14	

在上面的排序过程中，考虑当前有序序列是1，5，7，11，14的情况，下一个将要排序的元素是9。先将9与14比较，由于9比14小，接着将9与11比较。必须再一次继续比较，在将9与7比较之后，确定9比7大，因此，将9插入到7和11之间。

所举的第二个排序算法称为快速排序，将在第2章中作详细介绍。目前，只给出该算法一个较高层次的描述。设想对下列数据元素进行排序：

10，5，1，17，14，8，7，26，21，3

快速排序选用第一个数据元素，也就是10，将整个数据元素分成三个子集合：一部分比10小，一部分比10大，以及等于10的部分。这样将数据集合表示成如下：

(5，1，8，7，3)（10）(17，14，26，21)

现在必须对下面两个子序列排序：

(5，1，8，7，3)

和（17，14，26，21)

注意到它们是可以各自独立地进行排序的，划分的方法也可在两个子集上递归地实现。例如，考虑子序列

17，14，26，21

使用17划分上面的数据集合，可以得到

(14)（17）(26，21)

在将上面的序列26，21排序成21，26之后，可以得到

14，17，21，26

这是有序的序列。

相类似地，

(5，1，8，7，3)

可排序成

(1，3，5，7，8)

合并所有的有序序列，可以得到

1，3，5，7，8，10，14，17，21，26

这是最终有序的序列。

后面将证明快速排序比插入排序好得多。问题是：有多好？为了将快速排序与插入排序作比较，我们在Intel 486上实现快速排序，在IBM SP2上实现插入排序。IBM SP2是超级计算机，在1997年曾击败过象棋大师，而Intel 486只是普通的个人计算机。图1-1提供了实验结果，对于每个点上的数字，10个数据集合都是随机产生的，并得到了两个算法的平均时间。可以看到当数据元素少于400时，执行快速排序的Intel 486劣于执行插入排序的IBM SP2，而当数据元素大于400时，Intel 486完成的要比IBM SP2好得多。

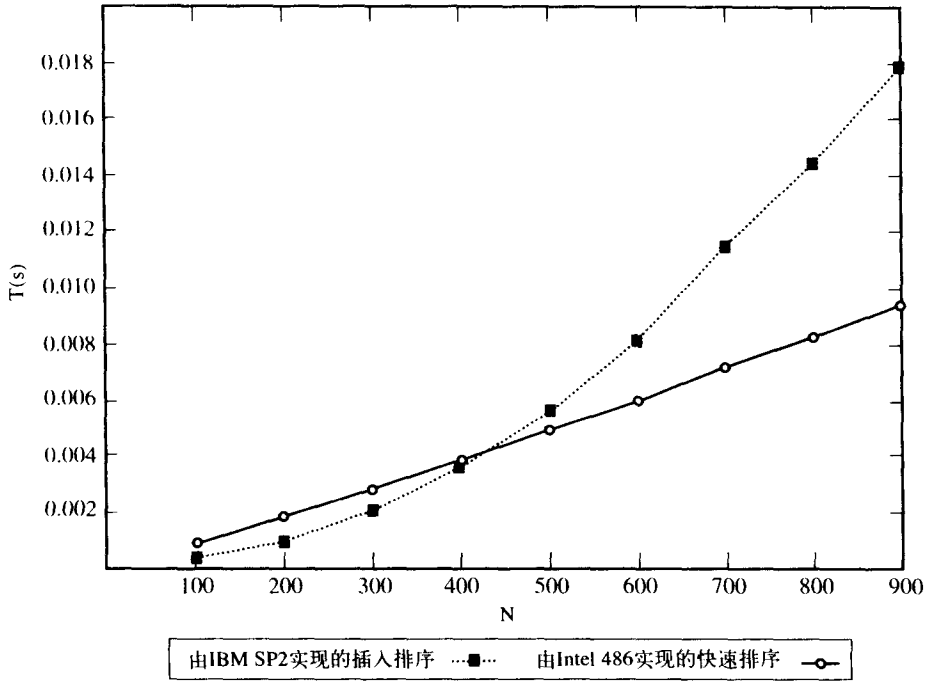


图1-1 插入排序与快速排序性能比较

上面的实验意味着什么呢？显然它表明一个重要事实：在高速计算机上执行低效算法比在低速计算机上执行高效算法差很多。换句话说，如果你很有经验，但没有好的算法知识，那么你也许不能够与一位对算法非常了解的笨拙同事相竞争。

如果承认研究算法是重要的，就必须能够分析算法以确定它们的性能。在第2章中，将对

算法分析相关的一些基本概念作简要介绍。在本章中，对算法分析在各种意义上都是介绍性的，在第2章末将介绍一些优秀的算法分析书籍。

在介绍了算法分析的概念后，将注意力转向问题的复杂度。我们要指出问题有易问题（easy problem）和难问题（difficult problem）之分。如果一个问题能够被有效算法解决，多半算法具有多项式时间（polynomial-time）复杂度，那么该问题称为易问题。反之，如果一个问题不能够被任何具有多项式时间的算法解决，那么它一定是难问题。通常，给定一个问题，如果已经知道存在多项式时间内解决该问题的算法，就可以确认它是一个易问题。然而，如果到目前还没有发现任何解决该问题的多项式时间算法，也不能推断在将来就找不到解决该问题的多项式算法。幸运的是，有一种用于度量问题复杂度的NP完全性（NP-completeness）理论。如果一个问题被证实是NP完全的（NP-complete），那么它被看作是个难问题，并且找到解决该问题的多项式算法的概率是非常小的。通常，NP完全性的概念都会在教科书的末尾才介绍。

许多问题看似不像是难问题，而事实上却是NP完全问题，这是非常有趣的。看下面一些例子。

首先，考虑0/1背包问题（0/1 knapsack problem）。这个问题非正式描述如下：假如我们要躲避入侵的军队必须离开心爱的家，随身携带一些有用的物品。但是，携带物品的总重量不能超过某个限制。在不超过重量限制的情况下，如何最大化所携带物品的价值呢？例如，假设有下列物品：

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
价值	10	5	1	9	3	4	11	17
重量	7	3	3	10	1	9	22	15

如果重量限制是14，那么最好的解是选择P_1，P_2，P_3和P_5。该问题是NP完全问题，随着物品的数目增大，很难找到一个最优解。

另一个看似容易的问题也是NP完全问题，这就是旅行商问题（traveling salesperson problem）。为了直观地解释该问题，考虑有许多城市，勤劳的推销商必须遍历每一座城市，每座城市不能遍历两次并且要求遍历路径最短。例如，一个旅行商问题实例的最优解如图1-2所示，其中两座城市间的距离是它们的欧几里得距离（Euclidean distance）。

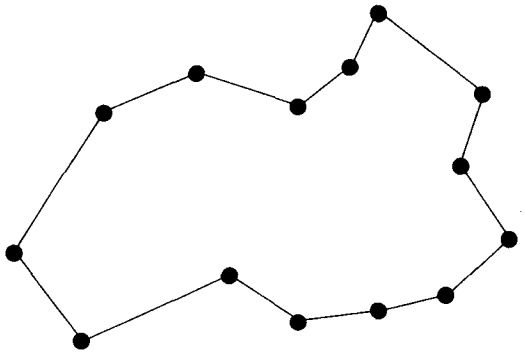


图1-2　一个旅行商问题实例的最优解

最后，考虑分割问题（partition problem），即对于给定的整数集合，能否划分这些整数为两个子集合S_1和S_2，使得S_1的和等于S_2的和？例如，对于下面的集合

$$\{1, 7, 10, 9, 5, 8, 3, 13\}$$

可以划分上面的集合为

$$S_1 = \{1, 10, 9, 8\}$$

和　$S_2 = \{7, 5, 3, 13\}$。

可以验证S_1的和等于S_2的和。

我们经常需要解决这样的分割问题。例如，公钥加密方案就涉及这样的问题，这至今仍然是一个NP完全问题。

在这里还有另一个许多读者都会感兴趣的问题。我们都知道藏有许多无价珍宝的美术陈列馆，这些珍宝是不能被偷盗和损坏的，所以在陈列馆中安排有警卫。图1-3显示了一个美术陈列馆。

在图1-3中，如果在陈列馆中安排4名警卫，用小圆圈表示，陈列馆的每面墙至少由一名警卫监控。这意味着整个陈列馆被这4名警卫有效地监控。美术陈列馆问题（art gallery problem）是：给定一个多边形形状的陈列馆，确定最少的警卫数以及监控整个陈列馆的警卫所在的位置。令许多读者惊讶的是该问题也是一个NP完全问题。

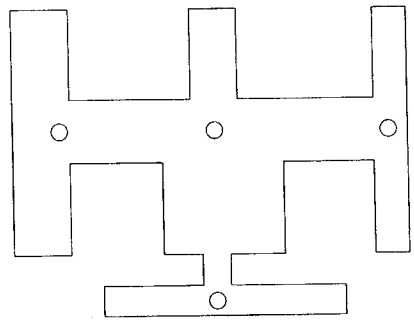


图1-3 美术陈列馆及安排的警卫

即使意识到好算法是最本质的这个事实，人们仍然怀疑研究算法设计是否是重要的，因为好算法很容易获得。换句话说，如果好算法能够简单地通过直观或一般常识就可得到，那么就不值得费力去研究算法的设计。

下面在介绍最小生成树问题时，将会看到这个表面上看似组合爆炸的问题，实际上有非常简单的算法能够高效解决它。下面给出最小生成树问题的非形式化描述。

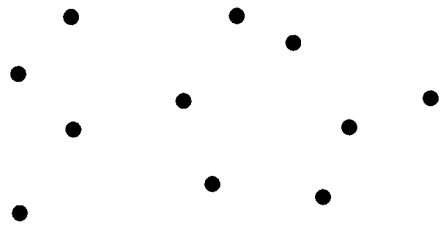


图1-4 说明最小生成树问题的城市集合

想象有许多城市，如图1-4所示。假设要连接所有城市成为一棵生成树（spanning tree）（生成树是连接所有城市而没有环的图），使得生成树的总长度最小。例如，对于如图1-4所示的城市集合，其最小生成树（minimum spanning tree）如图1-5所示。

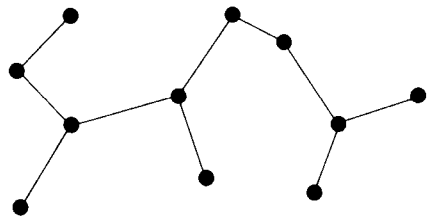


图1-5 对于图1-4所示城市集合的一棵最小生成树

找出这样的最小生成树的直观算法是将所有可能的生成树都找出来，并确定每棵生成树的总长度。在穷举完所有可能的生成树之后，那么最小生成树就得到了。因为可能的生成树数量的确非常巨大，所以这样花费的代价也非常大。已经证明对于n座城市，可能的生成树的总数目是n^{n-2}。假如$n = 10$，将要枚举10^8棵树。当$n = 100$时，这个数目将增加到$100^{98}(10^{196})$，对于如此大的数字，是没有计算机可以处理的。在美国，电话公司可能必须处理n为5 000的情况。因此，必须找到更好的算法。

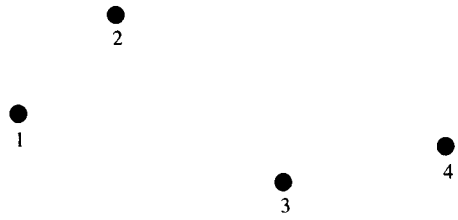


图1-6 说明一个有效的最小生成树算法的例子

事实上，解决这样的最小生成树问题有一个优秀的算法。我们通过举例来说明这个算法。如图1-6所示，算法首先找出d_{12}是所有城市间的最短距离，所以连接城市1和2，如图1-7a所示。之后，将{1，2}作为一个集合，剩余的其他城市作为另一个集合。找出两个城市集合间的最短距离是d_{23}，连接城市2和3。最后，连接城市3和4，整个过程如图1-7所示。

可以证明这个简单有效的算法总是能得到一个最优解，也就是该算法得到的生成树总是最小生成树。该算法的正确性证明并不容易，隐藏在此算法之后的策略称为贪心法（greedy method）。通常，基于贪心法的算法是相当有效的。

遗憾的是，许多类似于最小生成树的问题却不能使用贪心法解决。例如，旅行商问题就是这样的。

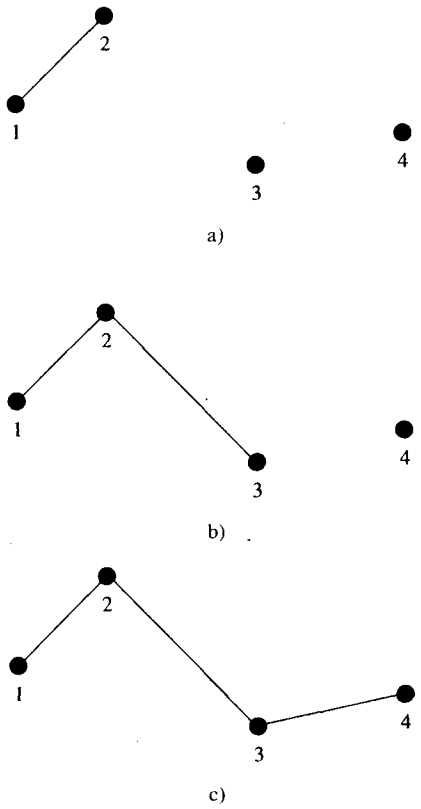


图1-7　最小生成树算法举例

不易使用穷举搜索的另一个例子是1中心问题（1-center problem）。在1中心问题中，给定一个点集合，必须找到一个覆盖所有点的圆，使圆的半径最小。例如，图1-8显示了一个1中心问题的最优解。对于该问题如何开始解答呢？本书的后续章节将说明1中心问题可以使用剪枝搜索策略（prune-and-search strategy）来解决。

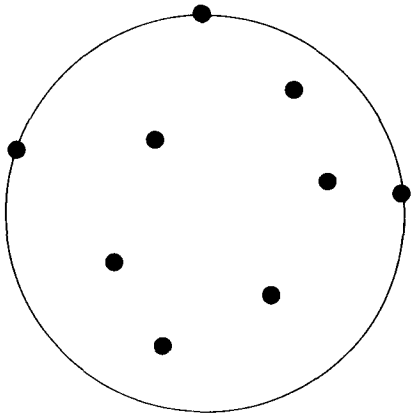


图1-8　1中心问题的解

算法设计的研究几乎就是对策略的研究。由于众多研究者的努力，发现了许多优秀的策略，这些策略可以用于设计算法。我们不能断言每个优秀的算法一定是基于某个一般性的策略，但是，可以明确地说完备的策略知识对于算法的研究绝对是非常重要的。

现在，推荐一些关于算法的书目，它们适合于作为进一步阅读的参考。

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974): *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Basse, S. and Van Gelder, A. (2000): *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Reading, Mass.

Brassard, G. and Bratley, P. (1988): *Algorithmics: Theory and Practice*, Prentice-Hall, Englewood Cliffs, New Jersey.

Coffman, E. G. and Lueker, G. S. (1991): *Probabilistic Analysis of Packaging & Partitioning Algorithms*, John Wiley & Sons, New York.

Cormen, T. H. (2001): *Introduction to Algorithms*, McGraw-Hill, New York.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990): *Introduction to Algorithms*, McGraw-Hill, New York.

Dolan A. and Aldous J. (1993): *Networks and Algorithms: An Introductory Approach*, John Wiley & Sons, New York.

Evans, J. R. and Minieka, E. (1992): *Optimization Algorithms for Networks and Graphs*, 2nd ed., Marcel Dekker, New York.

Garey, M. R. and Johnson, D. S. (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, California.

Gonnet, G. H. (1983): *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, Mass.

Goodman, S. and Hedetniemi, S. (1980): *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York.

Gould, R. (1988): *Graph Theory*, Benjamin Cummings, Redwood City, California.

Greene, D. H. and Knuth, D. E. (1981): *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston, Mass.

Hofri, M. (1987): *Probabilistic Analysis of Algorithms*, Springer-Verlag, New York.

Horowitz, E. and Sahni, S. (1976): *Fundamentals of Data Structures*, Computer Science Press, Rockville, Maryland.

Horowitz, E., Sahni, S. and Rajasekaran, S. (1998): *Computer Algorithms*, W. H. Freeman, New York.

King, T. (1992): *Dynamic Data Structures: Theory and Applications*, Academic Press, London.

Knuth, D. E. (1969): *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass.

Knuth, D. E. (1973): *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass.

Kozen, D. C. (1992): *The Design and Analysis of Algorithms*, Springer-Verlag, New York.

Kronsjö, L. I. (1987): *Algorithms: Their Complexity and Efficiency*, John Wiley & Sons, New York.

Kucera, L. (1991): *Combinatorial Algorithms*, IOP Publishing, Philadelphia.

Lewis, H. R. and Denenberg, L. (1991): *Data Structures and Their Algorithms*, Harper Collins, New York.

Manber, U. (1989): *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, Reading, Mass.

Mehlhorn, K. (1987): *Data Structures and Algorithms: Sorting and Searching*, Springer-Verlag, New York.

Moret, B. M. E. and Shapiro, H. D. (1991): *Algorithms from P to NP*, Benjamin Cummings, Redwood City, California.

Motwani, R. and Raghavan P. (1995): *Randomized Algorithms*, Cambridge University Press, Cambridge, England.

Mulmuley, K. (1998): *Computational Geometry:An Introduction through Randomized Algorithms*, Prentice-Hall, Englewood Cliffs, New Jersey.

Neapolitan, R. E. and Naimipour, K. (1996): *Foundations of Algorithms*, D.C. Heath and Company, Lexington, Mass.

Papadimitriou, C. H. (1994): *Computational Complexity*, Addison-Wesley, Reading, Mass.

Purdom, P. W. Jr. and Brown, C. A. (1985): *The Analysis of Algorithms*, Holt, Rinehart and Winston, New York.

Reingold, E., Nievergelt, J. and Deo, N. (1977): *Combinatorial Algorithms, Theory and Practice*, Prentice-Hall, Englewood Cliffs, New Jersey.

Sedgewick, R. and Flajolet, D. (1996): *An Introduction to the Analysis of Algorithms*, Addison-Wesley, Reading, Mass.

Shaffer, C. A. (2001): *A Practical Introduction to Data Structures and Algorithm Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

Smith, J. D. (1989): *Design and Analysis of Algorithms*, PWS Publishing, Boston, Mass.

Thulasiraman, K. and Swamy, M. N. S. (1992): *Graphs: Theory and Algorithms*, John Wiley & Sons, New York.

Uspensky, V. and Semenov, A. (1993): *Algorithms: Main Ideas and Applications*, Kluwer Press, Norwell, Mass.

Van Leeuwen, J. (1990): *Handbook of Theoretical Computer Science: Volume A:Algorithms and Complexity*, Elsevier, Amsterdam.

Weiss, M. A. (1992): *Data Structures and Algorithm Analysis*, Benjamin Cummings, Redwood City, California.

Wilf, H. S. (1986): *Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New York.

Wood, D. (1993): *Data Structures, Algorithms, and Performance*, Addison-Wesley, Reading, Mass.

对于深入的研究，推荐下列图书：

计算几何方面

Edelsbrunner, H. (1987): *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin.

Mehlhorn, K. (1984): *Data Structures and Algorithms 3:Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin.

Mulmuley, K. (1998): *Computational Geometry:An Introduction through Randomized Algorithms*, Prentice-Hall, Englewood Cliffs, New Jersey.

O'Rourke, J. (1998): *Computational Geometry in C*, Cambridge University Press, Cambridge, England.

Pach, J. (1993): *New Trends in Discrete and Computational Geometry*, Springer-Verlag, New York.

Preparata, F. P. and Shamos, M. I. (1985): *Computational Geometry: An Introduction*, Springer-Verlag, New York.

Teillaud, M. (1993): *Towards Dynamic Randomized Algorithms in Computational Geometry*, Springer-Verlag, New York.

图论方面

Even, S. (1987): *Graph Algorithms*, Computer Science Press, Rockville, Maryland.

Golumbic, M. C. (1980): *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York.

Lau, H. T. (1991): *Algorithms on Graphs*, TAB Books, Blue Ridge Summit, PA.

McHugh, J. A. (1990): *Algorithmic Graph Theory*, Prentice-Hall, London.

Mehlhorn, K. (1984): *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, Springer-Verlag, Berlin.

Nishizeki, T. and Chiba, N. (1988): *Planar Graphs: Theory and Algorithms*, Elsevier, Amsterdam.

Thulasiraman, K. and Swamy, M. N. S. (1992): *Graphs: Theory and Algorithms*, John Wiley & Sons, New York.

组合数学方面

Lawler, E. L. (1976): *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shamoys, D. B. (1985): *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons, New York.

Martello, S. and Toth, P. (1990): *Knapsack Problem Algorithms & Computer Implementations*, John Wiley & Sons, New York.

Papadimitriou, C. H. and Steiglitz, K (1982): *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New Jersey.

高级数据结构方面

Tarjan, R. E. (1983): *Data Structures and Network Algorithms*, Society of Industrial and Applied Mathematics, Vol. 29.

计算生物学方面

Gusfield, D. (1997): *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, England.

Pevzner, P, A. (2000): *Computational Molecular Biology: An Algorithmic Approach*, The MIT Press, Boston.

Setubal, J. and Meidanis, J. (1997): *Introduction to Computational Biology, PWS Publishing Company, Boston, Mass.*

Szpankowski, W. (2001): *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York.

Waterman, M. S. (1995): *Introduction to Computational Biology: Maps, Sequences and Genomes*, Chapman & Hall/CRC, New York.

近似算法方面

Hochbaum, D. S. (1996): *Approximation Algorithms for NP-Hard Problems*, PWS Publisher, Boston.

随机算法方面

Motwani, R. and Raghavan, P. (1995): *Randomized Algorithms*, Cambridge University Press, Cambridge, England.

在线算法方面

Borodin, A. and El-Yaniv, R. (1998): *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, England.

Fiat, A. and Woeginger, G. J. (editors) (1998): Online Algorithms: The State of the Arts, *Lecture Notes in Computer Science*, Vol. 1442. Springer-Verlag, New York.

许多学术期刊定期地发表关于算法的论文。下面推荐最为常见的一些：

- *Acta Informatica*
- *Algorithmica*
- *BIT*
- *Combinatorica*
- *Discrete and Computational Geometry*
- *Discrete Applied Mathematics*
- *IEEE Transactions on Computers*
- *Information and Computations*
- *Information Processing Letters*
- *International Journal of Computational Geometry and Applications*
- *International Journal of Foundations on Computer Science*
- *Journal of Algorithms*
- *Journal of Computer and System Sciences*
- *Journal of the ACM*
- *Networks*
- *Processings of the ACM Symposium on Theory of Computing*
- *Proceedings of the IEEE Symposium on Foundations of Computing Science*
- *SIAM Journal on Algebraic and Discrete Methods*
- *SIAM Journal on Computing*
- *Theoretical Computer Science*

第 2 章　算法复杂度与问题的下界

本章将讨论一些有关算法分析的基本问题，最主要的是设法弄清楚以下问题：

(1) 一些算法是高效的，而另一些却是低效的，那么怎样评价一个算法的好与坏呢？

(2) 一些问题容易解决，而另一些却不易解决，那么怎样评价一个问题的难易程度呢？

(3) 对于一个问题怎样知道哪个算法是最优的？也就是说，怎样知道对于同样问题不存在其他更好的算法呢？我们将要探讨这些彼此相关的问题。

2.1　算法的时间复杂度

如果执行一个算法花费较少的时间，需要较少的空间，那么通常认为该算法是好的。但是，依照传统，确定一个算法好坏程度最重要的因素是在执行时所需要的时间。在本书中，如果没有特别声明，我们关心的是时间准则。

为了度量一个算法的时间复杂度（time complexity），一种方法是为这个算法编写一个程序，看看程序执行得有多快。但这种方法不太合适，因为有许多与这个算法不相关的因素影响了程序的执行。例如，编程人员的能力、所使用的语言、操作系统，甚至特定语言的编译器都会影响运行程序所用的时间。

在算法分析中，总是选择出现在算法中某个特定的执行步，通过数学分析来确定完成该算法所需要的步数。例如，在排序算法中，数据项的比较是不可回避的，因此，经常用它来度量各种排序算法的时间复杂度。

当然，在一些排序算法中，数据的比较并不是一个决定性的因素，这种情况也是合理的。实际上，在某些排序算法中，可以很容易举出这样的实例：数据的移动才是最花费时间的。在此情况下，显然应该以数据的移动，而不是数据的比较来度量这种特定排序算法的时间复杂度。

通常认为运行算法的时间花费依赖于问题的规模（size of the problem）n。例如，在8.2节中定义的欧几里得旅行商问题（Euclidean traveling salesperson problem）中的点数就是问题的规模。正如所预想的，大多数算法随着问题规模n的增加，完成算法所需要的时间也随之增加。

假设执行一个算法花费$(n^3 + n)$步数，通常认为该算法的时间复杂度是n^3阶的。因为随着n的不断增加，项n^3比项n占优，项n与项n^3相比较变得不很重要了。现在，以非正式且常用的方式给出表述其正式且精确意义的定义。

定义　当且仅当存在两个正常数c和n_0，对于所有的$n \geqslant n_0$，使得$|f(n)| \leqslant c|g(n)|$，那么定义$f(n) = O(g(n))$。

从上面的定义，如果$f(n) = O(g(n))$，那么在某种意义上，$g(n)$随着n增大，$f(n)$以$g(n)$为界限。如果一个算法的时间复杂度是$O(g(n))$，这意味着对于某个常数c，当n足够大时，算法运行所需要的时间总是少于c倍的$|g(n)|$。

考虑这样一种情况，完成一个算法需要花费$(n^3 + n)$步数，那么

$$f(n) = n^3 + n = (1 + 1/n^2)n^3 \leqslant 2n^3 \quad 对于 n \geqslant 1$$

因此，可以说该算法的时间复杂度是$O(n^3)$，因为可以分别将c和n_0赋值为2和1。

接下来，将要弄清楚非常重要的一点，即对算法时间复杂度数量级阶的常见误解。

假设解决同一个问题的两个算法A_1和A_2的时间复杂度分别为$O(n^3)$和$O(n)$。如果请一个人为

算法A_1和A_2分别编写两个程序，并在同样的环境下运行这两个程序，算法A_2的程序一定比算法A_1的程序运行得快吗？通常，错误地认为算法A_2的程序将总是比算法A_1的程序运行得快。事实上，这并不一定正确。一个简单的原因是：在算法A_2中执行一步可能比在算法A_1中执行一步花费更多的时间。换句话说，尽管算法A_2比算法A_1所需要的步数少，但在某种意义上，算法A_1仍然会比算法A_2运行得快。假设算法A_1每一步所需的时间是算法A_2每一步所需时间的1/100，那么算法A_1和A_2的实际运算时间分别是n^3和$100n$。对于$n<10$，算法A_1比算法A_2运行得快，对于$n>10$，算法A_2比算法A_1运行得快。

现在读者可能理解了出现在函数$O(g(n))$定义中常量的重要性了，不能忽视这个常量，但是不管此常数有多大，随着n的增加，它的重要性在逐渐减小。如果算法A_1和A_2的时间复杂度分别为$O(g_1(n))$和$O(g_2(n))$，并且对于所有的n，$g_1(n)<g_2(n)$都成立，那么随着n变得足够大，算法A_1比算法A_2运行得快。

另外应当铭记的一点是我们总是至少在理论上能用硬件来实现算法。也就是说，总能设计一个电路来实现算法。如果两个算法都能由硬件实现，那么在一个算法中执行一步所需的时间与另一个算法中执行一步所需的时间相同。在此情况下，执行步数的量级显得更为重要了。如果算法A_1和A_2的时间复杂度分别是$O(n^3)$和$O(n)$，那么若两个算法都能完全由硬件实现，则算法A_2好于算法A_1。当然，只有我们很好掌握硬件实现算法的技术时，那么上面的讨论才是有意义的。

数量级的意义能够通过研究表2-1来领会。从表2-1中，能够注意到如下的事实：

表2-1 时间复杂度函数

时间复杂度函数	问题规模：n			
	10	10^2	10^3	10^4
$\log_2 n$	3.3	6.6	10	13.3
n	10	10^2	10^3	10^4
$n\log_2 n$	0.33×10^2	0.7×10^3	10^4	1.3×10^5
n^2	10^2	10^4	10^6	10^8
2^n	1 024	1.3×10^{30}	$>10^{100}$	$>10^{100}$
$n!$	3×10^6	$>10^{100}$	$>10^{100}$	$>10^{100}$

(1) 能够找出具有较低阶时间复杂度的算法将是非常有意义的。一个典型的例子是，对于一个具有n个数的列表进行顺序查找（sequential search），在最坏情况下需要$O(n)$次操作。如果对于一个n个数的有序列表，可以使用折半查找或二分搜索（binary search）。在最坏情况下，时间复杂度可以降到$O(\log n)$。当$n = 10^4$时，顺序查找可能需要10^4次操作，而折半查找只需要14次查找。

(2) 我们可能不喜欢像n^2，n^3等时间复杂度的函数，但是相对于2^n，还是可以接受的。例如，当$n = 10^4$时，$n^2 = 10^8$，但是，$2^n > 10^{100}$。数字10^{100}是非常大的，以至于不管运行得多快的计算机，算法都不能解决该问题。一个具有时间复杂度为$O(p(n))$的算法，其中$p(n)$是一个多项式函数，称为多项式算法（polynomial algorithm）。另一方面，不是以多项式函数为界的时间复杂度算法称为指数算法（exponential algorithm）。

在多项式算法与指数算法之间有很大的不同。遗憾的是很大一类指数算法似乎没有希望用多项式算法来替代。例如，每个解决欧几里得旅行商问题的算法到目前为止都是指数算法。类似地，解决像8.3节定义的可满足性问题（satisfiability problem）的每个算法到目前也是指数算法。但是，将会看到在3.1节定义的最小生成树问题（minimal spanning tree problem）是多项式算法可解决的。

在上面的讨论中，我们含糊了数据的概念。当然，对于某些数据，一个算法可能很快地运

行完，而对于其他的数据，算法可能运行得就完全不一样了。在下一节中将要讨论这些主题。

2.2 最好、平均和最坏情况的算法分析

对于任何算法，对其在三种情况下的行为感兴趣：最好情况、平均情况和最坏情况。通常，对最好情况下的分析是最容易的，最坏情况下的分析次之，而平均情况下的分析是最难的。实际上，一直都存在着涉及对平均情况分析的许多开放性问题。

例2-1 直接插入排序

一种最简单的排序方法是直接插入排序（straight insertion sort）。已知数字序列x_1，x_2，…，x_n，从左到右扫描这些数，如果x_i小于x_{i-1}，那么将x_i放到x_{i-1}的左边。换句话说，不断地向左移动x_i，直到它左边的数都小于或等于它。

算法2-1 直接插入排序

输入：x_1，x_2，…，x_n。
输出：x_1，x_2，…，x_n的有序序列。

```
For j := 2 to n do
Begin
  i := j-1
  x := x_j
  While x<x_i and i >0 do
  Begin
    x_{i+1} := x_i
    i := i-1
  End
  x_{i+1} := x
End
```

考虑输入序列：7，5，1，4，3，2，6。直接插入排序得到的有序序列过程如下：

7
5，7
1，5，7
1，4，5，7
1，3，4，5，7
1，2，3，4，5，7
1，2，3，4，5，6，7

在分析中，将利用数据移动$x := x_j$，$x_{i+1} := x_i$和$x_{i+1} := x$的次数作为算法时间复杂度的度量。在上面的算法中，有两层循环：外层循环和内层循环。对于外层循环，有两次数据移动操作总要执行，即$x := x_j$和$x_{i+1} := x$。由于内层循环可能执行，也可能不执行，用d_i表示在内层循环中对x_i执行的数据移动总次数。此时，显然直接插入排序的数据移动总次数为

$$X = \sum_{i=2}^{n}(2+d_i) = 2(n-1)+\sum_{i=2}^{n}d_i$$

最好情况：$\sum_{i=2}^{n}d_i = 0$，$X = 2(n-1) = O(n)$

当输入的数据已经有序时，将出现这种情况。

最坏情况：当输入的数据是逆序时，出现最坏的情况。在此情况下，

$d_2 = 1$
$d_3 = 2$
$\vdots$
$d_n = n-1$

因此，

$$\sum_{i=2}^{n} d_i = \frac{n}{2}(n-1)$$

$$X = 2(n-1)+\frac{n}{2}(n-1) = \frac{1}{2}(n-1)(n+4) = O(n^2)$$

平均情况：考虑元素x_i，$(i-1)$个数据元素已经排好了序。如果x_i是这i个元素中最大的，那么内层循环将不会执行，并且在整个此次内层循环中，没有任何数据的移动。如果x_i是这i个元素中第二大的，在内层循环中，将移动一个数据，以此类推。x_i是最大元素的概率是$1/i$，对于$1 \leqslant j \leqslant i$，这也是$x_i$是第$j$大元素的概率。因此，$(2 + d_i)$的平均是

$$\frac{2}{i}+\frac{3}{i}+\cdots+\frac{i+1}{i} = \sum_{j=1}^{i}\frac{(j+1)}{i} = \frac{i+3}{2}$$

直接插入排序的平均时间复杂度是

$$\sum_{i=2}^{n}\frac{i+3}{2} = \frac{1}{2}\left(\sum_{i=2}^{n} i+\sum_{i=2}^{n} 3\right) = \frac{1}{4}(n-1)(n+8) = O(n^2)$$

总之，在各种情况下，直接插入排序的时间复杂度分别如下：

最好情况：$2(n-1) = O(n)$

平均情况：$(n + 8)(n-1)/4 = O(n^2)$

最坏情况：$(n-1)(n + 4)/2 = O(n^2)$

例2-2　折半查找算法

折半查找（binary search）是一个有名的查找算法。在将一个数据集排序为递增或递减有序后，折半查找从序列的中间开始。如果查找的数据等于序列中间位置的数据，那么查找终止；否则，依据查找数据与中间位置数据比较的结果，可以递归地查找序列的左半部分或者右半部分。

算法2-2　折半查找

输入：有序数组a_1，a_2，…，a_n，$n>0$，X，其中$a_1 \leqslant a_2 \leqslant a_3 \leqslant \cdots \leqslant a_n$。

输出：如果$a_j = X$，那么输出j；如果不存在j使$a_j = X$，那么输出0。

```
i := 1    (*第一个数据*)
m := n    (*最后一个数据*)
While i≤m do
Begin
  j: = ⌊(i+m)/2⌋
  If X = a_j then output j and stop
  If X < a_j then m: = j-1
  else i := j + 1
End
j := 0
Output j
```

对折半查找最好情况下的分析是相当简单的。在最好的情况下，折半查找只需要一步就完成了。

最坏情况下的分析也是相当简单的。很容易看出来最多需要 $(\lfloor \log_2 n \rfloor + 1)$ 步数就能完成折半查找。贯穿于本书的剩余部分，除非有特别声明，$\log n$就是指$\log_2 n$。

为了简化讨论，假设$n = 2^k - 1$。

对于平均情况下的分析，如果有n个元素，那么对于该算法有一个元素仅需一步就能使算法成功地终止，该元素位于有序序列的第 $\left\lceil \frac{1+n}{2} \right\rceil$ 个位置。有两个元素只需要二步就能使算法成功地终止。一般来说，对于$t = 1, 2, \cdots, \lfloor \log n \rfloor + 1$，会有$2^{t-1}$个元素在$t$步之后使算法成功地终止。如果$X$不在序列中，那么算法在执行完 $\lfloor \log n \rfloor + 1$ 步后失败地终止。总的来说，可以说有$(2n + 1)$种不同的情况：有n种情况能使折半算法成功结束，另有$(n + 1)$种情况能使折半算法查找失败结束。

令$A(n)$表示在折半查找中平均的比较次数，并且$k = \lfloor \log n \rfloor + 1$，那么

$$A(n) = \frac{1}{2n+1}\left(\sum_{i=1}^{k} i2^{i-1} + k(n+1)\right)$$

现在要证明

$$\sum_{i=1}^{k} i2^{i-1} = 2^k(k-1) + 1 \tag{2-1}$$

上面的公式可以通过对k进行归纳来证明。当$k = 1$时，等式(2-1)显然是成立的。假设当$k = m$，$m>1$时，等式(2-1)是成立的。然后，证明在$k = m + 1$时，等式(2-1)也是成立的。即假设等式(2-1)是成立的，证明

$$\sum_{i=1}^{m+1} i2^{i-1} = 2^{m+1}(m+1-1) + 1 = 2^{m+1} \cdot m + 1$$

需要注意

$$\sum_{i=1}^{m+1} i2^{i-1} = \sum_{i=1}^{m} i2^{i-1} + (m+1)2^{m+1-1}$$

将等式(2-1)代入上面的公式，得到

$$\sum_{i=1}^{m+1} i2^{i-1} = 2^m(m-1) + 1 + (m+1)2^m = 2^m \cdot 2m + 1 = 2^{m+1} \cdot m + 1$$

至此，我们已经证明了等式(2-1)的有效性，使用等式(2-1)得到

$$A(n) = \frac{1}{2n+1}((k-1)2^k + 1 + k2^k)$$

当n非常大时，可以得到

$$A(n) \approx \frac{1}{2^{k+1}}(2^k(k-1) + k2^k) = \frac{(k-1)}{2} + \frac{k}{2} = k - \frac{1}{2}$$

所以，$A(n) < k = O(\log n)$。

现在，读者可能想知道通过假设$n = 2^k - 1$得到的结果对于一般的n是否有效。假如$t(n)$是一个非递减函数，并且$t(n) = O(f(n))$是通过假设$n = 2^k - 1$时得到的算法的时间复杂度，同时对于

某个$b \geqslant 1$且c'是一个常数，$f(bn) \leqslant c'f(n)$成立（这意味着f是一个平滑函数，而每个多项式函数都是这样的函数）。那么

$t(2^k) \leqslant cf(2^k)$，其中c是一个常数。

对于$0 \leqslant x \leqslant 1$，令$n' = 2^{k+x}$

$$t(n') = t(2^{k+x}) \leqslant t(2^{k+1}) \leqslant cf(2^{k+1}) = cf(2^{k+x} \cdot 2^{1-x}) \leqslant cc'f(2^{k+x}) = c''f(n')$$

因此，$t(n') = O(f(n'))$。

上面的讨论表明可以假设$n = 2^k - 1$得到算法的时间复杂度。在本书后面的章节中，必要时将假设$n = 2^k - 1$，而不再解释为什么可以这样了。

总之，对于折半查找，可以得到

最好情况：$O(1)$

平均情况：$O(\log n)$

最坏情况：$O(\log n)$

例2-3　直接选择排序

直接选择排序（straight selection sort）也许是最简单的一类排序，然而对这种算法的分析却相当有趣。直接选择排序可以很容易地描述如下：

(1) 找出最小的数，把这个最小的数与a_1交换，放在a_1的位置。

(2) 对余下的数，重复上面的步骤。也就是，找到第二小的数，放在a_2交换。

(3) 继续这个过程，直至找到最大的数。

算法2-3　直接选择排序

输入：a_1，a_2，…，a_n。

输出：a_1，a_2，…，a_n的有序序列。

```
For j: = 1 to n−1 do
Begin
  f:= j
  For k := j + 1 to n do
    If a_k<a_f then f := k
  a_j ↔ a_f
End
```

在上面的算法中，为了找出a_1，a_2，…，a_n中的最小数，必需设置一个标记f，并且初始化$f = 1$，然后比较a_f与a_2。如果$a_f < a_2$，那么不需要做任何事；否则，置$f = 2$，再比较a_f与a_3，依此类推。

很明显，在一个直接选择排序中有两种操作：两个元素的比较与标记的改变。两个元素进行比较的次数是一个固定数，即$n(n-1)/2$。也就是，不论输入什么数据，总要进行$n(n-1)/2$次比较。因此，选择标记的改变次数来度量直接选择排序的时间复杂度。

标记的改变依赖于数据。$n = 2$时，只有两个排列：

(1，2)

和　(2，1)。

对于第一个排列，标记不需要改变，而对第二个排列，标记必须改变。

令$f(a_1, a_2, \cdots, a_n)$表示在序列a_1，a_2，…，a_n中找出最小数而标记需要改变的次数。下表说明了$n = 3$的情况。

a_1,	a_2,	a_3	$f(a_1, a_2, a_3)$
1,	2,	3	0
1,	3,	2	0
2,	1,	3	1
2,	3,	1	1
3,	1,	2	1
3,	2,	1	2

为了确定$f(a_1, a_2, \cdots, a_n)$，做如下注释：

(1) 如果$a_n = 1$，那么$f(a_1, a_2, \cdots, a_n) = 1 + f(a_1, a_2, \cdots, a_{n-1})$，因为在最后一步标记必定改变。

(2) 如果$a_n \neq 1$，那么$f(a_1, a_2, \cdots, a_n) = f(a_1, a_2, \cdots, a_{n-1})$，因为在最后一步标记不必改变。

令$P_n(k)$表示$\{1, 2, \cdots, n\}$的一个排列$a_1, a_2, \cdots, a_n$，找到最小数需要改变k次标记。例如，$P_3(0) = 2/6$，$P_3(1) = 3/6$以及$P_3(2) = 1/6$，那么，为找到最小的数而需要改变标记的平均次数为

$$X_n = \sum_{k=0}^{n-1} kP_n(k)$$

那么，直接选择排序标记改变的平均次数为

$$A(n) = X_n + A(n-1)$$

为了确定X_n，下面用到之前讨论过的等式：

$$\begin{aligned} f(a_1, a_2, \cdots, a_n) &= 1 + f(a_1, a_2, \cdots, a_{n-1}) \quad \text{如果} a_n = 1 \\ &= f(a_1, a_2, \cdots, a_{n-1}) \quad \text{如果} a_n \neq 1 \end{aligned}$$

依据上面的公式，可以得到

$$P_n(k) = P(a_n = 1)P_{n-1}(k-1) + P(a_n \neq 1)P_{n-1}(k)$$

而$P(a_n = 1) = 1/n$，$P(a_n \neq 1) = (n-1)/n$。因此，可以得到

$$P_n(k) = \frac{1}{n}P_{n-1}(k-1) + \frac{n-1}{n}P_{n-1}(k) \tag{2-2}$$

此外，得到下面具有初始条件的公式：

$$\begin{aligned} P_1(k) &= 1, \text{若} k = 0, \\ &= 0, \text{若} k \neq 0, \\ P_n(k) &= 0, \text{若} k > 0, \text{并且若} k = n \end{aligned} \tag{2-3}$$

为了使读者对公式有更进一步的认识，现在注释如下：

$$P_2(0) = \frac{1}{2}$$

及$P_2(1) = \dfrac{1}{2}$

$$P_3(0) = \frac{1}{3}P_2(-1) + \frac{2}{3}P_2(0) = \frac{1}{3} \times 0 + \frac{2}{3} \times \frac{1}{2} = \frac{1}{3}$$

及$P_3(2) = \dfrac{1}{3}P_2(1) + \dfrac{2}{3}P_2(2) = \dfrac{1}{3} \times \dfrac{1}{2} + \dfrac{2}{3} \times 0 = \dfrac{1}{6}$。

接下来，要证明：

$$X_n = \sum_{k=1}^{n-1} kP_n(k) = \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} = H_n - 1 \tag{2-4}$$

其中H_n是第n个调和数（Harmonic number）。

采用数学归纳法证明等式(2-4)，也就是要证明

$$X_n = \frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} = H_n - 1$$

对于$n = 2$很明显是成立的。假设对于$n = m$，$m>2$时等式(2-4)是成立的。现在要证明等式(2-4)对于$n = m + 1$也是成立的。因此，要证明

$$X_{m+1} = H_{m+1} - 1$$

$$X_{m+1} = \sum_{k=1}^{m} kP_{m+1}(k) = P_{m+1}(1) + 2P_{m+1}(2) + \cdots + mP_{m+1}(m)$$

利用等式(2-2)，可以得到

$$\begin{aligned}
X_{m+1} &= \frac{1}{m+1}P_m(0) + \frac{m}{m+1}P_m(1) + \frac{2}{m+1}P_m(1) + \frac{2m}{m+1}P_m(2) + \cdots + \frac{m}{m+1}P_m(m-1) + \frac{m^2}{m+1}P_m(m) \\
&= \frac{1}{m+1}P_m(0) + \frac{m}{m+1}P_m(1) + \frac{1}{m+1}P_m(1) + \frac{1}{m+1}P_m(1) + \frac{2m}{m+1}P_m(2) + \frac{1}{m+1}P_m(2) \\
&\quad + \frac{2}{m+1}P_m(2) + \frac{3m}{m+1}P_m(3) + \cdots + \frac{1}{m+1}P_m(m-1) + \frac{m-1}{m+1}P_m(m-1) \\
&= \frac{1}{m+1}(P_m(0) + P_m(1) + \cdots + P_m(m-1)) + \frac{m+1}{m+1}P_m(1) \\
&\quad + \frac{2m+1}{m+1}P_m(2) + \cdots + \frac{(m-1)(m+1)}{m+1}P_m(m-1) \\
&= \frac{1}{m+1} + (P_m(1) + 2P_m(2) + \cdots + (m-1)P_m(m-1)) \\
&= \frac{1}{m+1} + H_m - 1 \qquad \text{（由递归假设）} \\
&= H_{m+1} - 1
\end{aligned}$$

既然直接选择排序的平均时间复杂度是：

$$A(n) = X_n + A(n-1)$$

可以得到

$$\begin{aligned}
A(n) &= H_n - 1 + A(n-1) = (H_n - 1) + (H_{n-1} - 1) + \cdots + (H_2 - 1) \\
&= \sum_{i=2}^{n} H_i - (n-1)
\end{aligned} \tag{2-5}$$

$$\begin{aligned}
\sum_{i=1}^{n} H_i &= H_n + H_{n-1} + \cdots + H_1 \\
&= H_n + \left(H_n - \frac{1}{n}\right) + \cdots + \left(H_n - \frac{1}{n} + \frac{1}{n-1} - \cdots - \frac{1}{2}\right) \\
&= nH_n - \left(\frac{n-1}{n} + \frac{n-2}{n-1} + \cdots + \frac{1}{2}\right)
\end{aligned}$$

$$= nH_n - \left(1 - \frac{1}{n} + 1 - \frac{1}{n-1} + \cdots + 1 - \frac{1}{2}\right)$$

$$= nH_n - \left(n - 1 - \frac{1}{n} - \frac{1}{n-1} - \cdots - \frac{1}{2}\right)$$

$$= nH_n - n + H_n$$

$$= (n+1)H_n - n$$

因此，

$$\sum_{i=2}^{n} H_i = (n+1)H_n - H_1 - n \tag{2-6}$$

将等式(2-6)代入等式(2-5)中，可以得到

$$A(n) = (n+1)H_n - H_1 - (n-1) - n = (n+1)H_n - 2n$$

当n足够大时，

$$A(n) \approx n \log_e n = O(n \log n)$$

直接选择排序的时间复杂度总结如下：

最好情况：$O(1)$

平均情况：$O(n \log n)$

最坏情况：$O(n^2)$

例2-4　快速排序

快速排序（quick sort）基于分治策略（divide-and-conquer strategy），后面将详细地说明分治策略。在此，简单地说分治策略是将一个问题划分成两个子问题，再分别各自独立地解决这两个子问题，随后，将结果合并成最终的解。将分治策略应用于排序，得到一个称为快速排序的排序方法。已知数的集合a_1，a_2，…，a_n，选择一个元素X将a_1，a_2，…，a_n划分为两个序列，如图2-1所示。

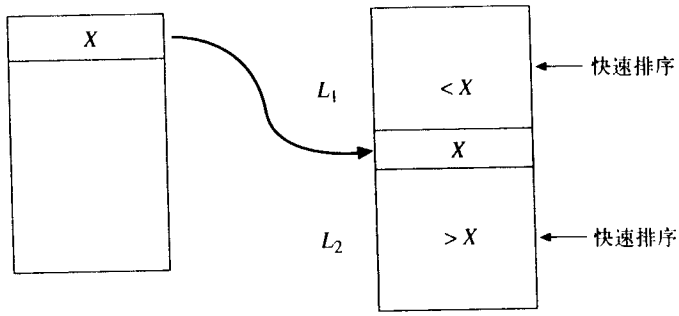


图2-1　快速排序

在划分之后，分别对L_1和L_2递归地应用快速排序，因为L_1中所有的a_i都小于或等于X，而L_2中所有的a_i都大于X，所以结果序列是一个有序序列。

这里的问题是：如何划分序列？当然不必用X来扫描整个序列，以确定a_i是小于还是等于X，这会引起大量的数据移动。正如在算法中说明的，可使用两个指针，向中间移动它们，在必要时，进行数据交换。

算法2-4 Quicksort(f ,l)

输入：a_f，a_{f+1}，…，a_l。
输出：a_f，a_{f+1}，…，a_l的有序序列。

```
If f ≥ l then Return
X := a_f
i := f
j := l
While i<j do
Begin
    While a_j ≥ X do
        j := j-1
    a_i ↔ a_j
    While a_i ≤ X do
        i := i + 1
    a_i ↔ a_j
End
Quicksort(f , j-1)
Quicksort(j + 1 , l)
```

下面的例子说明了快速排序的主要思想：

$X=3$	a_1	a_2	a_3	a_4	a_5	a_6
$a_j=a_6<X$	3	6	1	4	5	2
	↑i					↑j
	2	6	1	4	5	3
	↑i					↑j
$a_i=a_2>X$	2	6	1	4	5	3
		↑i				↑j
	2	3	1	4	5	6
		↑i				↑j
	2	3	1	4	5	6
		↑i			↑j	
	2	3	1	4	5	6
		↑i		↑j		
$a_j=a_3<X$	2	3	1	4	5	6
		↑i	↑j			
	2	1	3	4	5	6
		↑i	↑j			
	2	1	3	4	5	6
			i↑↑j			
	⊢← ≤3 →⊣		=3		⊢← ≥3 →⊣	

当X正好在中间位置将序列划分成两个子序列，此时快速排序呈现出最好的情况。换句话说，X划分出两个包含相同数目元素的子序列。在这种情况下，第一趟循环需要n步将序列划分成两个子序列。在下一趟循环，对每一个子序列都再需要$n/2$步（忽略用来划分的元素）。因此，对第二趟，又需要$2\cdot(n/2)=n$步。假设$n=2^p$，那么总共需要pn步。而$p=\log_2 n$。因此，在最

好情况下，快速排序的时间复杂度是$O(n \log n)$。

当输入的数据是正序或逆序时，快速排序将出现最坏情况。在这种情况下，选择的只是极值元素（最大或最小元素）。因此，快速排序在最坏情况下所需要的总步数是

$$n+(n-1)+\cdots+1=\frac{n}{2}(n+1)=O(n^2)$$

为了分析平均情况，令$T(n)$表示在平均情况下对n个元素执行快速排序所需要的步数。假如在划分操作之后，序列划分成两个子序列，第一个子序列包含s个元素，第二个子序列包含$(n-s)$个元素，s的取值范围从1到n。为了分析平均情况下的性能，需要考虑所有可能的情况。下面的公式用于得到$T(n)$：

$$T(n)=\operatorname*{Ave}_{1\le s\le n}(T(s)+T(n-s))+cn$$

其中cn表示在第一次划分中所需要的操作次数（注意在划分成两个子序列前，需要扫描每个元素。）

$$\begin{aligned}&\operatorname*{Ave}_{1\le s\le n}(T(s)+T(n-s))\\&=\frac{1}{n}\sum_{s=1}^{n}(T(s)+T(n-s))\\&=\frac{1}{n}(T(1)+T(n-1)+T(2)+T(n-2)+\cdots+T(n)+T(0))\end{aligned}$$

因为$T(0)=0$，

$$T(n)=\frac{1}{n}(2T(1)+2T(2)+\cdots+2T(n-1)+T(n))+cn$$

或者，$(n-1)T(n)=2T(1)+2T(2)+\cdots+2T(n-1)+cn^2$。

用$n=n-1$带入上面的公式，得到

$$(n-2)T(n-1)=2T(1)+2T(2)+\cdots+2T(n-2)+c(n-1)^2$$

所以，

$$\begin{aligned}(n-1)T(n)-(n-2)T(n-1)&=2T(n-1)+c(2n-1)\\(n-1)T(n)-nT(n-1)&=c(2n-1)\\\frac{T(n)}{n}&=\frac{T(n-1)}{n-1}+c\left(\frac{1}{n}+\frac{1}{n-1}\right)\end{aligned}$$

递归地，

$$\begin{aligned}\frac{T(n-1)}{n-1}&=\frac{T(n-2)}{n-2}+c\left(\frac{1}{n-1}+\frac{1}{n-2}\right)\\&\vdots\\\frac{T(2)}{2}&=\frac{T(1)}{1}+c\left(\frac{1}{2}+\frac{1}{1}\right)\end{aligned}$$

可以得到

$$\begin{aligned}\frac{T(n)}{n}&=c\left(\frac{1}{n}+\frac{1}{n-1}+\cdots+\frac{1}{2}\right)+c\left(\frac{1}{n-1}+\frac{1}{n-2}+\cdots+1\right)\\&=c(H_n-1)+cH_{n-1}\end{aligned}$$

$$= c(H_n + H_{n-1} - 1)$$
$$= c\left(2H_n - \frac{1}{n} - 1\right)$$
$$= c\left(2H_n - \frac{n+1}{n}\right)$$

最终，可以得到

$$\begin{aligned} T(n) &= 2cnH_n - c(n+1) \\ &\cong 2cn\log_c n - c(n+1) \\ &= O(n\log n) \end{aligned}$$

总之，快速排序的时间复杂度分别是

最好情况：$O(n\log n)$

平均情况：$O(n\log n)$

最坏情况：$O(n^2)$

例2-5 确定秩问题

设有一组数a_1，a_2，…，a_n的集合。如果$a_i > a_j$，那么认为a_i支配（dominate）a_j。如果要找出a_i支配a_j的总次数，可以简单地将这些数排成一个序列，该问题可以很快地解决。

现在将问题扩展至2维的情况，也就是，每个数据是平面上的一个点。在这种情况下，首先定义在2维空间中点的支配含义。

定义 已知两个点$A = (a_1, a_2)$和$B = (b_1, b_2)$，对于$i = 1, 2$，当且仅当$a_i > b_i$，那么A支配B。如果A不支配B，B也不支配A，那么A和B是不可比的。

考虑图2-2。

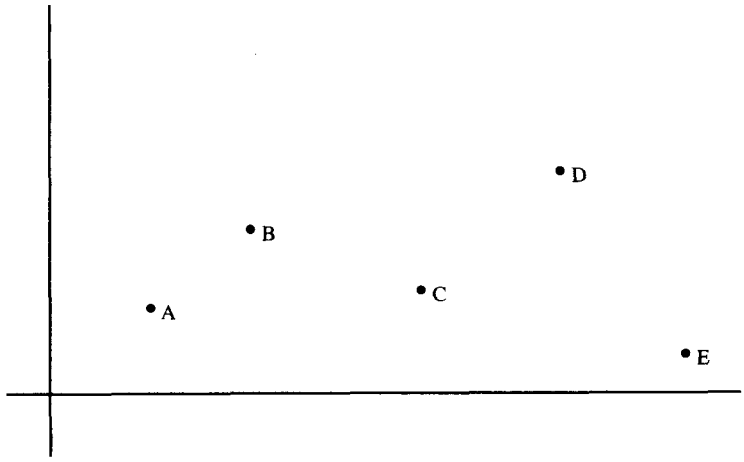


图2-2 说明支配关系的示例

对于图2-2中的点，可以得到如下的关系：

(1) B、C和D支配A。

(2) D支配A、B和C。

(3) 所有其他的点对都是不可比的。

在定义了支配关系后，可进一步可定义点秩（rank of a point）的概念。

定义 已知n个点的集合S，点X的秩定义为由X支配的点个数。

对于图2-2上的点，点A和E的秩是0，因为它们不支配任何点。点B和C的秩是1，因为只有点A被它们所支配；点D的秩是3，因为点A、B和C都被点D所支配。

我们的问题是找出每个点的秩。

解决该问题的直接方法是对全部的点对进行穷举的比较。这种方法的时间复杂度是$O(n^2)$。下面将证明在最坏情况下可用$O(n \log n)$时间解决该问题。

考虑图2-3，第一步找出一条垂直于x轴的直线L，该直线将点集划分成点数相等的两个子集。令A表示在直线L左边的子集，B表示在直线L右边的子集。显然，A中任何点的秩不受B存在的影响，但是B中点的秩可能受A存在的影响。

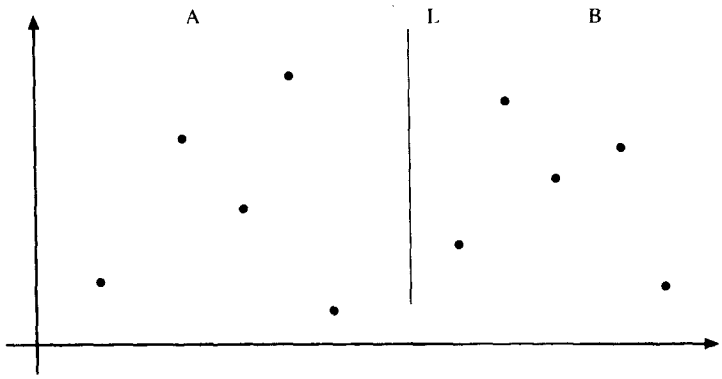


图2-3 解决秩问题的第一步

假设分别找出了A和B中点的局部秩（local rank），也就是，找出了A中点的秩（不考虑B）和B中点的秩（不考虑A）。现在给出在图2-3和图2-4中点的局部秩。

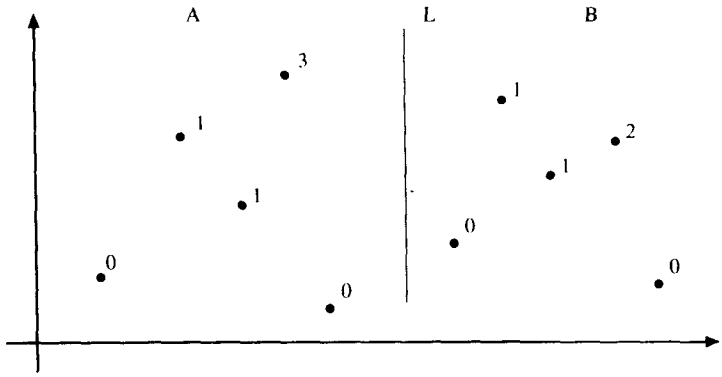


图2-4 在A和B中点的局部秩

现在将全部的点投影在直线L上。注意，B中的点P_1支配A中的点P_2，当且仅当点P_1的y值比点P_2的y值高。令P是B中的一点，点P的秩是B中点P的秩加上A中点的y值小于P的y值的点数。点秩的更新结果如图2-5所示。

接下来的算法确定了平面上每个点的秩。

算法2-5 确定秩的算法

输入：平面点P_1，P_2，…，P_n的集合S。

输出：集合S中每个点的秩。

步骤1 如果S只有一个点，那么返回该点的秩为0；否则，选择垂直于X轴的直线L，使得S中$n/2$个点的x值比L小（称这个点集为A），其余点的x值比L大（称这个点集为B），注意L是这组点的x值的中值。

步骤2 递归地使用这个确定点秩的算法找出集合A中点的秩和集合B中点的秩。

步骤3 依据y值对集合A和B中的点进行排序，顺序扫描这些点，对B中的每一个点确定A中点的y值小于该点的数量，那么该点的秩就是B中这个点秩（步骤2找出的），加上A中点的y值小于其y值的点数量。

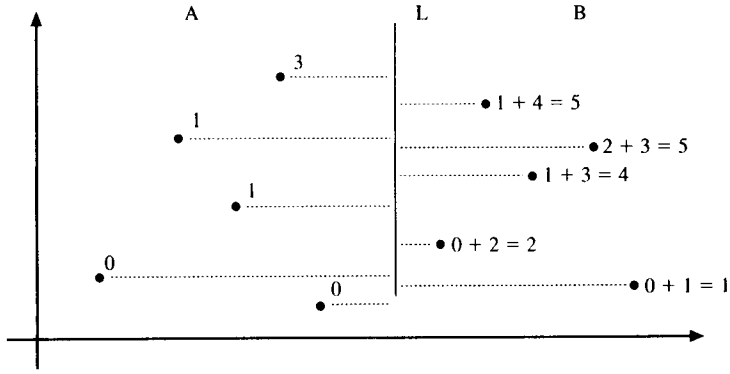


图2-5 秩的更新

算法2-5是一个递归算法。它基于分治策略，将一个问题分成两个子问题，分别独立地解决这两个子问题，然后将两个子问题的解合并成该问题的最终解。这个算法的时间复杂度依赖于下列步骤的时间复杂度。

(1) 在步骤1中，有一个找出一组数的中值操作，在第7章中将证明任何中值查找算法的最小时间复杂度为$O(n)$。

(2) 在步骤3中，有一个排序操作，在2.3节，将证明任何排序算法的最小时间复杂度为$O(n \log n)$，扫描花费$O(n)$步。

令$T(n)$表示完成秩的查找算法的总时间。那么

$$T(n) \leqslant 2T(n/2) + c_1 n \log n + c_2 n \leqslant 2T(n/2) + cn \log n$$

其中c_1、c_2和c为常数。令$n = 2^p$，那么

$$\begin{aligned}
T(n) &\leqslant 2T(n/2) + cn \log n \\
&\leqslant 2(2T(n/4) + cn \log(n/2)/2) + cn \log n \\
&= 4T(n/4) + c(n \log (n/2) + n \log n) \\
&\quad \vdots \\
&\leqslant nT(1) + c(n \log n + n \log (n/2) + n \log (n/4) + \cdots + n \log 2) \\
&= nT(1) + cn\left(\frac{1+\log n}{2}\log n\right) \\
&= c_3 n + \frac{c}{2} n \log^2 n + \frac{c}{2} n \log n
\end{aligned}$$

因此，$T(n) = O(n \log^2 n)$。

上面的时间复杂度既是对最坏情况也是对平均情况的分析，因为对于排序的最小时间复杂度$O(n \log n)$对于平均情况也是有效的。类似地，查找中值的时间复杂度$O(n)$对于平均情况也是有效的。

注意，这个查找每个点秩的算法比使用穷举测试算法好得多。如果要实施穷举全部点对的扫描，需要$O(n^2)$步来完成整个过程。

2.3 问题的下界

在前一节，许多确定算法时间复杂度的例子让我们知道怎样判断一个算法好。本节将从完全不同的角度考虑复杂度问题。

考虑确定点秩的问题，或者旅行商问题，现在要问：如何度量一个问题的困难度？

上面的问题可以直观地回答。如果问题可由一个具有较低时间复杂度的算法解决，那么该问题是简单的问题；否则，该问题是困难的问题。这个直观的定义导出了问题下界（lower bound of a problem）的概念。

定义 一个问题的下界是用来解决该问题的任意算法所需要的最小时间复杂度。

为了描述下界，使用记号Ω，它的定义如下：

定义 当且仅当存在正的常数c和n_0，对于所有的$n>n_0$，都有$|f(n)|\geqslant c|g(n)|$成立，那么

$$f(n)=\Omega(g(n))$$

尽管可以使用平均情况时间复杂度（average-case time complexity），但上面的定义通常是指最坏情况时间复杂度（worst-case time complexity）。如果使用平均情况时间复杂度，那么下界称为平均情况下界（average-case lower bound）；否则，称为最坏情况下界（worst -case lower bound）。在本书中，除非有其他的声明，提到下界都是指最坏情况下界。

从上面的定义中，似乎必须要枚举所有可能的算法，来确定每个算法的时间复杂度，才能找出最小的时间复杂度。这当然是不可能的，因为不可能简单地枚举出所有可能的算法，而且也不能肯定将来是否能发明出一种新的算法，它得到更好的界。

读者应该注意到，定义的下界不是唯一的。一个熟知的下界例子是排序的下界，它是$\Omega(n \log n)$。想象在这个下界找出之前，也许证明了排序的下界是$\Omega(n)$，因为在完成排序之前，每一个数据元素都要检验。事实上，甚至可以更极端。例如，在$\Omega(n)$被看作是排序的下界之前，有人可能将$\Omega(1)$作为一个下界，因为算法最少只需要一步就可以完成任何排序算法。

从上面的讨论中知道，对于排序可能存在三个下界：$\Omega(n \log n)$，$\Omega(n)$和$\Omega(1)$，它们都是正确的。但是，很明显$\Omega(n \log n)$是唯一有意义的，另外两个是不重要的下界。既然下界太低，就是不重要的，那么就希望下界尽可能得高。下界的研究总是以科学家建议有较低下界开始的，然后，通过找到更高的下界来提高下界。问题的更高下界代替原来的下界成为该问题当前有意义的下界。这种情况一直持续到找出另一个更高的下界。

我们必须明白每个更高下界的找出都是通过理论分析得到的，而不是纯粹的猜测。由于找出的下界不断地提高，不可避免地想知道是否存在下界的上限？例如，考虑排序的下界，是否有可能将$\Omega(n^2)$看作是排序的下界呢？不能，因为存在一个称为堆排序的算法，它的最坏情况时间复杂度是$\Omega(n \log n)$。因此，知道排序的下界至多是$\Omega(n \log n)$。

现在考虑下面的两种情况：

情况1：目前，一个问题被找出的最高下界是$\Omega(n \log n)$，而解决这个问题的最可行算法的时间复杂度为$\Omega(n^2)$。

在这种情况下，有三种可能性：

(1) 问题的下界太低，因此，我们尽可能地寻找一个更有力的或更高的下界。换句话说，应该尽可能地提高下界。

(2) 最可行的算法不够好。因此，尽可能地找出一个更好的具有较低时间复杂度的算法。换句话说，应该向下改进最好的时间复杂度。

(3) 下界可能被提高，算法也可能被改善，因此，应该尽力提高这两个方面。

情况2：当前的下界为$\Omega(n \log n)$，确实存在一个时间复杂度为$\Omega(n \log n)$的算法。

在这种情况下，我们明白下界与算法都不能进一步提高了。换句话说，已经找到了一个解决该问题的最优算法和真正有意义的下界。

需要强调上面的这一点。在本章的开始曾提出了一个问题：如何知道一个算法是最优的？现在有了答案：如果一个算法的时间复杂度等于这个问题的下界，那么该算法是最优的。因此下界和算法都不能再进一步提高了。

下面是有关下界的一些重要概念的总结：

(1) 问题的下界是所有解决该问题算法的最小时间复杂度。

(2) 下界越高越好。

(3) 如果当前所知问题的下界比解决这个问题的最可行算法的时间复杂度低，那么下界可以通过改善算法来提高，算法也可以通过降低时间复杂度来改善，或两方面同时进行改善。

(4) 如果当前所知道的下界等于可行算法的时间复杂度，那么算法和下界都不能再改善了。算法是一个最优算法，下界也是最高下界，是真正有意义的下界。

在下面几节中，我们将讨论一些找下界的方法。

2.4 排序的最坏情况下界

对于许多算法，可以把一个算法的执行过程描述成一棵二叉树。例如，考虑直接插入排序，假如输入3个互不相同的元素。在这样的情况下，有6种不同的排列：

a_1	a_2	a_3
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

当对上面的数据集进行直接插入排序时，每个排列都引起不同的反应。例如，输入是（2，3，1），直接插入排序执行如下：

(1) 比较$a_1 = 2$和$a_2 = 3$，由于$a_2>a_1$，因此，没有数据元素的交换。

(2) 比较$a_2 = 3$和$a_3 = 1$，由于$a_2>a_3$，因此，交换a_2和a_3，也就是$a_2 = 1$，$a_3 = 3$。

(3) 比较$a_1 = 2$和$a_2 = 1$，由于$a_1>a_2$，因此，交换a_1和a_2，也就是$a_1 = 1$，$a_2 = 2$。

如果输入数据是（2，1，3），算法的执行如下：

(1) 比较$a_1 = 2$和$a_2 = 1$，由于$a_1>a_2$，因此，交换a_1和a_2，即$a_1 = 1$和$a_2 = 2$。

(2) 比较$a_2 = 2$和$a_3 = 3$，由于$a_2<a_3$，因此，不发生数据交换。

图2-6表示在对3个数据元素排序时，直接插入排序是如何用二叉树来描述的。这棵二叉树也很容易修改，来处理4个数据元素的情况。很容易看到当应用到任何数目的数据元素时，直接插入排序可以用二叉决策树（binary decision tree）来描述。

通常，基本操作是比较和交换的排序算法可以用二叉决策树来描述。图2-7显示了如何使用二叉决策树来描述冒泡排序。在图2-7中，假设有3个互不相同的数据元素。图2-7也是非常简化的，以致不必占用大量的空间。因为对冒泡排序（bubble sort）比较熟悉，所以在这里就不对它分析了。

基于对特定输入数据集进行比较和交换操作的排序算法的执行，对应于从树的根结点到某个叶子结点的一条路径。因此，每个叶子结点对应于一种特定的排列。从树的根到叶子结点的最长路径称为树的深度（depth），对应着该算法的最坏情况时间复杂度。为了找到排序问题的

下界，必须在模型化排序算法的所有可能的二叉决策树中，找到某棵树的最小深度。

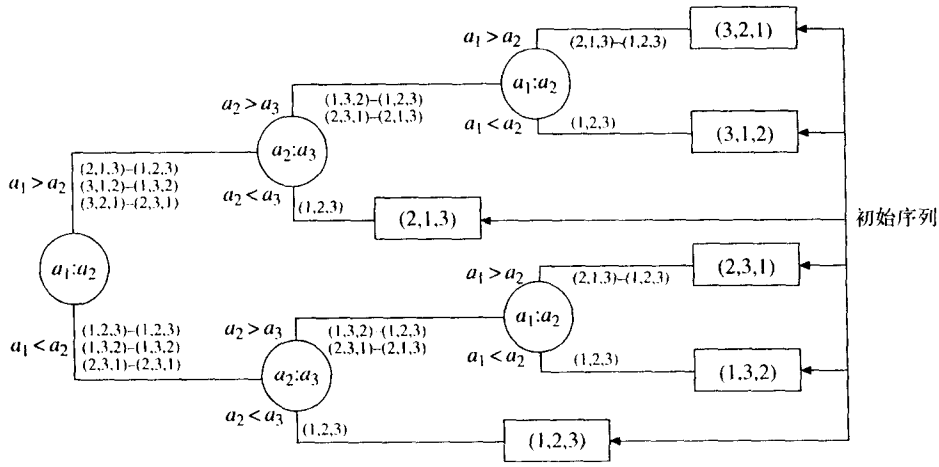


图2-6 用二叉树表示的对3个元素的直接插入排序

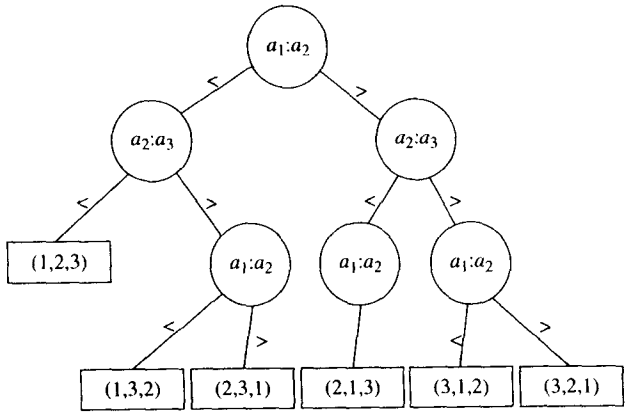


图2-7 描述冒泡排序的二叉决策树

注意下面的要点：

(1) 对于每个排序算法，因为有$n!$种不同的排列，所以相应的二叉决策树有$n!$片树叶。

(2) 如果树是平衡的（balanced），那么拥有固定数目叶子结点的二叉树深度是最小的。

(3) 当二叉树是平衡的，那么树的深度是$\lceil \log X \rceil$，其中X是叶结点的个数。

基于上面的推理，能很容易地推导出排序问题的下界是$\lceil \log n! \rceil$，也就是说，在最坏情况下，排序所需要的比较次数至少是$\lceil \log n! \rceil$。

$\log n!$对我们大多数来说可能有点陌生。我们的确不知道这个数字有多大，现在将讨论两个近似$\log n!$的方法。

方法1

基于下面的事实

$$\begin{aligned}\log n! &= \log(n(n-1)\cdots 1)\\ &= \sum_{i=1}^{n} \log i\\ &= (2-1)\log 2 + (3-2)\log 3 + \cdots + (n-n+1)\log n\end{aligned}$$

$$
\begin{aligned}
&> \int_1^n \log x \mathrm{d}x \\
&= \log \mathrm{e} \int_1^n \log_{\mathrm{e}} x \mathrm{d}x \\
&= \log \mathrm{e}[x \log_{\mathrm{e}} x - x]_1^n \\
&= \log \mathrm{e}(n \log_{\mathrm{e}} n - n + 1) \\
&= n \log n - n \log \mathrm{e} + 1.44 \\
&\geqslant n \log n - 1.44n \\
&= n \log n\left(1 - \frac{1.44}{\log n}\right)
\end{aligned}
$$

如果令$n = 2^2$，那么 $n \log n\left(1 - \frac{1.44}{2}\right) = 0.28n \log n$。

因此，根据假设$n_0 = 2^2$，及$c = 0.28$，可以得到

$$\log n! \geqslant cn \log n \text{对于} n \geqslant n_0$$

也就是说，排序的最坏情况下界是$\Omega(n \log n)$。

方法2 斯特林近似值

当n非常大时，斯特林（Stiring）近似值用下面的公式来近似$n!$的值

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n$$

这个公式几乎能在任何一本高级微积分书上找到。表2-2表明了斯特林近似值与$n!$有多接近。在表中，用S_n表示斯特林近似值。

表2-2 斯特林近似值的一些值

n	$n!$	S_n
1	1	0.922
2	2	1.919
3	6	5.825
4	24	23.447
5	120	118.02
6	720	707.39
10	3 628 800	3 598 600
20	2.433×10^{18}	2.423×10^{18}
100	9.333×10^{157}	9.328×10^{157}

使用斯特林近似值，将得到

$$
\begin{aligned}
\log n! &= \log \sqrt{2\pi} + \frac{1}{2}\log n + n \log \frac{n}{\mathrm{e}} \\
&= \log \sqrt{2\pi} + \frac{1}{2}\log n + n \log n - n \log \mathrm{e} \\
&\geqslant n \log n - 1.44n
\end{aligned}
$$

从上面的两种方法中，可以断定在最坏情况下，排序所需要的最少比较次数是$\Omega(n \log n)$。

需要提醒的是，上面的陈述并不意味着不能找到更高的下界。换句话说，一些新发现可能让我们知道排序的下界实际上可能更高。例如，很可能有人可找到排序的下界是$\Omega(n^2)$。

在接下来的一节中，将说明排序算法的最坏情况时间复杂度等于刚得出的下界，因为这样的算法存在，该下界就不能进一步提高了。

2.5 堆排序：在最坏情况下最优的排序算法

堆排序（heap sort）是一种时间复杂度为$O(n \log n)$的排序算法。在介绍堆排序之前，先研究一下直接选择排序，来理解为什么在最坏情况下它不是最优的。对于直接选择排序，需要$(n-1)$步得到最小的数，然后需要$(n-2)$步得到第二小的数，依此类推（均是在最坏的情况下）。因此，在最坏情况下，直接选择排序需要$O(n^2)$步。如果更仔细地观察直接选择排序，当尽力找第二小的数时，在寻找第一小的数时所推理出的信息根本没有用到。这就是直接选择排序执行效率如此差的原因。

考虑另一种排序算法，称为淘汰排序（knockout sort），它比直接选择排序好得多。淘汰排序与直接选择排序比较相似，它也是先找到最小的数，再找第二小的数，依此类推。但是在找到第一小的数后，它保留一些信息，对查找第二小的数非常有效。

考虑输入数列2，4，7，3，1，6，5，8。可以构建淘汰树（knockout tree）找最小的数，如图2-8所示。

在找到最小数之后，通过用∞代替1来找第二小的数。只需要考查淘汰树的一小部分，如图2-9所示。

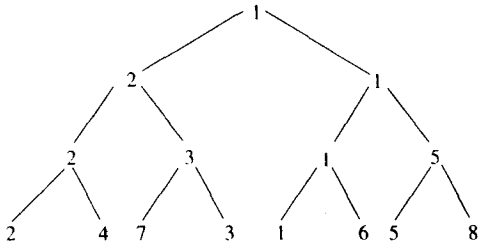


图2-8 找最小数的淘汰树

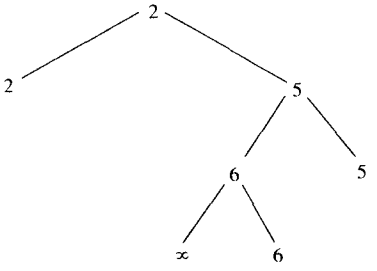


图2-9 找出第二小的数

在每次找出一个最小的数之后，用∞来代替它，很容易找出下一个最小的数。例如，现在已经找出前两个最小数，第三小数可以如图2-10找出。

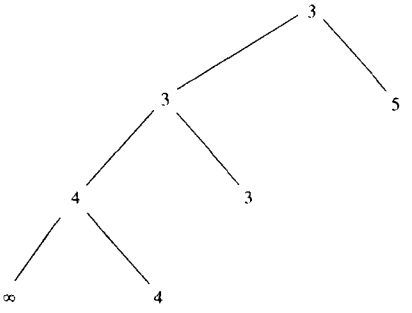


图2-10 在淘汰排序中找出第三小的数

下面确定淘汰排序的时间复杂度：

在进行$(n-1)$次比较后，找出第一小的数。对于所有其他选择，只需要$\lceil \log n \rceil - 1$次比较。因此，总的比较次数是

$$(n-1)+(n-1)(\lceil \log n \rceil - 1)$$

因此，淘汰排序的时间复杂度是$O(n \log n)$，这等于在2.4节中确定的下界。所以，淘汰排序是一个最优的排序算法。必须指明的是，在最好、平均和最坏情况下，时间复杂度为$O(n \log n)$的算法都是有效的。

淘汰排序比直接选择排序好的原因是它利用了前面的信息。遗憾的是，淘汰树需要额外的空间。完成淘汰排序需要约$2n$个附加的空间。淘汰排序可改进成堆排序，在本节的剩余部分将讨论它。

与淘汰排序相似，堆排序使用一种特殊的数据结构来存储数据。该数据结构称为堆（heap）。

堆是满足下列条件的二叉树：

(1) 此树是完全平衡的。

(2) 如果该二叉树的高度为h，那么叶子结点在h层或$h-1$层。

(3) 在h层的所有叶子尽可能在左边。

(4) 一个结点的所有后代相关的数据小于该结点上的数据。

图2-11显示了一个具有10个数的堆。

依照定义，堆的根结点$A(1)$是最大的数。假如堆已经构造成了（堆的构造将在以后讨论），那么可以输出$A(1)$。在输出最大的数$A(1)$之后，原来的堆就不再是一个堆了。现在用$A(n) = A(10)$来代替$A(1)$，那么，将有如图2-12所显示的树。

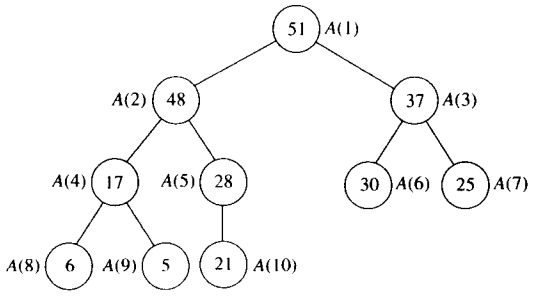


图2-11 一个堆

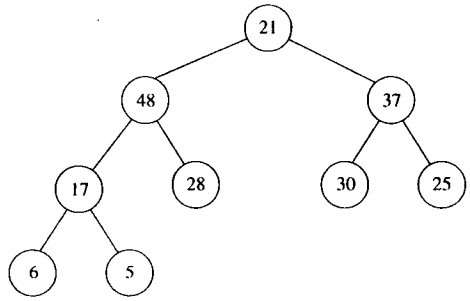


图2-12 用$A(10)$代替$A(1)$

图2-12所示的平衡二叉树不是堆，但是它可以很容易地恢复成一个堆，如图2-13所示。图2-13c所示的二叉树是一个堆。

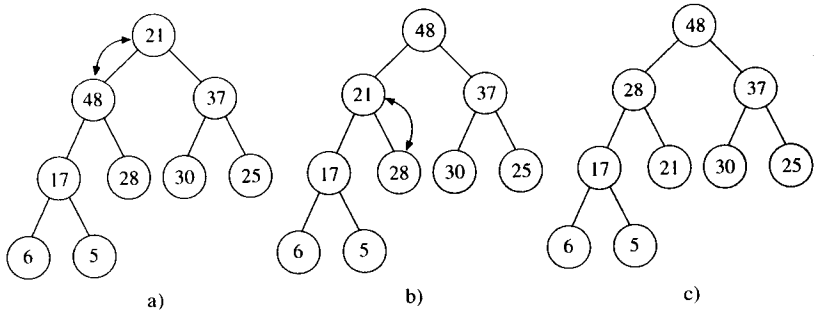


图2-13 堆的恢复

通过图2-14能更好地理解恢复过程。

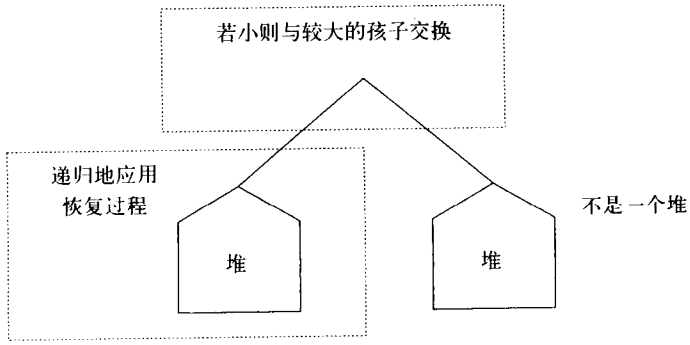


图2-14 恢复过程

算法2-6 堆的恢复过程Restore(i, j)

输入：$A(i)$，$A(i+1)$，…，$A(j)$。
输出：$A(i)$，$A(i+1)$，…，$A(j)$成为一个堆。

```
如果A(i)不是叶子，且A(i)的某个孩子含有比A(i)大的元素，那么
Begin
  令A(h)是A(i)的最大的孩子元素
  交换A(i)和A(h)
  Restore(h, j)
End
```

参数j用于确定$A(i)$是否叶子，以及$A(i)$是否有两个孩子。如果$i > j/2$，那么$A(i)$是叶子，且restore(i，j)什么也不做，因为$A(i)$自身已经是一个堆了。

在堆排序中有两个重要的方面：

(1) 堆的构建。

(2) 移走最大的数，并且恢复堆。

假设堆已经构建好了，那么堆排序描述如下：

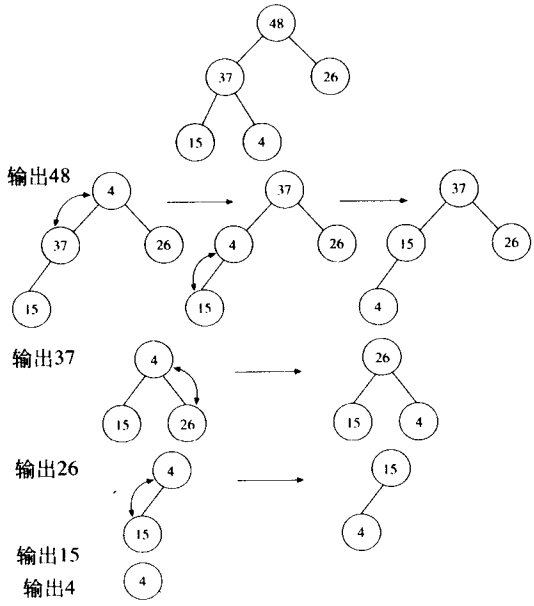
算法2-7 堆排序

输入：$A(1)$，$A(2)$，…，$A(n)$，其中每个$A(i)$已经是构建好的堆结点。
输出：$A(i)$的有序序列。

```
For i := n down to 2 do
Begin
  Output A(1)
  A(1): = A(i)
  Delete A(i)
  Restore(1, i-1)
End
Output A(1)
```

例2-6 堆排序

下面的步骤说明一个典型的堆排序实例。



堆漂亮的特性是能够用一个数组来表示一个堆。也就是，不需要指针，因为堆是一棵完全平衡的二叉树，每个结点和它的孩子都唯一地确定。规则是非常简单的：如果存在的话，那么$A(h)$的孩子就是$A(2h)$和$A(2h+1)$。

在图2-11中所示的堆现存储在一个数组中：

$A(1)$	$A(2)$	$A(3)$	$A(4)$	$A(5)$	$A(6)$	$A(7)$	$A(8)$	$A(9)$	$A(10)$
51	48	37	17	28	30	25	6	5	21

例如，考虑$A(2)$，它的左孩子是$A(4)=17$。

考虑$A(3)$，它的右孩子是$A(7)=25$。

堆排序的整个过程能在一个数组中进行。例如，例2-6中的堆排序可以描述如下：

```
                48   37   26   15   4
48 ←——          4    37   26   15
                 ⌣____⌣
                37    4   26   15
                      ⌣_____⌣
                37   15   26    4
37 ←——          4    15   26
                 ⌣_____⌣
                26   15    4
26 ←——          4    15
                 ⌣____⌣
                15    4
15 ←——          4
4  ←——
```

堆的构建

为了构建一个堆，考虑图2-14，其中的二叉树并不是一个堆，但是在树根下面的两棵子树是堆。对于这种树，可以使用恢复过程来“构建”堆。堆的构建基于上面的思想，从任意的完全平衡二叉树开始，反复地调用恢复过程来逐渐将它转变成一个堆。

令$A(1)$，$A(2)$，…，$A(n)$是一棵完全平衡二叉树，它在最高层的叶子结点尽量在左边排列。对此二叉树，我们能看到$A(i)$（$i=1, 2, \cdots, \lfloor n/2 \rfloor$）是具有孩子的内部结点，$A(i)$（$i=\lfloor n/2 \rfloor+1, \cdots, n$）一定是没有孩子的叶子结点。全部的叶子结点可以认为已经是堆。因此，不必对这些叶子执行任何操作。堆的构建从恢复根结点在$\lfloor n/2 \rfloor$处的子树开始。构建堆的算法如下：

算法2-8 堆的构建

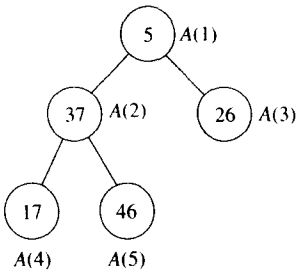
输入：$A(1)$，$A(2)$，…，$A(n)$。

输出：$A(1)$，$A(2)$，…，$A(n)$成为一个堆。

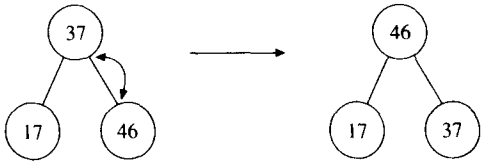
```
For i := ⌊n/2⌋ down to 1 do
    Restore(i, n)
```

例2-7 堆的构建

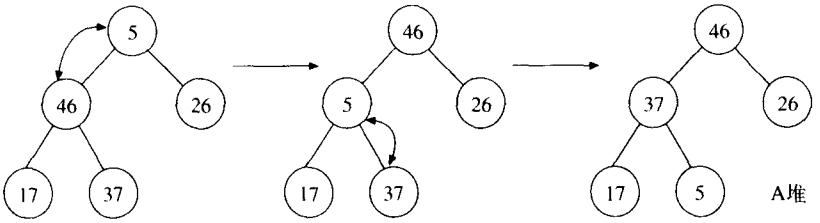
考虑下面的二叉树，它不是一个堆。



在这个堆中，$n=5$，$\lfloor n/2 \rfloor = 2$。因此，恢复根结点在$A(2)$的子树如下：



然后恢复$A(1)$：



堆排序是淘汰排序的改进，因为它使用线性数组来表示堆。接下来分析堆排序的时间复杂度。

构建堆的最坏情况分析

假设要排序n个数，因此完全平衡二叉树的深度d为$\lfloor \log n \rfloor$。对于每个内部结点，必须进行两次比较。假设一个内部结点的层次为L，那么，在最坏情况下完成恢复过程必须进行$2(d-L)$次比较。L层最多的结点数是2^L，那么在整个构建阶段，比较的总次数至多为

$$\sum_{L=0}^{d-1} 2(d-L)2^L = 2d\sum_{L=0}^{d-1} 2^L - 4\sum_{L=0}^{d-1} L2^{L-1}$$

2.2节中，在等式(2-1)中已证明了

$$\sum_{L=0}^{k} L2^{L-1} = 2^k(k-1)+1$$

所以，

$$\begin{aligned}
\sum_{L=0}^{d-1} 2(d-L)2^L &= 2d\sum_{L=0}^{d-1} 2^L - 4\sum_{L=0}^{d-1} L2^{L-1} \\
&= 2d(2^d-1) - 4(2^{d-1}(d-1-1)+1) \\
&= 2d(2^d-1) - 4(d2^{d-1} - 2^d + 1) \\
&= 4\cdot 2d - 2d - 4 \\
&= 4\cdot 2^{\lfloor \log n \rfloor} - \lfloor 2\log n \rfloor - 4 \\
&= cn - \lfloor 2\log n \rfloor - 4 \qquad \text{其中} 2 \leqslant c \leqslant 4 \\
&\leqslant cn
\end{aligned}$$

因此，在最坏情况下构建一个堆总的比较次数为$O(n)$。

从堆中删除元素的时间复杂度

如前所述，在一个堆构建后，堆的根是最大的数，现在它可被删除（或输出）。现在分析从具有n个元素的堆中，输出全部的数据元素所需要的比较次数。注意，在删除一个数后，如果剩余i个元素，那么在最坏的情况下，恢复堆需要$2\lfloor \log i \rfloor$次比较。因此，删除全部数据所需要的总步数为

$$2\sum_{i=1}^{n-1}\lfloor \log i \rfloor$$

为了估算这个公式，考虑$n = 10$的情况。

$\lfloor \log 1 \rfloor = 0$

$\lfloor \log 2 \rfloor = \lfloor \log 3 \rfloor = 1$

$\lfloor \log 4 \rfloor = \lfloor \log 5 \rfloor = \lfloor \log 6 \rfloor = \lfloor \log 7 \rfloor = 2$

$\lfloor \log 8 \rfloor = \lfloor \log 9 \rfloor = 3$

可以看到

2^1个数等于$\lfloor \log 2^1 \rfloor = 1$

2^2个数等于$\lfloor \log 2^2 \rfloor = 2$

及$10 - 2^{\lfloor \log 10 \rfloor} = 10 - 2^3 = 2$个数等于$\lfloor \log n \rfloor$。

总之，

$$\begin{aligned}2\sum_{i=1}^{n-1}\lfloor \log i \rfloor &= 2\sum_{i=1}^{\lfloor \log n \rfloor -1} i2^i + 2(n-2^{\lfloor \log n \rfloor})\lfloor \log n \rfloor \\ &= 4\sum_{i=1}^{\lfloor \log n \rfloor -1} i2^{i-1} + 2(n-2^{\lfloor \log n \rfloor})\lfloor \log n \rfloor\end{aligned}$$

利用$\sum_{i=1}^{k} i2^{i-1} = 2^k(k-1)+1$（2.2节中的等式(2-1)），可以得到

$$\begin{aligned}2\sum_{i=1}^{n-1}\lfloor \log i \rfloor &= 4\sum_{i=1}^{\lfloor \log n \rfloor -1} i2^{i-1} + 2(n-2^{\lfloor \log n \rfloor})\lfloor \log n \rfloor \\ &= 4(2^{\lfloor \log n \rfloor -1}(\lfloor \log n \rfloor -1-1)+1) + 2n\lfloor \log n \rfloor - 2\lfloor \log n \rfloor 2^{\lfloor \log n \rfloor} \\ &= 2\cdot 2^{\lfloor \log n \rfloor}\lfloor \log n \rfloor - 8\cdot 2^{\lfloor \log n \rfloor -1} + 4 + 2n\lfloor \log n \rfloor - 2\cdot 2^{\lfloor \log n \rfloor}\lfloor \log n \rfloor \\ &= 2\cdot n\lfloor \log n \rfloor - 4\cdot 2^{\lfloor \log n \rfloor} + 4 \\ &= 2n\lfloor \log n \rfloor - 4cn + 4 \qquad \text{其中} 2 \leqslant c \leqslant 4 \\ &= O(n \log n)\end{aligned}$$

因此，从一个堆中得到所有元素的有序序列，其最坏情况时间的复杂度为$O(n \log n)$。

总之，可推导出在最坏情况下，堆排序的时间复杂度为$O(n \log n)$。在此要强调堆排序的时间复杂度达到$O(n \log n)$，本质上是因为它以这样的方法使用数据结构，即每个输出操作至多需要$\lfloor \log i \rfloor$步，其中i为剩余的数据元素个数。对于堆排序来说，这个聪明的数据结构设计是最为本质的。

2.6 排序的平均情况下界

在2.4节中已讨论了排序的最坏情况下界。本节将推导出排序问题的平均情况下界，仍使

用二叉决策树模型。

正如前面所讨论，每个基于比较的排序算法都能使用二叉决策树描述。在这样的二叉决策树中，从树的根结点到叶子结点的一条路径对应于算法对一个特定输入实例的执行过程。另外，路径的长度等于对输入数集进行的比较次数。定义树的外部路径长度（external path length）为从根结点到每个叶子结点的路径长度的总和。那么基于比较的排序算法的平均时间复杂度等于被$n!$片叶子结点所分割算法对应的二叉决策树的外部路径长度。

为了找出排序的平均时间复杂度下界，必须找到拥有$n!$个叶子结点的所有可能二叉树的最小外部路径长度。在拥有固定数目的叶子结点的所有可能的二叉树中，平衡二叉树的外部路径长度最短。也就是，如果层次是d的话，那么所有叶子结点在d层或$d-1$层。考虑图2-15。图2-15a中的二叉树不是平衡的。这棵树的外部路径长度是$4\times3+1=13$。现在可以将Y的两个孩子附加到X上来减少它的外部路径长度，现在的外部路径长变成为$2\times3+3\times2=12$。

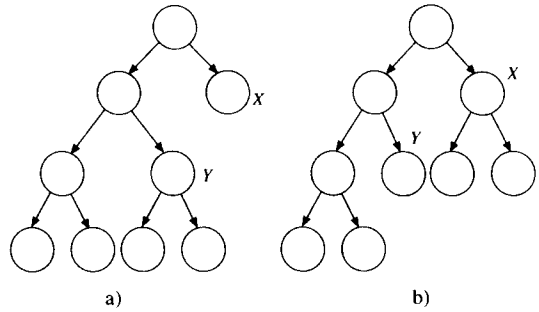


图2-15 非平衡二叉树的修改

一般的情况描述在图2-16中。假设在a层上有一个叶子结点，树的深度为d，其中$a\leqslant d-2$。这棵树可以这样修改，不改变叶子结点的个数而使外部路径长度减少。为了修改此树，选择在d层上有孩子的$d-1$层结点。令在a层上的叶子结点和在$d-1$层上的结点分别用X和Y表示。将Y的孩子移动到X上，对于结点Y，它原来有2个孩子，且路径长度的和为$2d$。现在，Y变成了一个叶子结点，且它的路径长度为$d-1$。这个移动使外部路径长度减少了$2d-(d-1)=d+1$。对于X结点，之前它有一个叶子结点。现在，它成为一个内部结点，它的两个孩子结点成为叶子结点。原先，它的路径长度是a，现在两个新的路径长度总和是$2(a+1)$。因此，添加操作使外部路径长度增加了$2(a+1)-a=a+2$。这样，净变化是$(d+1)-(a+2)=(d-a)-1\geqslant2-1=1$。即净变化减少了外部路径长度。

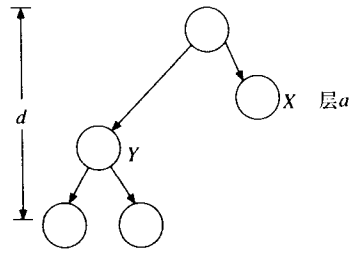


图2-16 不平衡的二叉树

因此，一棵非平衡二叉树可以通过修改使外部路径长度减少。当且仅当该树是平衡的，二叉树的外部路径长度最小。

令共有c个叶子结点。现在来计算有c个叶子结点的平衡二叉树的外部路径长度，通过下面的推理可以得出外部路径长度：

(1) 树的深度是$d=\lfloor\log c\rfloor$，叶子结点只出现在d层或$d-1$层上。

(2) 令在$d-1$层上有x_1个叶子结点，在d层上有x_2个叶子结点，那么有$x_1+x_2=c$。

(3) 为简化讨论，假设在d层上的结点数为偶数。读者很容易理解，如果在d层上的结点数是奇数，那么下面的结果依然成立。对d层上的每两个结点，在$d-1$层上都有一个父结点。这个父结点不是叶子结点。因此，可以有下面的公式：

$$x_1+\frac{x_2}{2}=2^{d-1}$$

(4) 求解这两个方程，可以得到

$$\frac{x_2}{2}=c-2^{d-1}$$

$$x_2 = 2(c-2^{d-1})$$
$$x_1 = 2^d - c$$

(5) 外部路径长度是

$$\begin{aligned} & x_1(d-1) + x_2 d \\ = & (2^d - c)(d-1) + (2c-2^d)d \\ = & c + cd - 2^d \end{aligned}$$

(6) 由于$d = \lceil \log c \rceil$，将$\log c \leqslant d < \log c + 1$代入到上面的等式，得到$c + cd - 2^d \leqslant c + c(\log c) - 2.2^{\log c} = c \log c - c$。因此，外部路径长度大于$c \log c - c = n! \log n! - n!$。所以排序的平均情况时间复杂度大于

$$\frac{n! \log n! - n!}{n!} = \log n! - 1$$

使用2.4节所讨论的结果，得出排序问题的平均情况下界是$\Omega(n \log n)$。

在2.2节中的例2-4中，已证明了快速排序的平均情况时间复杂度是$O(n \log n)$。因此，就平均情况而言，快速排序是最优的算法。

在2.2节中的例2-3中，已证明了直接选择排序的平均情况时间复杂度也是$O(n \log n)$。然而，必须明白这个时间复杂度是依据标记的改变。直接选择排序在平均情况和最坏情况下的比较次数是$n(n-1)/2$。因为在实际的编程中，主要的时间因素是比较的次数，所以直接选择排序在实践中是非常慢的。

著名的冒泡排序和直接插入排序一样，平均情况的时间复杂度是$O(n^2)$。实验告诉我们冒泡排序比快速排序慢得多。

最后，看看在2.5节中所讨论的堆排序。在最坏的情况下，堆排序的时间复杂度是$O(n \log n)$，堆排序的平均情况时间复杂度还没有找到。然而，因为在这一节中找出的下界，我们知道它必定大于或等于$O(n \log n)$，但是，因为它的最坏情况下的时间复杂度是$O(n \log n)$，所以它不能比$O(n \log n)$大。因此，我们推断出堆排序平均情况下的时间复杂度是$O(n \log n)$。

2.7 通过神谕改进下界

在前面各节中，已说明了如何用二叉决策树模型得到排序的下界，这只能侥幸得到一个好的下界，也就是存在一个最坏情况下的时间复杂度恰好等于这个下界的算法。因此，我们能确定下界不能再提高了。

在本节中，将要说明二叉决策树模型不能产生更有意义下界的情况。也就是，将说明通过使用二叉决策树模型能进一步提高所得到的下界。

考虑合并问题（merging problem）。如果合并算法基于比较和交换操作，那么能够使用决策树模型。合并问题的下界能用推导排序下界的方法得出。在排序中，叶子结点的数目是$n!$，排序的下界因此为$\lfloor \log_2 n! \rfloor$。在合并问题中，叶子结点的数目也是想区别的不同情况的数目。因此，给定2个有序序列A和B，分别有m和n个元素，那么能合并成多少个不同的队列呢？再一次简化讨论，假如全部$(m+n)$个元素是不同的，在不打乱序列A和B原来的次序情况下，将n个元素合并到m个元素中，总共有$\binom{m+n}{n}$种方法，这意味着能得到合并问题的下界为

$$\left\lceil \log \binom{m+n}{n} \right\rceil$$

然而，从来还没有找出任何合并算法能达到这个下界。

考虑一个通常的比较两个有序序列的最大元素，并输出较小元素的合并算法。在这种合并算法中，最坏情况下的时间复杂度为$m+n-1$，这大于或等于

$$\left\lceil \log\binom{m+n}{n} \right\rceil$$

即我们得到如下的不等式

$$\left\lceil \log\binom{m+n}{n} \right\rceil \leqslant m+n-1$$

我们怎样弥补这个差距？像前面讨论的一样，增大下界或者找到一个有更低时间复杂度的好算法。事实上，很有趣地是当$m=n$时，合并问题的另一个下界是$m+n-1=2n-1$。

可通过使用神谕方法（oracle approach）来说明该情况。神谕将提供给我们一种非常难的情况（一个特定的输入数据）。如果对这个数据集应用一种算法解决该问题，算法执行很困难。使用这个数据集，可以得出最坏情况下的下界。

假设有两个数列a_1，a_2，…，a_n和b_1，b_2，…，b_n。另外，考虑一个很困难的情况$a_1<b_1<a_2<\cdots a_n<b_n$。假设某个合并算法已经正确地将$a_1$，$a_2$，…，$a_{i-1}$和$b_1$，$b_2$，…，$b_{i-1}$生成下面的有序序列：

$$a_1,\ b_1,\ \cdots,\ a_{i-1},\ b_{i-1}$$

然而，假如这个合并算法不比较a_i和b_i。显然，算法没有办法确定b_{i-1}后面是a_i还是b_i。因此，a_i和b_i必须比较。基于相同的原因，能证明在a_i放在b_{i-1}后，b_i和a_{i+1}也必须能比较。总之，每个b_i必须同a_i和a_{i+1}比较。所以，当$m=n$时，合并算法总共需要比较$2n-1$次。需要提醒读者，只有当$m=n$时，这个合并算法的下界$2n-1$才是有效的。

因为当$m=n$时，一般的合并算法需要$2n-1$次。在最坏情况下的时间复杂度等于它的下界，所以得出结论合并算法是最优的。

上面的讨论说明有时能将下界提高。

2.8 通过问题转换求下界

在前面各节中，通过直接分析问题来找下界。有些时候，这显得很困难。例如，凸包问题（convex hull problem）是找一组平面点的最小凸多边形问题。凸包问题的下界是多少？看来很难直接找到这个问题有意义的下界。但是，下面将证明通过转变该问题成排序问题，可很容易得到这个问题的下界，排序问题的下界是已知的。

假如一组数据排成有序的x_1，x_2，…，x_n。不失一般性，假设$x_1<x_2<\cdots<x_n$。然后，将每个x_i和x_i^2结合形成一个2维的点$(x_i,\ x_i^2)$，全部新产生的点都位于抛物线$y=x^2$上。考虑这n个$(x_i,\ x_i^2)$点之外所构成的凸包。如图2-17所示，这个凸包由一组有序的数组成。换句话说，通过解决凸包问题，能解决排序问题。排序的总时间等于转换的时间加上解决凸包问题的时间。因此，凸包问题的下界等于排序问题的下界减去转换的时间。也就是由于转换花费$\Omega(n)$步，凸包的下界不低于$\Omega(n\log n)-\Omega(n)=\Omega(n\log n)$，因为一个算法只需要$\Omega(n\log n)$步解决凸包问题，所以它的下界不能更进一步提高了。

在一般情况下，假设想找出问题P_1的下界，并存在一个已知下界的问题P_2。进一步，假如用一种方法能将P_2转变成P_1，在这种方法能在解决P_1之后，也能解决P_2。令$\Omega(f_1(n))$和$\Omega(f_2(n))$分别表示P_1和P_2的下界，$O(g(n))$表示将P_2转化为P_1所需要的时间，那么

$\Omega(f_1(n)) + O(g(n)) \geqslant \Omega(f_2(n))$

$\Omega(f_1(n)) \geqslant \Omega(f_2(n)) - O(g(n))$

现在举另一个例子来说明这种方法的可行性。假如想找出欧几里得最小生成树问题的下界。因为很难直接得到P_1的下界，再次考虑排序问题P_2。定义转换：对每个x_i，令$(x_i, 0)$是一个2维的点。只要最小生成树由这些$(x_i, 0)$点构造而成，那么排序将完成。再一次假设当且仅当$j = i + 1$时，$x_1<x_2<\cdots<x_n$，在最小生成树的$(x_i, 0)$和$(x_j, 0)$之间有一条边。因此，欧几里得最小生成树问题的解也是排序问题的解。再次，我们知道欧几里得最小生成树的下界是$\Omega(n \log n)$。

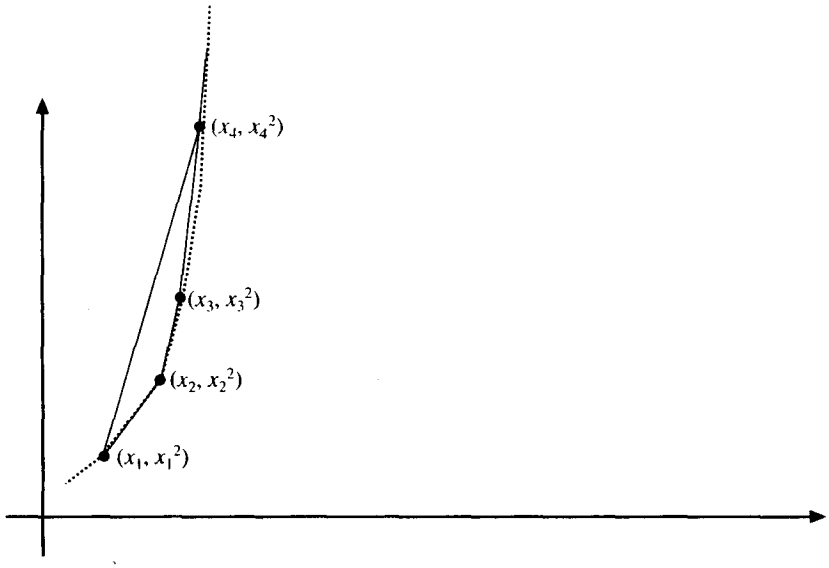


图2-17 一个排序问题的数据构造的凸包问题

2.9 注释与参考

本章介绍了算法分析的一些基本概念。对于算法分析这个主题更多的研究，可参阅Basse and Van Gelder(2000)；Aho，Hopcroft and Ullman(1974)；Greene and Knuth(1981)；Horowitz，Sahni and Rajasekaran(1998)和Purdom and Brown(1985a)。几位图灵奖获得者都是优秀的算法研究者。在1987年，ACM出版社出版了20位图灵奖获得者的演讲论文集（Ashenhurst，1987）。在该卷中，Rabin、Knuth、Cook和Karp的演讲都讨论了算法的复杂度。1986年的图灵奖获得者是Hopcroft和Tarjan。Tarjan的获图灵奖演讲是关于算法设计的，该演讲可参考文献Tarjan(1987)。文献Weide(1977)对算法分析技术也做了概述。

本章介绍了几种排序算法。为了对排序和查找有更全面的讨论，读者可以阅读文献Knuth(1973)。对于直接插入排序、二分搜索和直接选择排序等更多的分析，可分别阅读文献Knuth(1973)的5.2.1节、6.2.1节和5.2.3节。快速排序归功于Hoare(1961，1962)。堆排序由Williams(1964)发现。关于排序的下界，可参阅文献Knuth(1973)的5.3.1节。更多关于淘汰排序的信息可参阅Knuth(1973)的5.2.3节。

关于树的基本术语可在许多数据结构教科书中找到。例如，文献Horowitz and Sahni(1976)的5.1节和5.2节。树的深度也称树的高度。关于树的外部路径长度更多的分析和完全二叉树的作用可参阅文献Knuth(1969)的2.3.4.5节。

关于最小生成树问题的研究可在文献Preparata and Shamos(1985)的6.1节中找到。有更多的

关于秩确定问题的信息可在文献Shamos(1978)的4.1节和Preparata and Shamos (1985)的8.8.3节中找到。查找中值的平均时间复杂度的证明可在文献Horowitz and Sahni(1978)的3.6节中找到。

关于通过神谕改进下界的资料可参阅文献Knuth(1973)的5.3.2节和Horowitz and Sahni(1978)的10.2节。关于通过问题转换求下界的资料可参阅文献Shamos(1978)的6.1.4节和Preparata and Shamos(1985)的3.2节及5.3节。在文献Shamos(1978)和Preparata and Shamos(1985)中有更多的关于通过转换求下界的例子。

2.10 进一步的阅读资料

下界理论一直吸引着众多的研究者。关于该主题一些最近出版的论文包括：Dobkin and Lipton (1979)；Edwards and Elphick (1983)；Frederickson (1984)；Fredman (1981)；Grandjean (1988)；Hasham and Sack (1987)；Hayward (1987)；John (1988)；Karp (1972)；McDiarmid (1988)；Mehlhorn, Naher and Alt (1988)；Moran, Snir and Manber (1985)；Nakayama, Nishizeki and Saito (1985)；Rangan (1983)；Traub and Wozniakowski (1984)；Yao (1981)；and Yao (1985)。

最新的一些非常有兴趣的出版论文可参阅：Berman, Karpinski, Larmore, Plandowski and Rytter (2002)；Blazewicz and Kasprzak (2003)；Bodlaender, Downey, Fellows and Wareham (1995)；Boldi and Vigna (1999)；Bonizzoni and Vedova (2001)；Bryant (1999)；Cole (1994)；Cole and Hariharan (1997)；Cole, Farach-Colton, Hariharan, Przytycka and Thorup (2000)；Cole, Hariharan, Paterson and Zwick (1995)；Crescenzi, Goldman, Papadimitriou, Piccolboni and Yannakakis (1998)；Darve (2000)；Day (1987)；Decatur, Goldreich and Ron (1999)；Demri and Schnoebelen (2002)；Downey, Fellows, Vardy and Whittle (1999)；Hasegawa and Horai (1991)；Hoang and Thierauf (2003)；Jerrum (1985)；Juedes and Lutz (1995)；Kannan, Lawler and Warnow (1996)；Kaplan and Shamir (1994)；Kontogiannis (2002)；Leoncini, Manzini and Margara (1999)；Maes (1990)；Maier (1978)；Marion (2003)；Martinez and Roura (2001)；Matousek (1991)；Naor and Ruah (2001)；Novak and Wozniakowski (2000)；Owolabi and McGregor (1988)；Pacholski, Szwast and Tendera (2000)；and Peleg and Rubinovich (2000)；and Ponzio, Radhakrishnan and Venkatesh (2001)。

习题

2.1 给出在最好、最坏和平均情况下的冒泡排序所需要交换元素的次数，冒泡排序几乎在任何关于算法的书上都有定义。最好情况下和最坏情况下的分析是简单的，平均情况下的分析可以通过以下的过程得到：

(1) 定义一个排列的逆序。令a_1，a_2，…，a_n是集合（1，2，…，n）的排列。如果$i<j$且$a_j<a_i$，那么(a_i, a_j)称为这个排序的一个逆序。例如，(3，2)、(3，1)、(2，1) 和 (4，1) 是排列 (3，2，1，4) 的全部逆序。

(2) 找出已知正好有k个逆序的排序和n个元素恰好有k个逆序的排序次数之间的概率关系。

(3) 利用归纳法，证明冒泡排序所需要的平均情况下交换元素的次数为$n(n-1)/4$。

2.2 为冒泡排序写一个程序，运行一个实验说明它的平均性能确实是$O(n^2)$。

2.3 找到Ford-Johnson排序算法。该算法在很多关于算法的书中都有介绍。已经证明在$n\leqslant 12$时，算法是最优的。在计算机上执行这个算法，与另一个排序算法比较。你喜欢这个算法吗？如果不喜欢，设法确定分析有什么问题。

2.4 证明：对5个数进行排序，至少需要7次比较。说明Ford-Johnson算法达到此下界。

2.5 证明：在有n个数的序列中找出最大的数至少需要$n-1$次比较。

2.6 证明：找出n个数的序列中第二大的数，至少需要$n-2+\lceil \log n \rceil$次比较。

提示：如果没有确定最大数，就不能确定第二大的数。因此，可以进行下面的分析：

(1) 证明：找出最大的数至少需要$n-1$次比较。

(2) 证明：总是存在某个比较序列，使得找到第二大数，需要另外的$\lceil \log n \rceil - 1$次比较。

2.7 证明：如果$T(n) = aT\left(\frac{n}{b}\right) + n^c$，那么对于$n$的$b$次幂及$T(1) = k$，有

$$T(n) = ka^{\log_b n} + n^c\left(\frac{b}{a-b^c}\right)\left(\left(-\frac{a}{b^c}\right)^{\log_b n} - 1\right)。$$

2.8 证明：如果$T(\sqrt{n}) = \sqrt{n}T(\sqrt{n}) + n$，$T(m) = k$，$2^i\sqrt{n} \leqslant m$，那么$T(n) = kn^{(2^i-1)/2^i} + in$。

2.9 阅读文献Horowitz and Sahni 1978中的定理10.5，该定理的证明对找出下界提供了一种好方法。

2.10 证明：对任何只基于比较的查找算法，二分查找是最优的。

2.11 已知下面的函数对，使得第一个函数比第二个函数大的最小的n值是多少？

(a) 2^n，$2n^2$。

(b) $n^{1.5}$，$2n\log_2 n$。

(c) n^3，$5n^{2.81}$。

2.12 $\Omega(n \log n)$时间是对从1到C范围内的n个整数的排序问题的下界吗？其中C是常数。为什么？

第3章 贪 心 法

贪心法（greedy method）是解决某些优化问题的一种策略。假设我们可以通过一系列的决策来解决某个问题。贪心法使用如下的方法：在每个阶段做出的决策都是局部最优的决策。对于某些问题，正如下面将看到的，这些局部最优解将最终成为全局最优解。这里需要强调的是只有一部分优化问题能够通过贪心法来解决。假如局部最优解不能够导出全局最优解时，贪心法仍然是值得推荐的。就像在后面将会看到的，因为它至少得到了一个通常人们可接受的解。

现在通过一个例子来描述贪心法的本质。研究已知n个数的集合，要求从中选择出k个数，使得在所有可能挑出的k个数中，它们的和最大。

为解决该问题，可以考查从n个数中选择出k个数的所有方法。这当然是一种比较愚蠢的方法，因为可以挑出前k个最大的数，它们就是问题的解。或者，可以用下面的算法来解决该问题：

For i := 1 to k do

选择出最大的数，并将它从输入中删除。

Endfor

上面的算法是一种典型的贪心方法，在每一步选择出最大的数。

考虑另一种使用贪心法的情况。在图3-1中，要求找到从v_0到v_3的最短路径。对于该图，可以通过找出一条从v_i到v_{i+1}（i = 0，1，2）的最短路径来解决这个问题。即首先找到从v_0到v_1的最短路径，然后找到从v_1到v_2的最短路径，等等。显然，最后将会得到最优解。

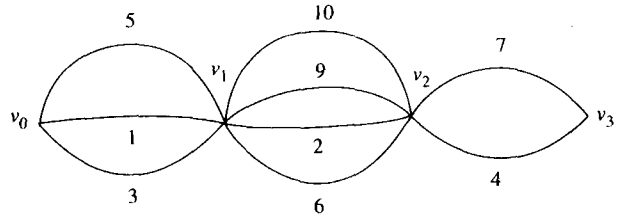


图3-1 一个贪心法解决问题的实例

然而，也可以很容易地给出一个使用贪心法不能解决的例子，考虑图3-2。

在图3-2中，也要求找出一条从v_0到v_3的最短路径。如果使用贪心方法，那么第1步要先找出一条从v_0到某个结点的最短路径，选择结点$v_{1,2}$。下一步要找出从$v_{1,2}$到某个结点的最短路径，在第2步中选择$v_{2,1}$。这样最终的解是

$$v_0 \rightarrow v_{1,2} \rightarrow v_{2,1} \rightarrow v_3$$

这条路径的总长度是1 + 9 + 13 = 23。

虽然能以快速方式得到，但这个解并不是最优解。事实上，最优解是

$$v_0 \rightarrow v_{1,1} \rightarrow v_{2,2} \rightarrow v_3$$

它的总长度是3 + 3 + 1 = 7。

我们将在后续的章节中介绍解决这个问题的方法。

在用贪心法解决图3-2所示的问题时出现了什么错误呢？为了回答这个问题，我们把思路

转移到下棋上来。一名优秀的棋手决不会简单地看看棋盘就走一步在当时是最优的棋，他要“向前看”，即他要想象出最有可能走的几步，同时想象他的对手将如何应对，他一定也清楚他的对手也会向前看。因此，整个棋局看起来如图3-3所示的一棵树。

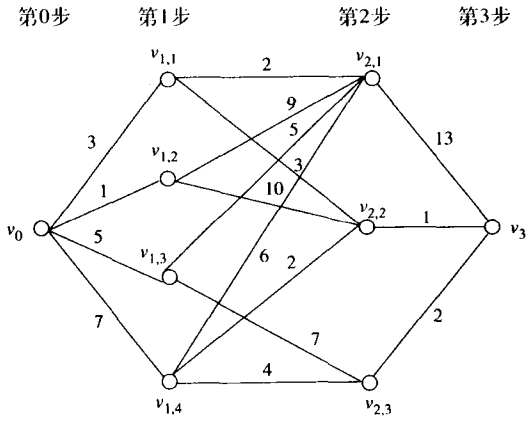


图3-2 一个贪心法解决不了的问题实例

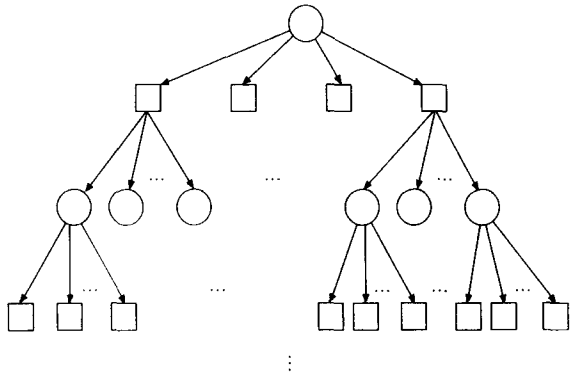


图3-3 一棵棋树

在图3-3中，圆圈表示一名棋手，方块表示他的对手。只有当整理出这样一棵棋树，才能走出聪明的、正确的一步。考虑图3-4，这是一个虚构的残局树。

从这棵残局树中，能够依据下面的推理做出应该如何走棋的决策。

(1) 由于对手总是想让我们输，我们标记I和K为LOSE（输），如果到达了其中任一个状态，我们将会输。

(2) 由于我们总是想赢，我们就把F和G标记为WIN（赢）。此外，把E标记为LOSE。

(3) 又由于对手总是想让我们输，我们把B和C分别标记为LOSE和WIN。

(4) 由于我们想赢，A标记为WIN。

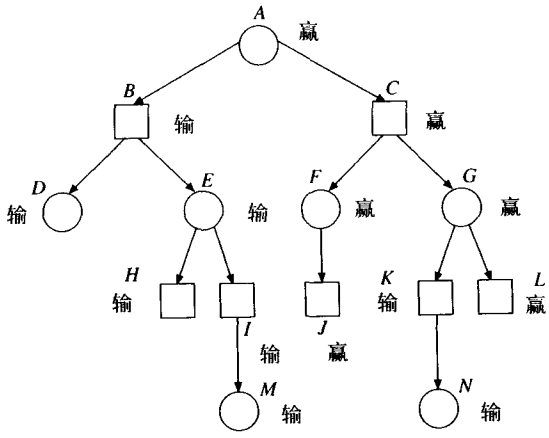


图3-4 一棵残局树

从上面的讨论，我们明白在做决策的时候，经常必须要向前看，然而贪心法从来不向前看。考虑图3-2，为了找出从v_0到v_3的最短路径，也需要向前看。对于从第1步的结点中选出一个结点，我们必须知道第1步的每个结点到v_3的最短路径。令$d\min(i, j)$表示从结点i到j间的最短距离，那么

$$d\min(v_0, v_3) = \begin{cases} 3 + d\min(v_{1,1}, v_3) \\ 1 + d\min(v_{1,2}, v_3) \\ 5 + d\min(v_{1,3}, v_3) \\ 7 + d\min(v_{1,4}, v_3) \end{cases}$$

现在读者应该明白由于贪心法不向前看，因此它不能得到最优解。在本章的剩余部分中，将展示贪心法可以解决的许多有趣例子。

3.1 生成最小生成树的Kruskal算法

用贪心法解决的一个有名问题是最小生成树问题（minimum spanning tree problem）。在本节中，将介绍生成最小生成树的Kruskal方法（Kruskal's method）。最小生成树可以在欧几里得空间点上或在一个图上定义。对于Kruskal方法，最小生成树是在图上定义的。

定义 设$G=(V, E)$表示一个带权的连通无向图，其中V表示顶点集合，E表示边集合。G的一棵生成树是一棵无向树$S=(V, T)$，其中T是E的子集。生成树的权是T的所有权之和。G的最小生成树是G的具有最小权值的生成树。

例3-1 最小生成树

图3-5显示了由5个顶点和8条边构成的图。图3-6显示了该图的一些生成树。图3-7显示了该图的一棵最小生成树，它的权为：50 + 80 + 60 + 70 = 260。

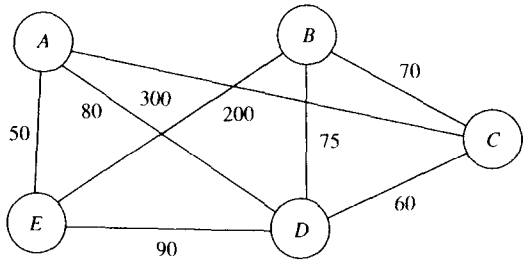
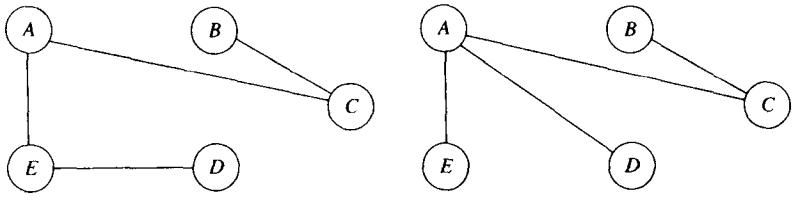


图3-5 带权连通无向图



a) b)

图3-6 两棵生成树

构造最小生成树的Kruskal方法可以简单地描述如下：

(1) 从边集合中选择一条具有最小权值的边，它将形成初始构造的部分子图，这个子图以后将发展成为一棵最小生成树。

(2) 将次小权值的边加入这个构造的部分子图中，但必须保证这个边的加入不会形成回路，否则将放弃这条边。

(3) 如果生成树包含了$n-1$条边，则终止；否则转到(2)。

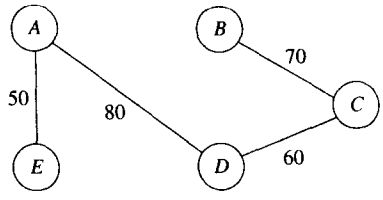


图3-7 最小生成树

算法3-1 Kruskal最小生成树算法

输入：一个带权的连通无向图$G=(V, E)$。

输出：图G的一棵最小生成树。

```
T := φ
While  T所含的边数少于n-1  do
Begin
  从E中选择一条最小权值的边(v, w)
  从E中删除边(v, w)
  If（将(v, w)加入T中没有形成一个回路）then
      将(v, w)加入T
  Else
     放弃(v, w)
End
```

例3-2 Kruskal算法

考虑图3-5中的无向图，边排序成如下顺序：

$(A, E)(C, D)(B, C)(B, D)(A, D)(E, D)(E, B)(A, C)$

现在构造最小生成树，如图3-8所示。

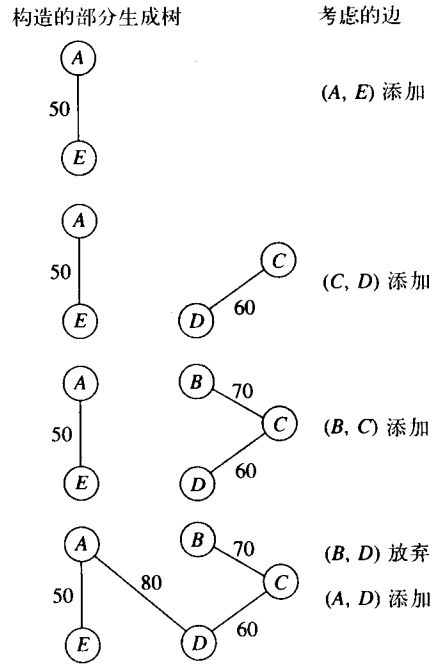


图3-8 使用Kruskal算法生成一棵最小生成树

读者应当注意到我们不必将边排序。实际上，在2.5节所介绍的堆也可以用于选择下一条最小权值的边。

仍然有一个问题要解决：如何能有效地判定加入的边是否形成回路呢？这可以很容易地解决。在Kruskal算法执行期间，构造的部分子图是由许多树构成的生成森林，因此可以将一棵树的点集限制在一个单独的集合中。在图3-9中有两棵树，它们可以分别表示为$S_1 = \{1, 2, 3, 4\}$和$S_2 = \{5, 6, 7, 8, 9\}$。假如下一条要加入的边是(3，4)。由于顶点3和4都在集合S_1中，这将形成一条回路。所以边(3，4)不能加入。同样地，也不能将边(8，9)加入，由于顶点8和9都在S_2中。但是，可以加入边(4，8)。

基于上述讨论，我们明白Kruskal算法由以下的步骤所决定：

(1) 排序。这将花费$O(m \log m)$步，其中m是图中边的数目。

(2) 两个集合的合并。将两棵树合并时，该步是必须的。当插入一条边连接两棵子树时，实质上是将两个集合合并。例如，要将边(4，8)加入图3-9的生成树中，是将集合{1，2，3，4}和{5，6，7，8，9}合并成{1，2，3，4，5，6，7，8，9}。因此，应当完成一种操作，也就是将两个集合合并。

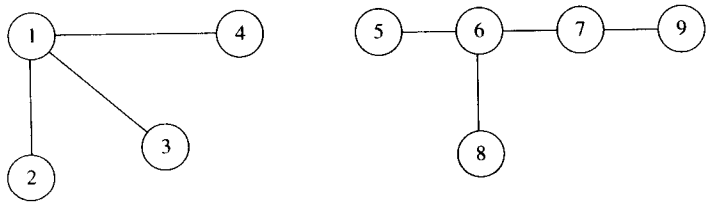


图3-9 一个生成森林

(3) 在集合中找出一个元素。注意，当判定是否能够加入一条边的时候，必须考察这两个顶点是否在一个顶点集中。因此，应当完成一种操作，称为查找操作（find operation），这个操作判定一个元素是否在一个集合中。

在第9章中，将会证明合并和查找操作都将花费$O(m)$步。因此，Kruskal算法的总时间由排序来决定，它花费的时间为$O(m \log m)$。在最坏的情况下，$m = n^2$，因此，Kruskal算法的时间复杂度为$O(n^2 \log n)$。

现在证明Kruskal算法是正确的，也就是，它产生了一棵最小生成树。假定与所有边相关的权值都不同，并且$|e_1|<|e_2|<\cdots<|e_m|$，其中$m = |E|$。令T表示由Kruskal算法产生的生成树，T'表示一棵最小生成树。我们将证明$T = T'$。假设不是这样，令e_i是T中最小权值的边，但它不在

T'中。显然$i \neq 1$，将e_i加入T'中，这必然会在T'中形成一条回路。令e_j是这个回路中的一条边，但它不是T中的边，这样的e_j一定存在。否则这个回路中所有的边都在T中，这意味着在T中有一个回路，而这是不可能的。有两种可能的情况，第一种情况：e_j比e_i的权值小，也就是$j<i$。令T_k表示由Kruskal算法判断e_k是否要加入之后所生成的树。显然，对于$k<i$，T_k是T和T'的子树，这是由于已经假设了e_i是T中最小权值的边且不在T'中。由于$j<i$且e_j不是T中的边，e_j必然不会被Kruskal算法选择，这是由于将e_j加入到T_{j-1}中会形成一条回路。然而，由于T_{j-1}也是T'的一棵子树并且e_j是T中的一条边，将e_j加入到T_{j-1}中不会形成一条回路。因此，这种情况是不可能出现的。第二种情况：e_j比e_i的权值大，比如$j>i$。在这种情况下，将e_j移走，这将会产生一棵新的生成树，它的总权值比T'小。因此，T'必然与T相同。

显然Kruskal算法使用了贪心方法。在每一步中，下一条将要加入的边都是局部最优的。有趣的是最终将会得到一个全局最优解。

3.2 生成最小生成树的Prim算法

在3.1节中已介绍了找出最小生成树的Kruskal算法，在本节中，将介绍一个分别由Dijkstra和Prim独自发现的算法。Prim算法（Prim's algorithm）是一步步地建立一棵最小生成树。在任何时候，令X表示包含在部分构成的最小生成树中的顶点集，$Y = V-X$。下一条将要加入的边(u, v)是在X和Y($u \in X$，$v \in Y$)之间具有最小权值的边。此种情况如图3-10所示。下一条要加入的边是(u, v)。当这条边加入后，将v从Y中删除并将其加入X中。Prim方法的一个要点是可以从任何一个顶点开始，这相当方便。

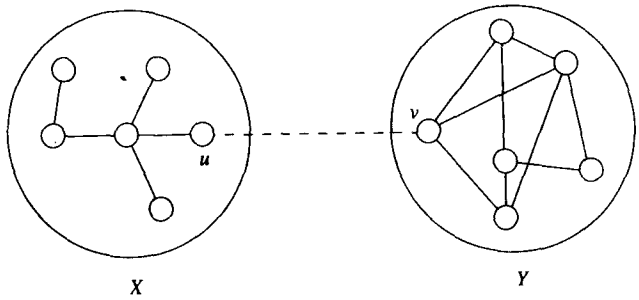


图3-10 Prim方法的一个实例

接下来，简单地介绍一下Prim算法，稍后将详细介绍。

算法3-2 找到最小生成树的Prim算法

输入：带权的连通无向图$G = (V, E)$。

输出：G的一棵最小生成树。

步骤1. 令x是V中任意顶点，令$X = \{x\}$，$X = V-\{x\}$。

步骤2. 从E中选择一条边(u, v)，使得$u \in X$，$v \in Y$，并且(u, v)是顶点集X与Y之间的边中权值最小的。

步骤3. 将u连接到v，令$X = X \cup \{v\}$，$Y = Y-\{v\}$。

步骤4. 如果Y是空的，那么终止，那么产生的树是最小生成树；否则，返回到步骤2。

例3-3 基本的Prim算法

再次研究图3-5，下面将展示如何用Prim算法生成一棵生成树。假定B是选择的初始顶点，图3-11说明了生成过程。在过程的每一步中，下一条将要加入的边都是最小地增加了总权值。例如，当部分构成的树包含顶点B，C和D时，余下的顶点集是$\{A, E\}$，连接$\{A, E\}$与$\{B, C,$

D}且具有最小权值的边是(A，D)，因此，(A，D)被加入。由于顶点A没有包含在部分构成的最小生成树中，因此，这条边的加入将不会形成回路。

读者一定对是否可以从另外其他某个顶点开始感兴趣，事实确实是这样。假设初始从顶点C开始，图3-12说明了最小生成树的构造过程。

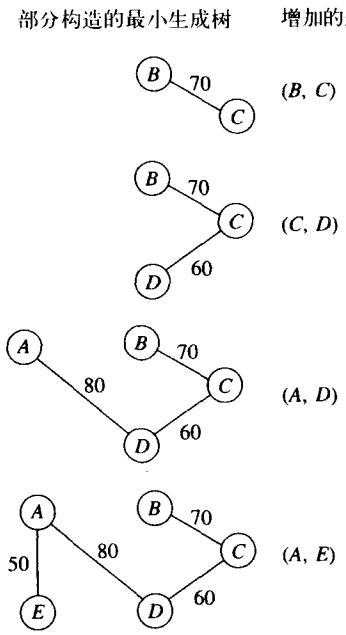


图3-11 从顶点B开始用Prim算法找出最小生成树

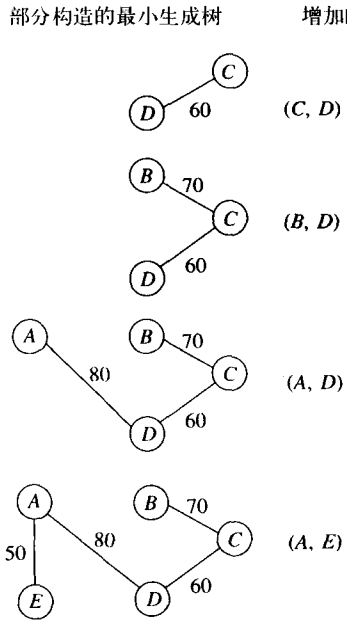


图3-12 从顶点C开始使用基本的Prim算法找出最小生成树

现在证明Prim算法是正确的，即生成的树的确是一棵最小生成树。令$G = (V, E)$表示一个带权的连通图。不失一般性，假定所有的边权值都不同。令T表示G的一棵最小生成树，T_1表示T的一棵子树，如图3-13所示。令V_1表示T_1中的顶点集，且$V_2 = V-V_1$，令(a，b)是E中具有最小权值的边，使得$a\in V_1$，$b\in V_2$。我们将证明(a，b)一定在T中。假设不是这样的，由于树是连通的，那么在T中一定有一条从a到b的路径。令(c，d) 是路径上的一条边，使得$c\in V_1$，$d\in V_2$，(c，d)的权一定比(a，b)的权大。因此，可以将(c，d)删除而将(a，b)加入从而生成一棵更小的生成树。这证明了T一定不是一棵最小生成树。因此，(a，b)一定在T中，Prim算法是正确的。

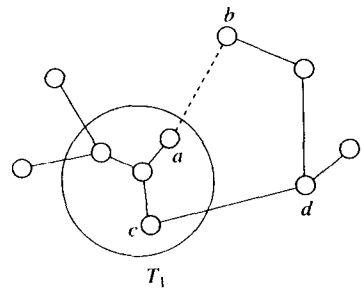


图3-13 证明Prim算法正确的最小生成树

上面的算法只是Prim算法的一个简要概述。以这种方式表示出来比较好理解。我们从$X = \{x\}$和$Y = V-\{x\}$开始，为了找出X与Y之间最小权的边，必须考查所有邻接于x的边，在最坏的情况下，将花费$n-1$步，其中n是V中的顶点数。假设y已经加入X中，即假定$X = \{x, y\}$，$Y = V-\{x, y\}$，为了找到X与Y之间最小权的边，看起来还有一个问题：是否需要再一次地考查所有依附于x的边？（当然，不需要考查x与y之间的边，由于它们都在X中。）Prim通过维护两个向量找到了一个简便的方法来避免这个麻烦。

设有n个顶点，分别标记为1，2，…，n，并设有两个向量C_1和C_2，令X表示Prim算法中部分构造的树的顶点集，且$Y = V-X$。令i是Y中的一个顶点，在所有邻接于X中顶点与Y中顶点i的

边中，令边(i, j) $(j \in X)$ 是具有最小权的边，向量C_1和C_2用于存储这个信息，令$w(i, j)$表示边(i, j)的权，那么在Prim算法的每一步中，

$C_1(i) = j$

和 $C_2(i) = w(i, j)$。

现在证明使用这两个向量能够避免重复检查边。不失一般性，假定初始时$X = \{1\}$，$Y = \{2, 3, \cdots, n\}$。显然，对于Y中的每个顶点i，如果边$(i, 1)$存在，那么$C_1(i) = 1$，$C_2(i) = w(i, 1)$，最小的$C_2(i)$决定了下一个要加入X的顶点。

再次假定顶点2是选择加入X中的顶点，因此，$X = \{1, 2\}$，$Y = \{3, 4, \cdots, n\}$。Prim算法需要决定X与Y之间最小权值的边，但是，在$C_1(i)$和$C_2(i)$的帮助下，不用再检查邻接于顶点i的边了。假定i是Y中一个顶点，如果$w(i, 2) < w(i, 1)$，将$C_1(i)$从1变到2，将$C_2(i)$从$w(i, 1)$变到$w(i, 2)$。如果$w(i, 2) \geqslant w(i, 1)$，那么不变。在$Y$中所有顶点更新了之后，可以通过考查$C_2(i)$来选择加入到$X$中的顶点，最小的$C_2(i)$决定了下一个要加入的顶点。正如读者所看到的，现在已经成功避免了重复检查所有的边，每条边仅检查一次。

下面将详细地讲述Prim算法。

算法3-3 构造最小生成树的Prim算法

输入：一个带权连通无向图$G = (V, E)$。

输出：G的一棵最小生成树。

步骤1. 令$X = \{x\}$，$Y = V - \{x\}$，其中x是V中任意顶点。

步骤2. 对于V中每个顶点y_j，置$C_1(y_j) = x$，$C_2(y_j) = \infty$。

步骤3. 对于V中每一个顶点y_j，检查y_j是否在Y中，边(x, y_j)是否存在。如果y_j在Y中、边(x, y_j)存在，并且$w(x, y_j) = b < C_2(y_j)$，置$C_1(y_j) = x$，$C_2(y_j) = b$；否则不变。

步骤4. 令y是Y中的一个顶点使得$C_2(y)$最小，$z = C_1(y)$(z一定在X中)，在部分构成树T中，将y用边(y, z)连接到z。令$X = X + \{y\}$，$Y = Y - \{y\}$，置$C_2(y) = \infty$。

步骤5. 如果Y是空集，那么终止这个过程，产生的树T是一棵最小生成树；否则，置$x = y$，并返回步骤3。

例3-4 Prim算法

考虑图3-14。

参照图3-15，将说明Prim算法是如何产生一棵最小生成树的。首先假定顶点3是初始选择的。

对于Prim算法，当一个顶点加入到部分构成的树时，C_1中的每个元素都应该检查。因此，在最坏情况下和平均情况下，Prim算法的时间复杂度是$O(n^2)$，其中n是V中的顶点数。注意到Kruskal算法的时间复杂度是$O(m \log m)$，其中m是E中边数。在m较小的情况下，Kruskal算法更好一点。在最坏情况下，就像在前面提到的，m可能等于$O(n^2)$，Kruskal算法的时间复杂度为$O(n^2 \log n)$，这比Prim算法的时间复杂度大。

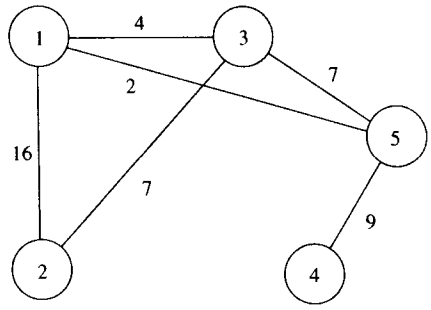


图3-14 说明Prim算法的图

3.3 单源最短路径问题

在最短路径问题（shortest path problem）中，给定一个有向图$G = (V, E)$，其中每条边与一个非负的权相关。这个权可以看作是该边的长度，G中的路径长度定义为在这条路径上所有的边长度之和。单源最短路径问题（single-source shortest path problem）是找出从指定的标记

为v_0的源点到V中所有其他顶点各自的最短路径。

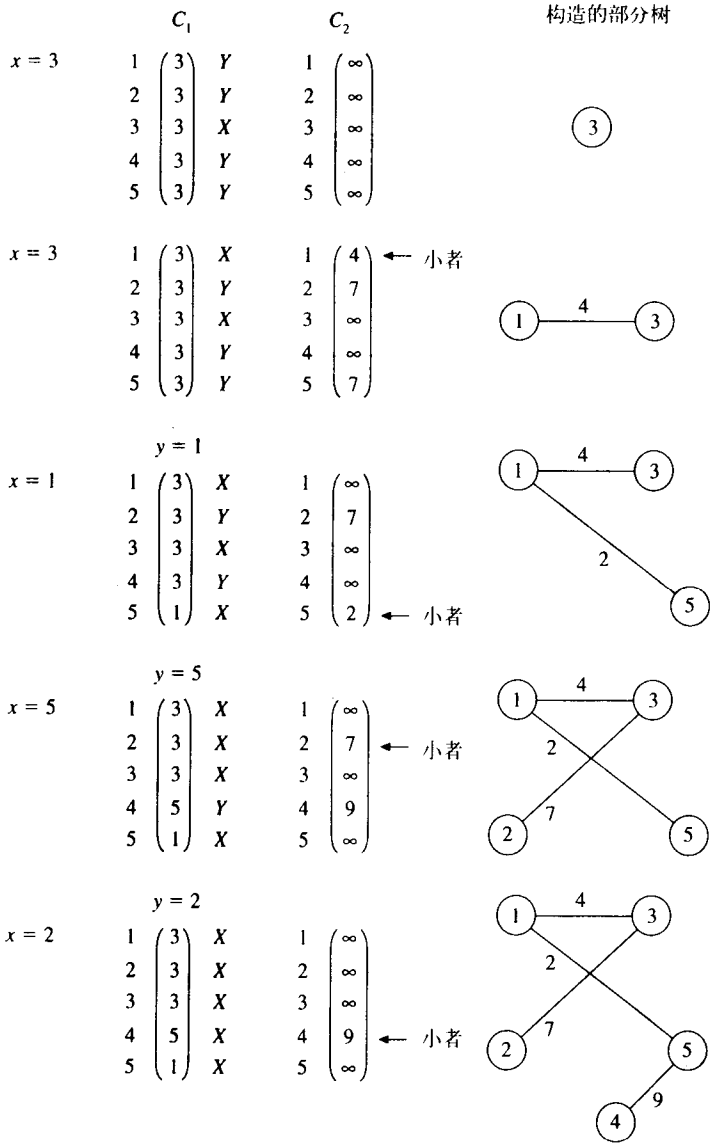


图3-15 从顶点3开始Prim算法找出一棵最小生成树

在本节将说明可以用贪心法来解决单源最短路径问题，这是由Dijkstra提出的，因此这个算法称为Dijkstra算法（Dijkstra's algorithm）。Dijkstra算法的思想与前面提到的最小生成树算法思想相似，基本的思想非常简单：从v_0到所有其他顶点的最短路径可以一个一个地找出。首先找出与v_0最近邻顶点的最短路径，然后找出与v_0第二近顶点的最短路径，这个过程一直重复到找出与v_0第n近顶点的最短路径，其中n是图中除了v_0以外的顶点数。

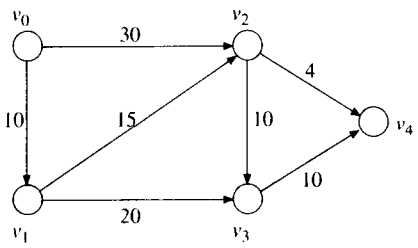


图3-16 说明Dijkstra算法的图

例3-5 单源最短路径问题

考虑图3-16中的有向图，找到所有从v_0出发的最短

路径。

算法首先确定与v_0最近的顶点是v_1，从v_0到v_1最短的路径是v_0v_1。与v_0第二近的顶点是v_2，相应的最短路径是$v_0v_1v_2$。注意到虽然这条路径是由两条边所构成的，但它仍然比只包含一条边的v_0v_2短。与v_0第三和第四近的顶点分别是v_4和v_3，整个过程可以由表3-1来说明。

表3-1 找出从v_0出发的最短路径举例

i	v_0的第i近邻点	从v_0到第i近邻点的最短路径（长度）
1	v_1	v_0v_1 (10)
2	v_2	$v_0v_1v_2$ (25)
3	v_4	$v_0v_1v_2v_4$ (29)
4	v_3	$v_0v_1v_3$ (30)

如同在最小生成树算法中一样，在Dijkstra算法中也将顶点集分成两个集合：S和$V-S$，其中S包含所有与v_0第i近的顶点，这些顶点是在最初的i步内得到的。因此，在第$(i+1)$步工作是找到与v_0第$(i+1)$近的顶点。对于这点，最重要的是不能采取任何不正确的行动。参见图3-17，图中表示了已经找到了与v_0最近的顶点v_1。看起来由于v_1v_3是连接S与$V-S$的最短边，应该选择v_1v_3作为下一条边，v_3就是第二近的顶点，但这其实是错误的。原因是要找从v_0出发的最短路径。在这种情况下，v_2是第二近的顶点，而不是v_3。

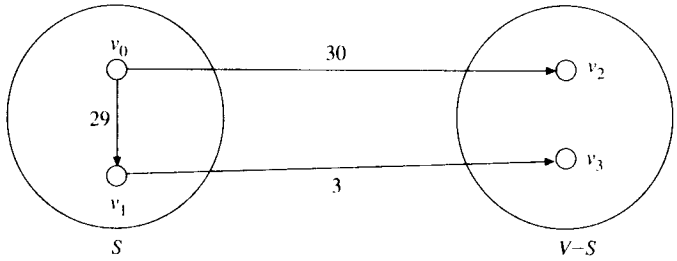


图3-17 两个顶点集S和$V-S$

下面来说明用在Dijkstra算法中高效地找出下一个与v_0最近顶点的一个重要技巧。如图3-16所示，令$L(v_i)$表示当前找到的从v_0到v_i的最短距离。在最开始时，$S=\{v_0\}$，且由于v_1和v_2都与v_0相连接，我们得到

$L(v_1)=10$

及 $L(v_2)=30$

由于$L(v_1)$是最短的，因此，v_1是与v_0最近的顶点。令$S=\{v_0, v_1\}$，那么现在只有v_2和v_3连接到S。对于v_2，它之前的$L(v_2)$等于30。然而，在v_1放入S之后，可以选用路径$v_0v_1v_2$了，它的长度为$10+15=25<30$。因此，就v_2来说，$L(v_2)'$可以计算如下：

$$\begin{aligned}L(v_2)' &= \min\{L(v_2),\ L(v_1)+v_1v_2\text{的长度}\}\\ &= \min\{30,\ 10+15\}\\ &= 25\end{aligned}$$

上面的讨论说明，由于新加入的顶点，使得当前找出的从v_0到v_2的最短路径可能不够短。如果这种情况出现了，那么应该更新最短距离。

令u表示最新加入S中的顶点，$L(w)$表示当前找到的从v_0到w的最短距离，$c(u, w)$表示连接u和w的边长度，那么根据下面的公式来更新$L(w)$：

$$L(w)=\min(L(w),\ L(u)+c(u,\ w))$$

Dijkstra算法解决单源最短路径问题概括如下。

算法3-4 生成单源最短路径的Dijkstra算法

输入：一个有向图$G=(V, E)$及一个源点v_0，对于每条边$(u, v)\in E$，有一个非负数字$c(u, v)$与之相关，$|V|=n+1$。
输出：对于每个顶点$v\in V$，得到从v_0到v的最短路径的长度。

```
S := {v0}
For i := 1 to n do
Begin
    If (v0, vi) ∈ E then
        L(vi) := c(v0, vi)
    else
        L(vi) := ∞
End
For i := 1 to n do
Begin
    从V−S中选择u使L(u)是最小的
    S := S ∪ {u}  (*将u放到S中*)
    For  V−S的所有w  do
        L(w) := min(L(w), L(u) + c(u, w))
End
```

例3-6 Dijkstra算法

考虑图3-18中的有向图。

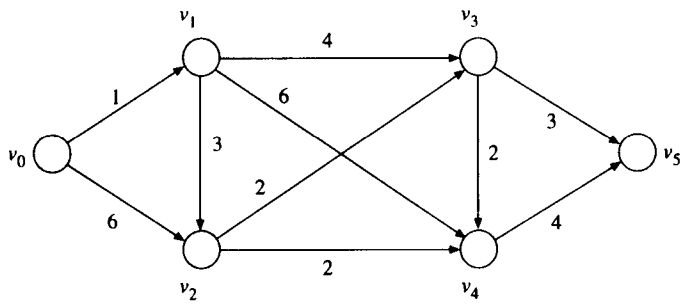


图3-18 一个带权有向图

Dijkstra算法如下执行：

(1) $S=\{v_0\}$

$L(v_1)=1$

$L(v_2)=6$

所有其他$L(v_i)$都等于∞。

$L(v_1)$是最小的，v_0v_1是从v_0到v_1的最短路径。

$S=\{v_0, v_1\}$

(2) $L(v_2)=\min(6, L(v_1)+c(v_1, v_2))$

$=\min(6, 1+3)$

$=4$

$L(v_3)=\min(\infty, L(v_1)+c(v_1, v_3))$

$=1+4$

$=5$

$L(v_4) = \min(\infty, L(v_1) + c(v_1, v_4))$
$= 1 + 6$
$= 7$

$L(v_2)$是最小的，$v_0v_1v_2$是从v_0到v_2的最短路径。

$S = \{v_0, v_1, v_2\}$

(3) $L(v_3) = \min(5, L(v_2) + c(v_2, v_3))$
$= \min(5, 4 + 2)$
$= 5$

$L(v_4) = \min(7, L(v_2) + c(v_2, v_4))$
$= \min(7, 4 + 2)$
$= 6$

$L(v_3)$是最小的，$v_0v_1v_3$是从v_0到v_3的最短路径。

$S = \{v_0, v_1, v_2, v_3\}$

(4) $L(v_4) = \min(6, L(v_3) + c(v_3, v_4))$
$= \min(6, 5 + 2)$
$= 6$

$L(v_5) = \min(\infty, L(v_3) + c(v_3, v_5))$
$= 5 + 3$
$= 8$

$L(v_4)$是最小的，$v_0v_1v_2v_4$是从v_0到v_4的最短路径。

$S = \{v_0, v_1, v_2, v_3, v_4\}$

(5) $L(v_5) = \min(8, L(v_4) + c(v_4, v_5))$
$= \min(8, 6 + 4)$
$= 8$

$v_0v_1v_3v_5$是从v_0到v_5的最短路径。

表3-2对输出做了总结。

表3-2 从顶点v_0出发的最短路径

顶 点	从顶点v_0出发的最短路径（长度）
v_1	v_0v_1 (1)
v_2	$v_0v_1v_2$ (1 + 3 = 4)
v_3	$v_0v_1v_3$ (1 + 4 = 5)
v_4	$v_0v_1v_2v_4$ (1 + 3 + 2 = 6)
v_5	$v_0v_1v_3v_5$ (1 + 4 + 3 = 8)

Dijkstra算法的时间复杂度

很容易看出在最坏的情况下，由于重复的操作来计算$L(w)$，因此，Dijkstra算法的时间复杂度是$O(n^2)$。从另一个角度看，由于每条边都要检查，解决单源最短路径问题的最小步数是$\Omega(e)$，其中e是图中的边数。在最坏的情况下，$\Omega(e) = \Omega(n^2)$。因此，从这个角度说，Dijkstra算法是最优的。

3.4 二路归并问题

已知两个有序列表L_1和L_2，$L_1 = (a_1, a_2, \cdots, a_{n_1})$，$L_2 = (b_1, b_2, \cdots, b_{n_2})$，应用下面所述

的线性归并算法（linear merge algorithm）可以将L_1和L_2合并为一个有序的列表。

算法3-5 线性归并算法

输入：两个有序列表，$L_1=(a_1, a_2, \cdots, a_{n_1})$和$L_2=(b_1, b_2, \cdots, b_{n_2})$。
输出：一个由L_1和L_2中元素所组成的有序表。

```
Begin
    i := 1
    j := 1
do
    比较a_i 和b_j
    if  a_i > b_j  then  输出b_j，并且j := j + 1
    else  输出a_i，并且i := i + 1
while (i <= n_1 and j <= n_2)
    if  i>n_1  then  输出b_j，b_{j+1}，…，b_{n_2}
    else输出a_i，a_{i+1}，…，a_{n_1}。
End
```

很容易看出，在最坏情况下需要比较的次数为$m+n-1$。当m和n相等时，可以证明线性归并算法的比较次数是最优的。如果要归并两个以上的有序表，可以用线性归并算法，只需要分别重复地归并两个有序表。这些归并过程称为二路归并（2-way merge）。由于每一步只归并两个有序表，假设已知三个有序表L_1，L_2和L_3，分别由50，30和10个元素组成，可以将L_1和L_2归并形成L_4，在最坏的情况下，这个归并步骤需要$50+30-1=79$次比较，然后通过$80+10-1=89$次比较将L_4和L_3归并。这个归并序列中需要比较的次数为168。也可以先归并L_2和L_3，再归并L_1，那么只需要比较的次数128。有许多不同的归并序列，分别需要不同的比较次数。现在关心下面的问题：有m个有序表，每个表有n_i个元素，那么哪一个归并序列是最优的归并过程，能够用最少的比较次数将这些有序表归并？

为了简化讨论，下面用$n+m$，而不是$n+m-1$，作为大小分别为n和m的表归并所需要的比较次数，这显然不会影响算法设计。看一个例子，有大小为(20，5，8，7，4)的五个有序表(L_1，L_2，L_3，L_4，L_5)。假设按如下方式归并这些表：

将L_1与L_2归并产生Z_1，需要$20+5=25$次比较
将Z_1与L_3归并产生Z_2，需要$25+8=33$次比较
将Z_2与L_4归并产生Z_3，需要$33+7=40$次比较
将Z_3与L_5归并产生Z_4，需要$40+4=44$次比较
总数 = 142次比较

这个归并方式可以很容易地用一棵二叉树表示，如图3-19a所示。

令d_i表示二叉树一个叶子节点的深度，n_i表示与这个叶子节点相关的表L_i的大小，那么相应于这个归并过程总的比较次数可以简单地记为$\sum_{i=1}^{5} d_i n_i$。在我们的例子中，$d_1=d_2=4$，$d_3=3$，$d_4=2$及$d_5=1$。因此，总的比较次数可以计算为$4\cdot 20+4\cdot 5+3\cdot 8+2\cdot 7+1\cdot 4=80+20+24+14+4=142$，这是正确的。

假设用贪心法，总是将当前两个最短的表归并，那么，归并方式为

将L_2与L_5归并产生Z_1，需要$5+4=9$次比较
将L_3与L_4归并产生Z_2，需要$8+7=15$次比较
将Z_1与Z_2归并产生Z_3，需要$9+15=24$次比较
将Z_3与L_1归并产生Z_4，需要$24+20=44$次比较

总数 = 92次比较

上面的归并过程以一棵二叉树的形式表示出来，如图3-19b所示。

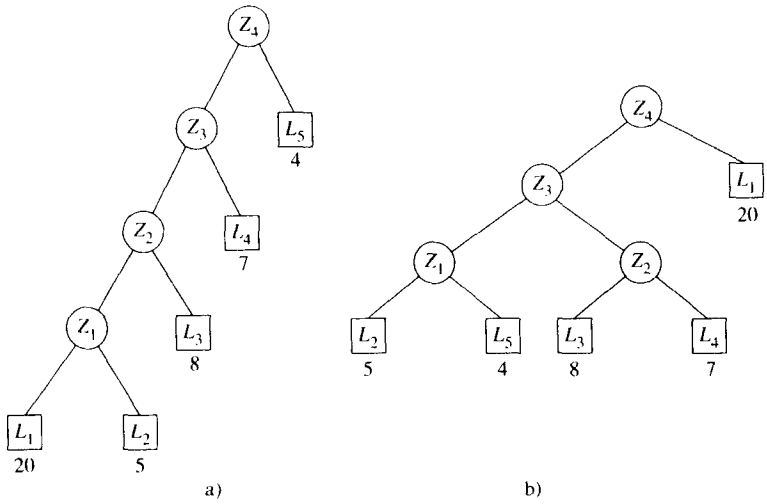


图3-19 不同的归并序列

可以再次使用公式 $\sum_{i=1}^{5} d_i n_i$，在这个例子中$d_1 = 1$，$d_2 = d_3 = d_4 = d_5 = 3$，总的比较次数可以计算为$1 \cdot 20 + 3 \cdot (5 + 4 + 8 + 7) = 20 + 3 \cdot 24 = 92$，这比图3-19a中的比较次数小。

用贪心法找到最优二路归并树描述如下：

算法3-6 贪心法生成最优二路归并树

输入：m个有序表，L_i，$i = 1, 2, \cdots, m$，每个L_i由n_i个元素组成。

输出：最优二路归并树。

步骤1. 生成m棵树，其中每棵树都只有一个权为n_i的结点（外部结点）。

步骤2. 选择两棵具有最小权的树T_1和T_2。

步骤3. 生成一棵新的树T，其根的子树为T_1和T_2，权为T_1和T_2权的和。

步骤4. 用T代替T_1和T_2。

步骤5. 如果只剩下一棵树，那么停止，返回；否则转到步骤2。

例3-7

有6个有序表，长度分别为2，3，5，7，11和13。找出一棵对于这些表来说具有最小权路径长度的扩展二叉树。

首先将2和3归并，然后寻找将5个长度为5，5，7，11和13的有序表归并成问题解的方法，接着将5和5归并，等等。归并序列如图3-20所示。

为了证明上述贪心法的正确性，首先证明存在一棵最优二路归并树，其中具有最小值的两个叶子结点安排为兄弟。令A表示与根结点距离最大的内部结点。比如，在图3-20中标记为5的结点就是这样一个结点。A的孩子L_i 和L_j一定是叶子结点。假如A的孩子的大小为n_i和n_j。如果n_i和n_j 不是两个最小的，那么可以将L_i和L_j与两个最小的结点交换而不增加二路归并树的权。因此，可以得到一棵最优二路归并树，其中两个最小的叶子结点分配为兄弟。

基于以上讨论，可以假定T是L_1，L_2，$\cdots$，L_m的一棵最优二路归并树，L_1，L_2，$\cdots$，L_m分别具有长度n_1，n_2，$\cdots$，n_m。不失一般性，$n_1 \leqslant n_2 \leqslant \cdots \leqslant n_m$，其中具有最短长度的两个表$L_1$和$L_2$是兄弟。令$A$是$L_1$和$L_2$的双亲，$T_1$表示一棵树，其中$A$被一个长度为$n_1 + n_2$的表所代替。令$W(X)$表

示二路归并树X的权，那么有

$$W(T) = W(T_1) + n_1 + n_2 \tag{3-1}$$

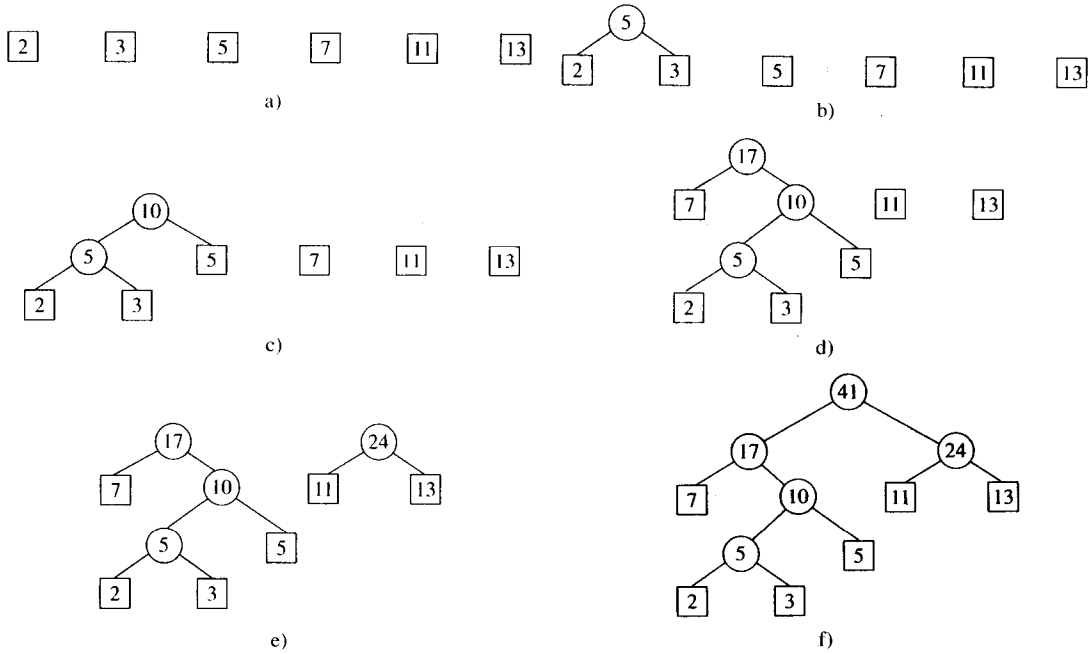


图3-20 最优二路归并序列

现在通过归纳法证明贪心法的正确性。显然对于$m = 2$，这个算法产生了一棵最优二路归并树。现在，假如对于$m-1$个表，这个算法产生了一棵最优二路归并树。对于包含m个表L_1，L_2，…，L_m的例子，可以先将两个表L_1和L_2合并，然后将这个算法应用到有$m-1$个表的实例上，将由这个算法产生的最优二路归并树记为T_2。在T_2中，有一个长度为$n_1 + n_2$的叶子结点，将这个结点分开生成两个孩子L_1和L_2，分别具有长度n_1和n_2，如图3-21所示，将这棵新生成的树标记为T_3，得到

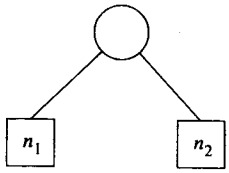


图3-21 一棵子树

$$W(T_3) = W(T_2) + n_1 + n_2 \tag{3-2}$$

我们称T_3是L_1，L_2，…，L_m的一棵最优二路归并树。假设不是这样，那么

$$W(T_3)>W(T)$$

这意味着

$$W(T_2)>W(T_1)$$

然而这是不可能的，由于根据归纳假设，T_2是$m-1$个表的最优二路归并树。

贪心法生成一棵最优二路归并树的时间复杂度

对于给定的m个数字n_1，n_2，…，n_m，可以构造一个小堆表示这些数，其中根结点的值比它孩子的值小。然后，将具有最小值的根结点移去后，重新生成一棵树能够在$O(\log n)$时间内完成，将一个结点插入到这个小堆中也可以在$O(\log n)$时间内完成。由于主循环执行了$n-1$次，那么生成一棵最优扩展二叉树所花费的总时间为$O(n \log n)$。

例3-8 赫夫曼（Huffman）编码

考虑一个电信中的问题，用0和1序列表示一组消息。因此，要发送一个消息，仅需要传送一串0与1序列。具有最优的带权外部路径长度（optimal weighted external path length）的扩展二叉树（extended binary tree）是生成最优编码集的应用，即这些信息的二进制串。假定有7段消息，它们的访问频率为2，3，5，8，13，15和18。为了最小化传送和解码的代价，可以用短的串表示使用频繁的消息，那么，可以先将2和3合并，然后是5和5等等，生成一棵最优扩展二叉树，产生的扩展二叉树如图3-22所示。

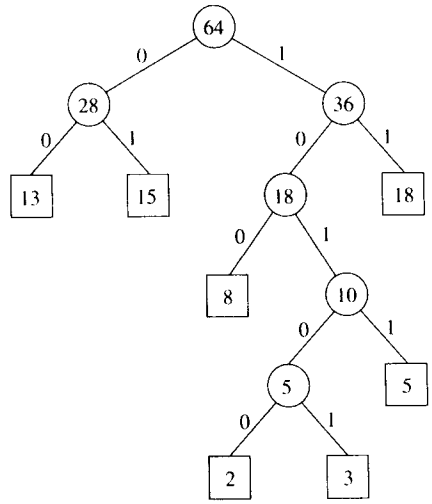


图3-22 赫夫曼编码树

相应于使用频率为2，3，5，8，13，15和18的消息编码分别为10100，10101，1011，100，00，01和11，频繁使用的消息编码较短。

3.5 用贪心法解决最小圈基问题

在这一节中，将介绍最小圈基问题（minimum cycle basis problem），同时将说明如何使用贪心法解决这个问题。

参见图3-23，在这个无向图中有三个圈，分别为{ab，bc，ca}，{ac，cd，da}和{ab，bc，cd，da}。

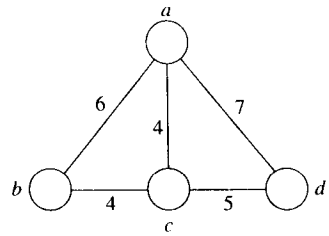


图3-23 由圈构成的图

通过适当的操作，可以将两个圈合并成另一个圈。这个操作是环和操作（ring sum operation），定义如下：令A和B是两个圈，那么$C = A \oplus B = (A \cup B) - (A \cap B)$是一个圈。对于上述情况的圈，令

$A_1 = \{ab, bc, ca\}$

$A_2 = \{ac, cd, da\}$

和 $A_3 = \{ab, bc, cd, da\}$

可以很容易证明

$A_3 = A_1 \oplus A_2$

$A_2 = A_1 \oplus A_3$

和 $A_1 = A_2 \oplus A_3$

这个例子说明{A_1，A_2}，{A_1，A_3}和{A_2，A_3}都可以认为是图3-23中图的圈基，因为对于每个圈基，用它可以生成图中所有的圈。正式的定义为：一个图的圈基是圈的集合，使得对圈基集合中的某些圈应用环和操作就可以生成图中任何一个圈。

假如每条边与一个权相关，圈的权是这个圈中所有边的权之和。圈基的权是这个圈基中所有圈的权之和。带权的圈基问题（weighted cycle basis problem）定义如下：给定一个图，找到这个图的最小圈基。对于图3-23，最小的圈基是{A_1，A_2}，它具有最小的权。

贪心法解决最小圈基问题基于如下思想：

(1) 我们能够确定最小圈基的规模，用K表示。

(2) 假设能够找到所有的圈（在一个图中找到所有的圈决不简单，但是这与该问题不相关，在这里不讨论这个技术。）根据它们的权将所有的圈排序成为一个非递减序列。

(3) 从圈的有序序列中，一个一个地将圈加入到圈基中。对于每一个加入的圈，检查它是

否已经存在于部分构成的圈基中的一些圈的线性组合。如果是，那么删除这个圈。

(4) 如果在圈基中有K个圈，那么停止。

上述贪心法是一个典型的贪心策略。假设想找到一组最小化某个参数的对象集，有时我们能确定这个集合的最小规模K，然后用贪心法将对象一个一个地加入这个集合中，直到集合的规模成为K为止。在下一节中，将举出另一个使用相同贪心法的例子。

现在回到原始问题。首先证明能够确定最小圈基的规模。我们不给出正式的证明，相反地，通过一个例子来非正式地解释这个概念。参见图3-24，假如构造这个图的一棵生成树，如图3-25a所示。这棵生成树没有圈，然而，如果将一条边加入这棵生成树中，那么会形成一个圈，如图3-25b所示。

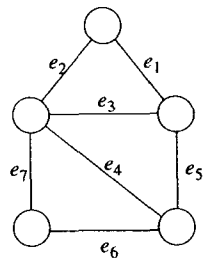


图3-24 说明最小圈基规模的图

实际上，正如图3-25所示，每加入一条边都会生成一个新的独立圈，独立的圈数目等于能够加入生成树中的边数目。由于生成树中边的数目是$|V|-1$，因此，能够加入的边总数等于

$$|E|-(|V|-1) = |E|-|V| + 1$$

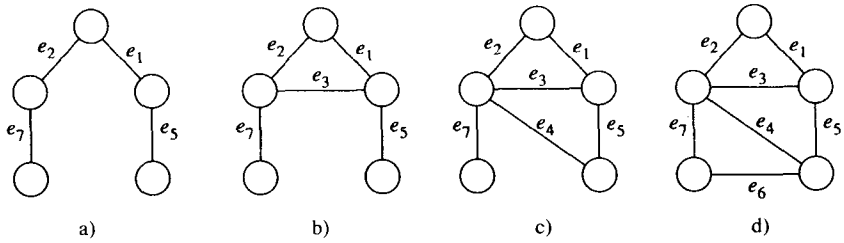


图3-25 生成树和圈的关系

我们已经证明了找到最小圈基规模的公式，剩下的唯一问题是判定一个圈是否圈基的线性组合。在这不用形式化的方法，仅通过一个例子来描述该技术。判定过程可以通过用一个邻接矩阵（incidence matrix）表示圈来完成。每行对应一个圈，每列对应一条边。独立还是不独立的判定可以通过除去每个包含两行高斯消除法（Gaussian elimination）的操作来完成，这些操作可以是环和（异或（exclusive-or））操作。现在通过一个例子来说明这个问题，参见图3-26。有三个圈$C_1 = \{e_1, e_2, e_3\}$，$C_2 = \{e_3, e_4, e_5\}$，$C_3 = \{e_1, e_2, e_5, e_4\}$，表示前两个圈的矩阵如下：

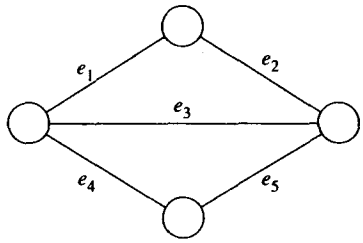


图3-26 说明圈的独立性检验的图

$$\begin{array}{c} \\ C_1 \\ C_2 \end{array}\begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & & \\ & & 1 & 1 & 1 \end{bmatrix} \end{array}$$

如果将C_3加入，矩阵变为如下形式：

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array}\begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & & \\ & & 1 & 1 & 1 \\ 1 & 1 & & 1 & 1 \end{bmatrix} \end{array}$$

作用于第1行与第3行的异或操作产生如下矩阵：

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array}\begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & & \\ & & 1 & 1 & 1 \\ & & 1 & 1 & 1 \end{bmatrix} \end{array}$$

作用于C_2和C_3的异或操作生成一个空行，说明C_3是C_1与C_2的线性组合。

现在通过考察图3-27来完成讨论，假定每条边的权为1，共有六个圈表示如下：

$C_1 = \{e_1, e_2, e_3\}$

$C_2 = \{e_3, e_5, e_4\}$

$C_3 = \{e_2, e_5, e_4, e_1\}$

$C_4 = \{e_8, e_7, e_6, e_5\}$

$C_5 = \{e_3, e_8, e_7, e_6, e_4\}$

$C_6 = \{e_2, e_8, e_7, e_6, e_4, e_1\}$

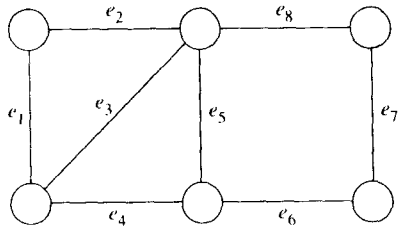


图3-27 说明找到最小圈基过程的图

对于这种情况，$|E| = 8$，$|V| = 6$，因此，$K = 8-6+1 = 3$。贪心法执行过程如下：

(1) C_1被加入。

(2) C_2被加入。

(3) C_3被加入然后被舍弃，由于发现C_3是C_1与C_2的线性组合。

(4) C_4被加入。这是由于圈基现在已经有三个圈，$K=3$就停止，因此最小圈基是$\{C_1, C_2, C_4\}$。

3.6 用贪心法解决2终端一对多问题

在超大规模集成电路（*VLSI*）的设计中，有一种线路布线问题（channel routing problem）。这样的问题有许多种情况，在第5章中将介绍一个能够用*A**算法方法（A* algorithm method）解决的线路布线问题。本节讨论的线路布线问题是一般布线问题的相当简化的版本，所谓的2终端一对多问题（2-terminal one to any problem）将会在下面清楚地解释。

参见图3-28，其中有几个终端做了标记，每个做了标记的上面一行终端都必须以一一对应的方式连接到做了标记的下面一行的终端，并且要求任何两条线都不能交叉。此外，所有的线或者垂直或者水平，每一条水平线对应于一条线路。

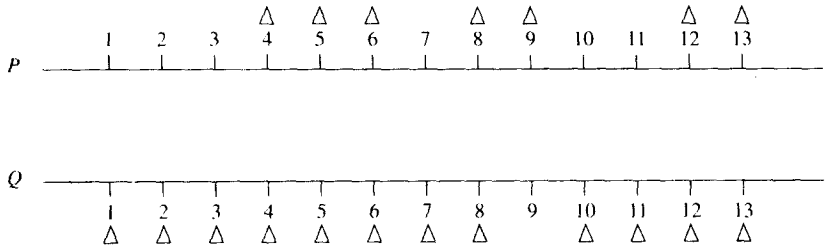


图3-28 2终端一对多问题举例

对于同一问题可能有好几种解决方案，在图3-29中，列出了图3-28中问题的两种可行解。可以看出图3-29a中的方案比图3-29 b中的解用更少的路径。

对于一个实际的线路布线问题，当然想将线路数目最小化，在这个简单的例子中，只是试图将解的密度（density of a solution）最小化。

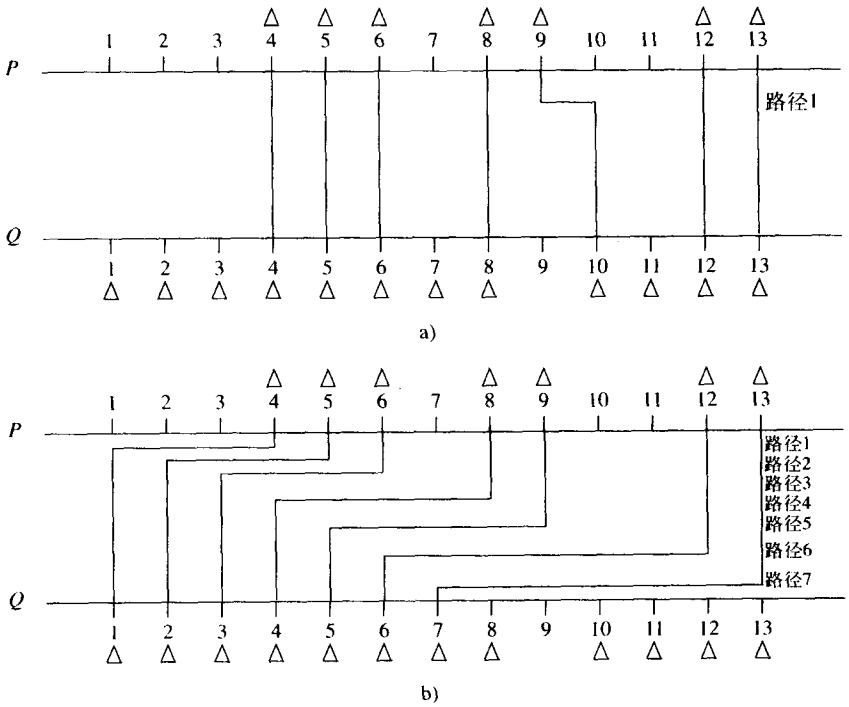


图3-29 图3-28中问题实例的两种可行解

为了解释密度的含义，重画图3-29中的两种解（如图3-30所示）。假设用一条垂线从左向右扫描，那么垂线要与方案中的线相交。在每个点上，解的局部密度（local density）是与垂线相交的线数目。

比如，如图3-30b所示，在终端7的垂直线与4条线相交，因此，在图3-30b中，解在终端7的局部密度是4。解的密度是最大的局部密度。很容易看出图3-30a、b中解的密度分别是1和4。

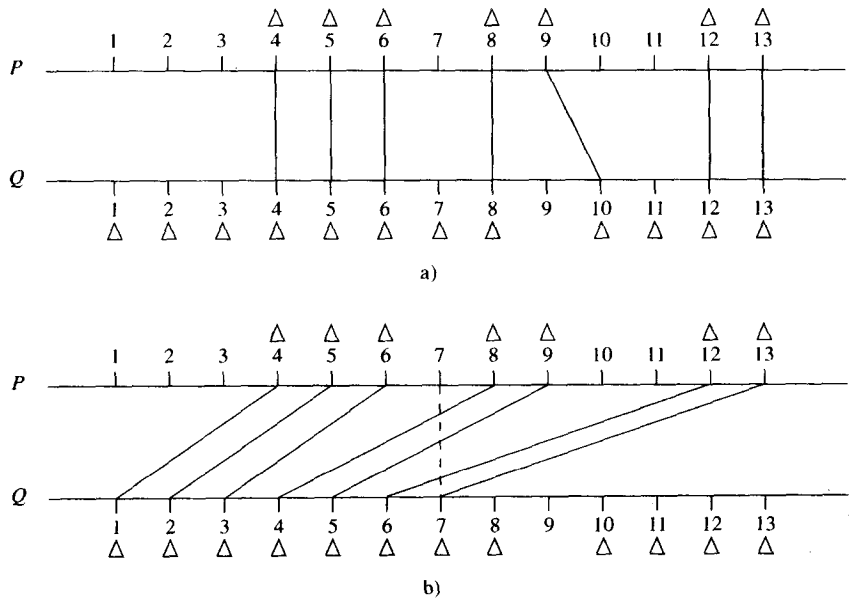


图3-30 重画图3-29

为什么使用密度作为要最小化的参数呢？答案很简单：这将使问题简化。如果用路径数目作为参数，那么问题将变得更加复杂而难以解决。另一方面，密度是路径数的下界，因此，可以用密度作为算法优良程度的标识。

可以使用上一节解决最小圈基问题的方法来找到具有最小密度的解，即要有一个方法来确定问题实例的最小密度。一旦最小密度被确定了，就可以使用贪心法找到具有最小密度的解。在这里不讨论如何确定最小密度，因为它非常复杂而且与所讨论的贪心法无关。我们讨论的重点是证明贪心法总是可以找到具有最小密度的解。

现在说明贪心法如何完成这个任务。给定一个问题实例，其中已经确定了它的最小密度，对于图3-28中的例子，最小密度是1。

令上面一行的终端标记为P_1，P_2，…，P_n，下面一行要连接的终端标记为Q_1，Q_2，…，Q_m，$m>n$。进一步假定P_i和Q_i都是从左向右标记的，也就是，如果$j>i$，那么$P_j(Q_j)$在$P_i(Q_i)$的右边。

已知最小密度d和上面的标记方法，贪心法的使用如下：

(1) 将P_1连接到Q_1。

(2) 在P_i连接到Q_j之后，考察P_{i+1}是否可以连接到Q_{i+1}，如果连接P_{i+1}到Q_{j+1}的线将密度增加到$d+1$，那么就尝试将P_{i+1}连接到Q_{j+2}。

(3) 重复以上过程直到所有的P_i都被连接。

下面通过将贪心法应用到图3-28中的例子来说明如何使用这种方法。首先确定$d=1$，由于这个信息，终端的连接方式如图3-31所示。注意到P_5不能连接到Q_2，Q_3和Q_4上，因为这样的连接会将密度增加到2，这超过最小密度。类似地，基于同样的原因也不能将P_6连接到Q_8。

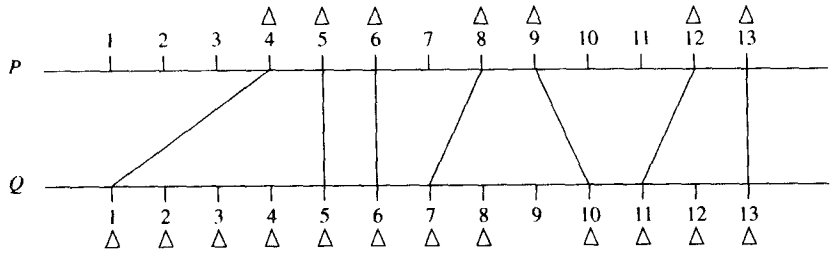


图3-31 用贪心法解决图3-28中的问题实例

注意到算法将不会产生使连接终端的线段相交的解，如图3-32所示。实际上，可以很容易地看出不会出现这样的连接，由于可以将这样的连接转化为更小的或相等密度的连接，如图3-33所示。

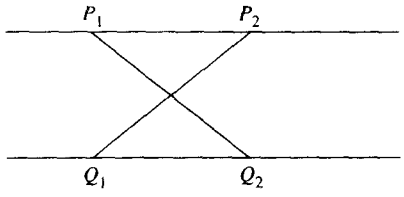


图3-32 交叉连接

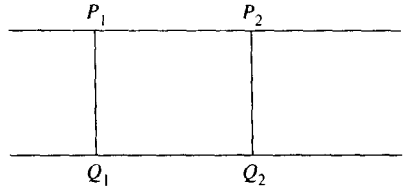


图3-33 从图3-32的交叉连接转换得到的连接

最后，将证明贪心法用来解决问题。由于假设最小密度为d，那么存在一个可行的解S_1，即$((P_1, Q_{j_1}), (P_2, Q_{j_2}), \cdots, (P_n, Q_{j_n}))$，它的密度为$d$。将证明这个解能够用贪心法转化为解$S_2$，即$((P_2, Q_{i_1}), (P_2, Q_{i_2}), \cdots, (P_n, Q_{i_n}))$，下面通过归纳法来证明。

假设在S_1中最初的k个连接能够转化为S_2中最初的k个连接，并且保证密度为d，然后证明这

个转化对$k = k + 1$也一样能完成。这个假设对$k = 1$显然是真的。如果对于k是真的，那么就得到一个部分解$((P_1, Q_{j_1}), (P_2, Q_{j_2}), \cdots, (P_k, Q_{j_k}))$，并且保证密度为$d$。考虑$P_{k+1}$，在$S_1$中$P_{k+1}$连接到了$Q_{j_{k+1}}$，假设在$Q_{j_{k+1}}$的左边有一个终端$Q_{j_{k+1}}$，在保证最小密度为$d$的条件下可以将它连接到$P_{k+1}$，那么将$P_{k+1}$连接到这个终端上，这就是用贪心法所获得的结果。假设不存在这样一个终端，那么用贪心法也可以将P_{k+1}连接到$Q_{j_{k+1}}$上。

上面的讨论证明了任何可行解都可以用贪心法转化为另一种解，因此贪心法可以解决该问题。

在用贪心法的最后一个例子中，决定要优化的参数最小值，然后用贪心法直接达到这个目标。在圈基问题中，将圈逐个加入直到获得了圈基的最小值。在2终端一对多问题中，首先计算最小密度，然后用贪心法将终端连接起来，任何时候算法都确保产生的密度不超过d。

3.7 用贪心法解决1螺旋多边形最小合作警卫问题

最小合作警卫问题（minimum cooperative guards problem）是艺术陈列馆问题的一种变形，其定义如下：给定一个多边形，它表示一个艺术陈列馆，要求安排最小数目的警卫，使得多边形中每一个点至少可以被一名警卫看到。例如，参见图3-34中的例子，对于这个多边形，需要警卫的最小数目为2，艺术陈列馆问题是一个NP难问题。

最小合作警卫问题对艺术陈列馆问题增加了更多的限制，注意到如果一名警卫安置在艺术陈列馆中不被别的警卫看到的地方，这是非常危险的。我们通过警卫的可视图（visibility graph）G来表示它们之间的关系。在G中，每一名警卫用一个顶点表示，当且仅当相应的两个警卫能看到对方时他们之间有一条边。除了找到能够监视整个多边形的最少警卫外，还要求这些警卫的可视图是连通的。换句话说，没有一名警卫是孤立的，在每一对警卫之间都有一条路径，称这样的问题为*最小合作警卫问题*。

再次考虑图3-34，相应于两名警卫的可视图显然是两个孤立顶点的集合。为了满足最小合作警卫问题的要求，必须再安排一名警卫，如图3-35所示。

可以再次证明最小合作警卫问题是NP难的。因此，在一般多边形上的任何多项式算法都不可能解决该最小合作警卫问题。但是可以证明用贪心法可以解决具有1螺旋多边形（1-spiral polygon）形式的此问题。

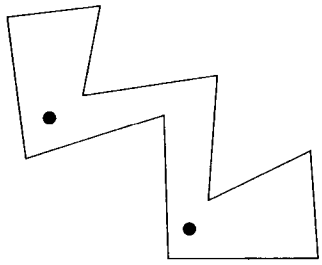


图3-34 艺术陈列馆问题的解决方案

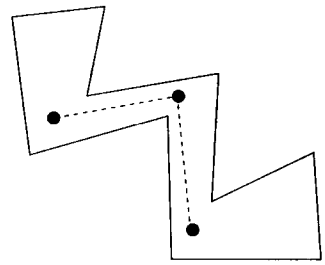


图3-35 对于图3-34中多边形的最小合作警卫问题的解决方案

在定义1螺旋多边形之前，首先定义反射（凸反射）链（reflex（convex）chain）。一个简单多边形的反射（凸反射）链表示为$RC(CC)$，它是这个多边形边的链，这个链满足链上的所有的顶点都是多边形的内部凸反射。我们保证反射链表示最大的反射链，即它不包含在其他任何反射链中。一个1螺旋多边形P是一个简单多边形，它的边界能够分割成一个反射链和一个凸反射链，图3-36描述了一个典型的1螺旋多边形。

逆时针遍历1螺旋多边形边界，称反射链的开始（结束）顶点为$v_s(v_e)$。已知v_s和v_e，可以定义两个区域，称为开始区域（starting region）和结束区域（ending region）。从$v_s(v_e)$开始，沿着反射链的第一条（最后一条）边画一条线，直到这条线与多边形的边界相交于$l_1(r_1)$。线段$\overline{v_s l_1}(\overline{v_e r_1})$和从$v_s(v_e)$出发的凸链的第一个（最后一个）部分形成一个区域，称为开始（结束）区域。图3-37举了两个例子，注意到开始和结束区域可能重叠，必须在开始区域和结束区域各安排一名警卫。

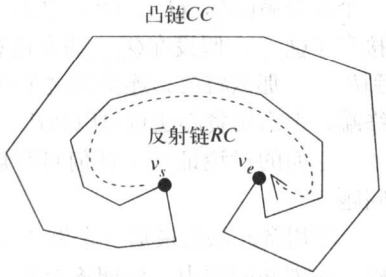


图3-36 典型的1螺旋多边形

贪心法解决1螺旋多边形最小合作警卫问题如下所示。首先把一名警卫放到l_1，然后，问这个警卫能看到多大面积？这个问题可以通过画一条RC的切线来解决，这条切线从l_1开始与CC相交于l_2，将一名警卫放到l_2，这是相当有道理的。注意到如果$\overline{l_1 l_2}$的左边没有警卫，那么生成的可视图将不连通。重复这个过程直至到达结束区域。图3-38显示一个典型的例子，在图中l_1，l_2，l_3，l_4和l_5点的警卫构成了最小合作警卫问题的最优解。

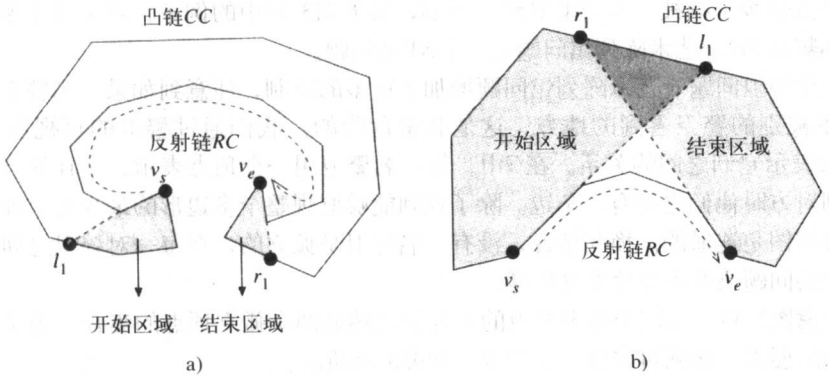


图3-37 1螺旋多边形的开始和结束区域

在给出正式算法前，必须定义一些新的术语。参见图3-39，令a是1螺旋多边形P内的一点，可以画出从顶点a出发的RC的两条切线。如果RC的外部完全位于从a出发的切线的右边（左边），就称它为关于RC的a的左（右）切线（left（right）tangent）。画一条关于RC的a的左（右）切线直到它与P的边界相交于点$b(b')$，那么$\overline{ab}(\overline{ab'})$称为关于$a$的左（右）支撑线段（left(right) supporting line segment）。在图3-39中说明。如果点a在开始（结束）区域，那么定义关于a的右（左）支撑线段为$\overline{av_s}(\overline{av_e})$。

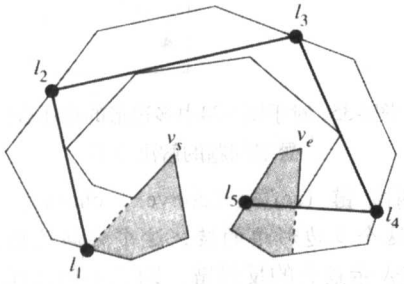


图3-38 1螺旋多边形的警卫$\{l_1, l_2, l_3, l_4, l_5\}$的集合

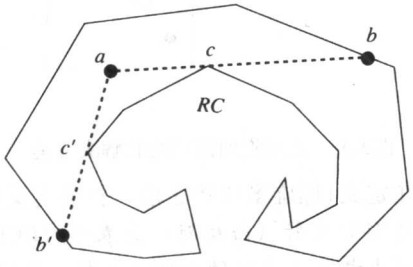


图3-39 关于a的左和右支撑线段

下面引入一个算法来解决1螺旋多边形最小合作警卫问题。

算法3-7 解决1螺旋多边形最小合作警卫问题的算法

输入：一个1螺旋多边形P。
输出：最小合作警卫问题解的点集。
步骤1. 找到P的反射链RC和凸链CC。
步骤2. 找到CC与分别从v_s和v_e出发沿着RC的第一条边和最后一条边的直线的交点l_1和r_1。
步骤3. 令$k = 1$。
While l_k不在结束区域中do
画出l_k关于RC的左切线直到它与CC相交于点l_{k+1}。（$\overline{l_k l_{k+1}}$是关于l_k的左支撑线段。）
令$k = k + 1$。
End While
步骤4. 返回$\{l_1, l_2, \cdots, l_k\}$。

现在来证明算法3-7的正确性。首先引入一些标号和术语，多边形P从点a到b按逆时针方向边界的一个子链记为$C[a, b]$。

假设A是相互配合的警卫的集合，它能看到整个简单多边形P，并且A的可视图是连通的，那么称A为关于P的最小合作警卫问题的一个可行解。注意到A不必是最小的。

令点x和y在P的凸链上，且$\overline{xy}$是一个支撑线段，$\overline{xy}$将P分为三个子多边形Q，Q_s和Q_e，Q是以$C[y, x]$和$\overline{xy}$为边界的子多边形，但是不包括x或y。Q_s和Q_e是另外两个分别包含v_s和v_e的子多边形，如图3-40所示。

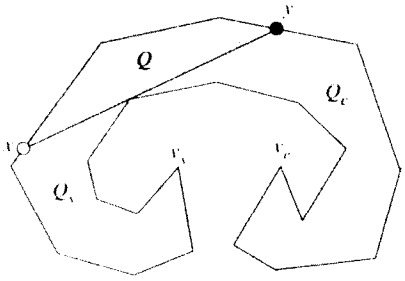


图3-40 支撑线段$\overline{xy}$和区域Q，Q_s和Q_e

假设A是一个关于P的最小合作警卫问题的可行解，那么将至少有A的一名警卫移到Q中，否则，A的可视图将不会连通，这是由于任何在Q_s和Q_e中的两名警卫都不会看到对方。

令P为一个1螺旋多边形，$\{l_1, l_2, \cdots, l_k\}$为算法3-7作用于$P$所产生的点集。令$L_i$是由$C[l_i, l_{i-1}]$和$\overline{l_{i-1}l_i}$为边界的区域，但是不包含$l_{i-1}$，其中$1 \leqslant i \leqslant k$。令$A$为关于$P$的最小合作警卫问题的任意可行解，那么至少存在$A$的一名警卫放在每个$L_i$中，其中$1 \leqslant i \leqslant k$。此外，很明显$L_i$不相交。令$N$是$A$的大小，那么$N \geqslant k$。这说明$k$是关于$P$的最小合作警卫问题的任何可行解中警卫的最小数目。

在证实了$\{l_1, l_2, \cdots, l_k\}$的最小性（minimality）之后，将证实$\{l_1, l_2, \cdots, l_k\}$的可视性（visibility），即，证明$P$中每个点都至少被$\{l_1, l_2, \cdots, l_k\}$中的一名警卫看到。显然，当且仅当一组警卫能够看到反射链的所有边时，他们才能看到整个1螺旋多边形。令支撑线段$\overline{l_i l_{i+1}}$与反射链相交于c_i，$1 \leqslant i \leqslant k-1$，并且$c_0 = v_s$，$c_k = c_e$。显然$l_i$可以看到$C[c_i, c_{i-1}]$，$1 \leqslant i \leqslant k-1$，因此，$\{l_1, l_2, \cdots, l_k\}$能看到整个反射链$C[v_s, v_e]$，能看到整个1螺旋多边形。此外，根据算法3-7本身，$\{l_1, l_2, \cdots, l_k\}$的可视图是连通的。这说明算法3-7的输出是最小合作警卫问题的一个可行解。由于警卫的数目是最少的，可以得出结论算法3-7产生了一个最优解。

对于算法3-7的时间复杂度分析，令1螺旋多边形P的顶点数目为n。第1步和第2步可以通过对P的边界的线性扫描在$O(n)$时间内完成，第3步可以进行对反射链逆时针和对凸链顺时针的线性扫描来找到所有需要的左支撑线段。步骤3也可以在$O(n)$时间内完成。因此，算法3-7是线性的。

3.8 实验结果

为了说明贪心法策略的能力，我们在IBM PC机上用C语言实现了找到最小生成树的Prim算

法，也实现了另外一个直接的算法，这个算法可以产生图$G=(V, E)$的所有可能的$(n-1)$条边的集合。如果这些边形成一个圈，那么它们将被忽略，反之这棵生成树的总长度被计算出来。

检查了所有生成树之后，最小生成树就被找到。这个直接的方法同样用C语言来完成。每一个数据集合相应于一个随机产生的图$G=(V, E)$。表3-3总结了实验结果，显然最小生成树问题不能够通过直接方法来解决，Prim算法是非常有效的。

表3-3 检测Prim算法有效性的实验结果

执行时间（秒）（平均20倍）							
$\|V\|$	$\|E\|$	直接方法	Prim	$\|V\|$	$\|E\|$	直接方法	Prim
10	10	0.305	0.014	50	200	—	1.209
10	20	11.464	0.016	50	250	—	1.264
10	30	17.629	0.022	60	120	—	1.648
15	30	9738.883	0.041	60	180	—	1.868
20	40	—	0.077	60	240	—	1.978
20	60	—	0.101	60	300	—	2.033
20	80	—	0.113	70	140	—	2.527
20	100	—	0.118	70	210	—	2.857
30	60	—	0.250	70	280	—	2.967
30	90	—	0.275	70	350	—	3.132
30	120	—	0.319	80	160	—	3.791
30	150	—	0.352	80	240	—	4.176
40	80	—	0.541	80	320	—	4.341
40	120	—	0.607	80	400	—	4.560
40	160	—	0.646	90	180	—	5.275
40	200	—	0.698	90	270	—	5.714
50	100	—	1.030	90	360	—	6.154
50	150	—	1.099	90	450	—	6.264

3.9 注释与参考

关于更多贪心法的讨论，可以参阅文献Horowitz and Sahni (1978)及Papadimitriou and Steiglitz (1982)。文献Korte and Louasz (1984)介绍了贪心法的结构框架。

解决最小生成树问题的Kruskal算法和Prim算法能分别在文献Kruskal (1956)和Prim (1957)中找到，Dijkstra算法最早出现于文献Dijkstra (1959)。

由贪心法产生出最优归并树最早出现于文献Huffman (1952)，文献Schwartz (1964)给出了一个算法来生成Huffman编码集，最小圈基问题产生于文献Horton (1987)。

贪心法解决2终端一对多线路分配问题由文献Atallah and Hambrusch (1986)提出，线性算法解决1螺旋多边形的最小合作警卫问题由文献Liaw and Lee (1994)提出。

3.10 进一步的阅读资料

尽管贪心法的概念很早以前就提出了，但是现在依然在出版很多有趣的文章讨论它。对于深入研究贪心法感兴趣的人来说，推荐以下的文章：Bein and Brucker (1986)；Bein，Brucker and Tamir (1985)；Blot，Fernandez de la Vega，Paschos and Saad (1995)；Coffman，Langston and Langston (1984)；Cunningham (1985)；El-Zahar and Rival (1985)；Faigle (1985)；Fernandez-Baca and Williams (1991)；Frieze，McDiarmid and Reed (1990)；Hoffman (1988)及

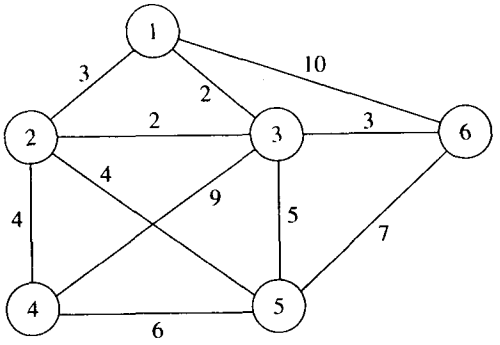
Rival and Zaguia (1987)。

当然贪心法也可以用作近似算法（见本书第9章）。下面的文章是很好的参考资料：Gonzalez and Lee (1987)；Levcopoulos and Lingas (1987)及Tarhio and Ukkonen (1988)。

对于一些新近出版的文章感兴趣，可以参阅以下文献：Akutsu，Miyano and Kuhara (2003)；Ando，Fujishige and Naitoh (1995)；Bekesi，Galambos，Pferschy and Woeginger (1997)；Bhagavathi，Grosch and Olariu (1994)；Cidon，Kutten，Mansour and Peleg (1995)；Coffman，Langston and Langston (1984)；Cowureur and Bresler (2000)；Csur and Kao (2001)；Erlebach and Jansen (1999)；Gorodkin，Lyngso and Stormo (2001)；Gudmundsson，Levcopoulos and Narasimhan (2002)；Hashimoto and Barrera (2003)；Iwamura(1993)；Jorma and Ukkonen (1988)；Krogh，Brown，Mian，Sjolander and Haussler (1994)；Maggs and Sitaraman (1999)；Petr (1996)；Slavik (1997)；Tarhio and Ukkonen (1986)；Tarhio and Ukkonen (1988)；Tomasz (1998)；Tsai，Tang and Chen (1994)；Tsai，Tang and Chen (1996)；and Zhang，Schwartz，Wagner and Miller (2003)。

习题

3.1 使用Kruskal算法找出下图的最小生成树。



3.2 使用Prim算法找出题3.1中图的最小生成树。

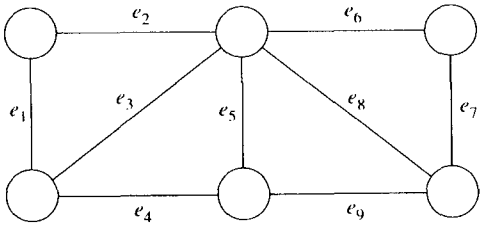
3.3 证明Dijkstra算法的正确性。

3.4 (a) 说明当图包含负权的边时，Dijkstra算法为什么将失效。

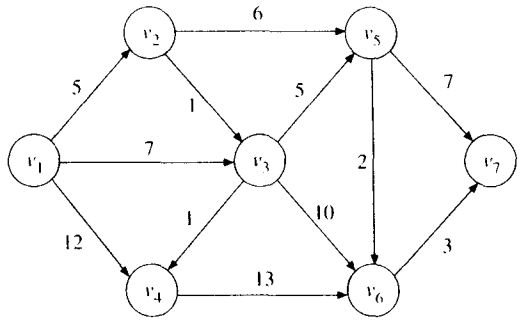
(b) 修改Dijkstra算法，使得它能计算在有负权边但是没有负权环的任意一幅图中，从源结点到每个结点的最短路径。

3.5 对于下列8个消息(M_1，M_2，…，M_8)得到一个最优的赫夫曼编码集，这8个消息的访问频率(q_1，q_2，…，q_8) = (5，10，2，6，3，7，12，14)，画出这个编码集的解码树。

3.6 找到下图的一个最小圈基。



3.7 用Dijkstra算法找到从v_1到所有其他结点的最短路径。



3.8 给出一个具有启发性的贪心法解决0/1背包问题，然后给出一个例子证明贪心法不是总能产生最优解。

3.9 完成Prim算法和Kruskal算法，做实验比较两种算法的性能。

3.10 阅读文献Papadimitriou and Steiglitz(1982)的12.4节，了解矩阵胚与贪心法间的关系。

3.11 背包问题定义如下：

已知正整数P_1，P_2，…，P_n，W_1，W_2，…，W_n和M，寻找在$\sum_{i=1}^{n} W_i X_i \leqslant M$条件下，$X_1$，$X_2$，…，$X_n$，$0 \leqslant X_i \leqslant 1$，使得$\sum_{i=1}^{n} P_i X_i$最大化。

给出一个贪心法找到背包问题的一个最优方案，并且证明它的正确性。

3.12 考虑在一台机器上调度n个作业的问题。设计一种算法找到一种调度方案使得它的平均完成时间是最小的，并证明算法的正确性。

3.13 Sollin算法最早在文献Boruvka(1926)中提出，是为了在连通图G上找到一棵最小生成树。初始时，G中的每个顶点被视为一个单结点的树，不选择任何边。在每一步，为每棵树选择一条最小权的边e，使得e只有一个顶点在T中。如果必要的话，除去所选边的备份。当只得到一棵树或者所有的边都被选中了，那么终止算法。证明算法的正确性并且求出算法的最大步数。

第4章 分治策略

分治策略（divide-and-conquer strategy）是一种设计高效算法有力的范型。该方法首先将问题划分为两个较小的子问题，每个子问题除了输入规模较小以外，其他都与原问题是一样的。然后，分别解决这两个子问题，最后将子问题的解合并成为原问题的最终解。

关于分治法非常重要的一点是将原问题划分为两个子问题，且子问题与原问题是相同的。因此，这两个子问题能够再次使用分治法解决。或者换言之，这两个子问题可以递归求解。

考虑一个简单问题：找出具有n个元素集合S中的最大元素。分治法通过将输入分成两个集合解决该问题，每个子集含有$n/2$个元素，称这两个子集为S_1和S_2。现在找出集合S_1和S_2各自的最大元素，将集合S_i的最大元素表示为$X_i(i = 1, 2)$，那么集合S的最大元素可通过比较X_1和X_2来确定，大者就是集合S的最大元素。

在上面的讨论中，不经意地提及要找出最大的X_i。但是，如何找呢？可以再次使用分治策略。也就是，划分集合S_i为两个子集合，找出子集合的大者，然后合并结果。

一般地，分治算法由三步组成。

步骤1：如果问题规模较小，那么使用某种方法直接解决它；否则，划分原问题成两个子问题，最好以相同的规模划分。

步骤2：对子问题使用分治算法递归地解决这两个子问题。

步骤3：合并子问题的解成为原问题的解。

步骤2的递归产生越来越小的子问题。最终，这些子问题将小到可以容易地直接解决。

下面通过解决找集合中最大元素的例子说明分治法的含义。假如有八个数字：29，14，15，1，6，10，32和12。分治策略找出这八个数字中的最大者，如图4-1所示。

如图4-1所示，八个数字首先划分成两个子集合，每个子集合进一步划分成由两个元素组成的子集合。当子集合仅有两个数字时，不必再划分。这两个数字的简单比较可确定最大者。

在找出四个最大者之后，合并过程开始。这样，29与15比较，10与32比较。最终，通过32与29的比较找出32是最大者。

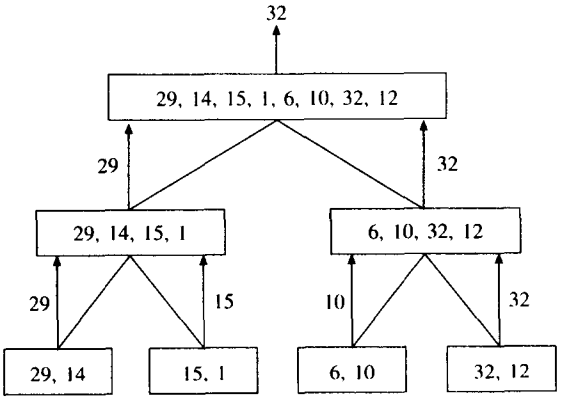


图4-1 找出八个数字中最大者的分治策略

一般，分治算法的复杂度$T(n)$由下面的公式确定：

$$T(n)=\begin{cases}2T(n/2)+S(n)+M(n) & n\geqslant c\\ b & n<c\end{cases}$$

其中$S(n)$表示划分问题成为两个子问题所需的时间步数，$M(n)$表示合并两个子问题所需的时间步数，b是一个常数。找出最大者的问题，划分和合并都花费常数级步数。因此，可以得到

$$T(n)=\begin{cases}2T(n/2)+1 & n>2\\ 1 & n=2\end{cases}$$

假设$n=2^k$，可以得到

$$
\begin{aligned}
T(n) &= 2T(n/2) + 1 \\
&= 2(2T(n/4) + 1) + 1 \\
&\quad \vdots \\
&= 2^{k-1}T(2) + \sum_{j=0}^{k-2} 2^j \\
&= 2^{k-1} + 2^{k-1} - 1 \\
&= 2^k - 1 \\
&= n - 1
\end{aligned}
$$

读者可能注意到应用分治策略找出最大元素只是为了说明其含义。显然在此例中，分治策略并不非常有效，因为直接地扫描这n个数字并做$n-1$次比较找出最大者与分治策略具有同样的效率。但是，在后面各节中将说明在许多情况下，特别是在几何问题上，分治策略确实是非常好的方法。

4.1 求2维极大点问题

在2维空间中，如果$x_1>x_2$且$y_1>y_2$，那么称点(x_1, y_1)支配点(x_2, y_2)。如果一个点没有其他点支配，那么称其为极大者（maxima）。已知有n个点的集合，求极大点问题（maxima finding problem）是指找出在n个点中的所有极大点。例如，图4-2中的圆圈点就是极大点。

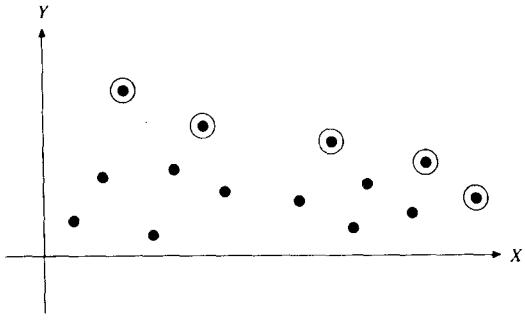


图4-2 极大点

解决求极大点问题的直接方法是比较每一对点，该方法需要$O(n^2)$次点的比较。正如下面将要证明的，如果使用分治策略，那么可以在$O(n \log n)$步数内解决该问题，这的确是很好的改进。

如果使用分治策略，那么首先找出垂直于x轴的中位线L，它划分整个点集为两个子集，如图4-3所示。分别用S_L和S_R表示中线L的左边点集合和右边点集合。

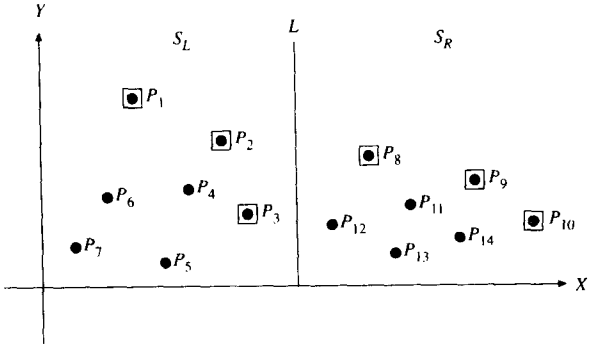


图4-3 S_L和S_R的极大点

下一步，分别找出S_L和S_R的极大点。如图4-3所示，P_1、P_2和P_3是S_L中的极大点，而P_8、P_9和P_{10}是S_R中的极大点。

合并过程相当简单。由于在S_R中点的x值总是大于S_L中每个点的x值，所以S_L中的点是极大点当且仅当它的y值不小于S_R中极大点的y值。例如图4-3所示的情形，因为P_3被S_R中的P_8所支配，所以它被丢弃。因此，S中的极大点集合是P_1、P_2、P_8、P_9和P_{10}。

基于上面的讨论，现在对解决求2维极大点问题的分治算法总结如下：

算法4-1　在平面中找出极大点的分治方法

输入：n个平面点的集合。

输出：S的极大点。

步骤1. 如果S只含一个点，那么输出该点为极大点；否则，找一条垂直于x轴的直线L，它划分点集合为两个子集合S_L和S_R，每个子集含有$n/2$个点。

步骤2. 递归地找出S_L和S_R的极大点。

步骤3. 将S_L和S_R的极大点投影到直线L上，并根据它们的y值排序这些点。对投影进行线性扫描，如果S_L中的极大点的y值小于S_R中某个极大点的y值，那么舍弃它。

该算法的时间复杂度依赖于下面步骤的时间复杂度：

(1) 在步骤1中，有一个找出n个数的中位数操作，后面将证明该操作需要在$O(n)$步数内完成。

(2) 在步骤2中，有两个子问题，均有$n/2$个点。

(3) 在步骤3中，依据y值对n个点排序后，投影和线性扫描需要在$O(n \log n)$步数内完成。

令$T(n)$表示在平面内对找出n个点中极大点的分治算法时间复杂度，那么

$$T(n)=\begin{cases}2T(n/2)+O(n)+O(n\log n) & n>1\\ 1 & n=1\end{cases}$$

令$n=2^k$，易证得

$$T(n)=O(n\log n)+O(n\log^2 n)=O(n\log^2 n)$$

因此，得到一个$O(n\log^2 n)$时间的算法。读者记得，如果穷尽地检验每对点的直接方法，需要$O(n^2)$步。

注意解决该问题的分治策略是由合并阶段的排序决定的。因为排序是一劳永逸的，所以我们并未做太有效的工作。也就是，应该先执行排序。如果这么做，那么合并的复杂度是$O(n)$，所需的总时间步数是

$$O(n\log n)+T(n)$$

其中

$$T(n)=\begin{cases}2T(n/2)+O(n)+O(n) & n>1\\ 1 & n=1\end{cases}$$

$T(n)$很容易确定是$O(n\log n)$。因此，具有预排序的分治策略找出极大点总的时间复杂度是$O(n\log n)$。换句话说，我们得到了一个更有效的算法。

4.2 最近点对问题

最近点对问题（closest pair problem）定义如下：已知n个点的集合，找出最接近的一对点。1维最近点对问题可在$O(n\log n)$步数内解决，这通过对n个点排序，并执行线性扫描，对排好序的队列考查每对相邻点间的距离，可以确定最近点对。

在2维空间里，可用分治策略设计有效算法来找出最近点对。如同解决求极大点问题，首

先划分集合S为S_L和S_R，使得S_L中的每个点位于S_R中的每个点的左边，并且S_L中的点与S_R中的点数相同。也就是，找一条垂直于x轴的纵向线L，使集合S划分成两个相同规模大小的子集合。分别在S_l和S_R中解决最近点对问题，得到d_L和d_R，分别表示在S_L和S_R中最近点对的距离。令$d = \min(d_L, d_R)$。如果S中的最近点对(P_a, P_b)是一个点在S_L中，另一个点在S_R中，那么P_a和P_b一定是在以直线L为中心的一个片隙（slab）内，它是以线$L-d$和$L+d$为界，如图4-4所示。

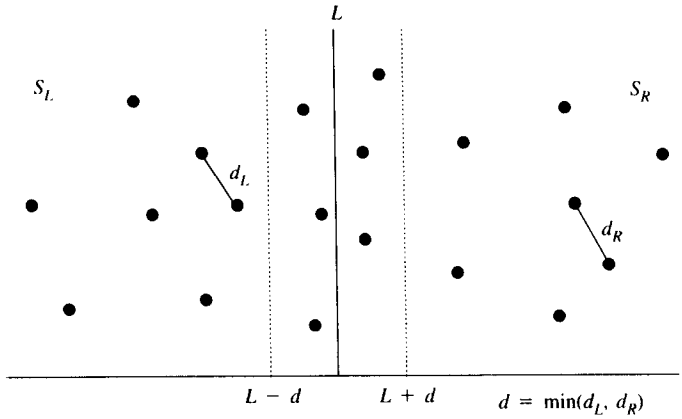


图4-4 最近点对

上述的讨论表明在合并步骤内，只考查在片隙内的点。尽管平均看来，在片隙内点的数量不可能太多，在最坏情况下片隙内有n个点。这样，在片隙内找出最近点对的蛮力方法需要计算$n^2/4$个距离和比较。这种合并步骤对分治算法并不有利。幸运的是，正如下面要证明的，合并步骤能在$O(n)$时间内完成。

如果在S_l中的点P与在S_R中的点Q成为最近点对，那么点P和Q间的距离必定小于d。因此，不必考虑远离点P的点。如图4-5所示，只需检验图4-5中的阴影区域。如果点P正好在直线L上，那么阴影区域最大。即使这样，阴影区域也只包含在$d \times 2d$的长方形A中，如图4-6所示。这样，只需考查在长方形A中的点。

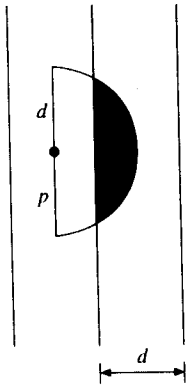


图4-5 考查分治最近点对算法的合并过程的有限区域

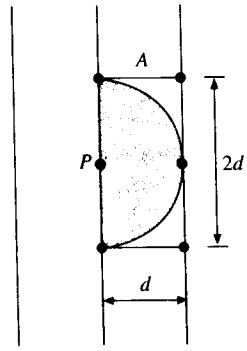


图4-6 长方形A含有可能的点P的最近邻点

接下来的问题是：在长方形A中有多少点？由于已知在S_L和S_R中的每对点间的距离都大于或等于d，那么至多有6个点在该长方形中，如图4-6所示。基于此观察，我们知道对于片隙中每个点P，在合并步骤中，只需检验片隙另一半有限数量的点。不失一般性，假定点P在片隙

的左半部分，将点P的y值表示为y_p。对于点P，只需要检验片隙另一部分的点，这些点的y值在$y_p + d$和$y_p - d$内。正如前面的讨论，最多有6个这样的点。

如前所述，在使用分治算法前，需要对n个点依据其y值排序。在预排序后，合并步骤将花费$O(n)$步数。

下面是解决2维最近点对问题的分治算法。

算法4-2 解决2维最近点对问题的分治算法

输入：平面上n个点的集合S。

输出：两个最近点间的距离。

步骤1. 根据点的y值和x值对S中的点排序。

步骤2. 如果S中只含一个点，那么，返回∞作为距离。

步骤3. 找出垂直于x轴的中位线L，将集合S划分成相同大小的子集S_L和S_R。子集S_L中的点位于直线L的左边，子集S_R中的点位于直线L的右边。

步骤4. 将步骤2和步骤3递归地应用于解决子集S_L和S_R的最近点对问题。令$d_L(d_R)$表示在$S_L(S_R)$中最近点对间的距离，并令$d = \min(d_L, d_R)$。

步骤5. 将以$L-d$和$L + d$为界，将片隙内所有的点投影到直线L上。对于在边界为$L-d$和L片隙内的点P，将它的y值表示为y_p。对于每个这样的点，找出所有在另一半以L和$L + d$为界的点，它们的y值都落在$y_p + d$和$y_p - d$内。如果点P与在另一半片隙内的点的距离d'小于d，令$d = d'$。最后的y值就是答案。

整个算法的时间复杂度等于$O(n \log n) + T(n)$，并且

$$T(n) = 2T(n/2) + S(n) + M(n)$$

$S(n)$和$M(n)$分别是步骤3中划分步骤和步骤4中合并步骤的时间复杂度。划分步骤花费$O(n)$步数，因为点已按x值排序。合并步骤首先必须找出在$L-d$与$L + d$间的点，因为已按x值排序，这在$O(n)$步数内完成。对于每个点，至多需要检验另外12个点，每半个片隙6个点。这样，线性扫描花费$O(n)$步数。换句话说，合并步骤是$O(n)$时间复杂度。

这样，

$$T(n) = \begin{cases} 2T(n/2) + O(n) + O(n) & n > 1 \\ 1 & n = 1 \end{cases}$$

很容易证明

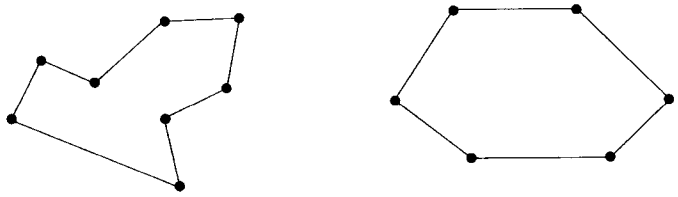
$$T(n) = O(n \log n)$$

总的时间复杂度是$O(n \log n)$（预排序的时间复杂度）$+ O(n \log n) = O(n \log n)$。

4.3 凸包问题

凸包问题已在第2章提到。在本节将说明凸包问题可用分治策略漂亮地解决。

凸多边形（convex polygon）具有在多边形内连接任意两点的线段一定还在该多边形内的性质。图4-7a显示一个非凸多边形，图4-7b显示了一个凸多边形。



a) b)

图4-7 凹多边形和凸多边形

平面点集的凸包定义为包含所有点的最小凸多边形。例如，图4-8显示了一个典型的凸包。

为了找出凸包，可以使用分治策略。参见图4-9，平面点集已被垂直于x轴的直线分成两个子集S_L和S_R。子集S_L和S_R的凸包已构成，分别表示为Hull(S_L)和Hull(S_R)。为了合并Hull(S_L)和Hull(S_R)成为一个凸包，可以使用Graham扫描（Graham's scan）。

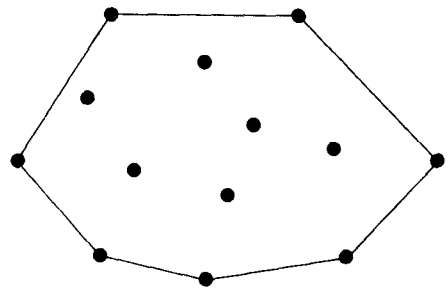


图4-8 一个凸包

为了进行Graham扫描，选择一个内部点。将该点作为起始点，这样其他每个点相对于该点都形成极角（polar angle）。将所有的点按照极角排序，Graham扫描方法逐个检查这些点，删去产生折回角（reflexive angle）的点，如图4-10所示，h，f，e和i将被删去，剩余的点为凸包顶点。

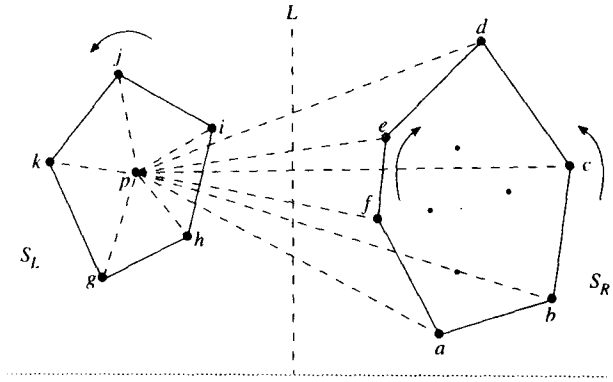


图4-9 构造凸包的分治策略

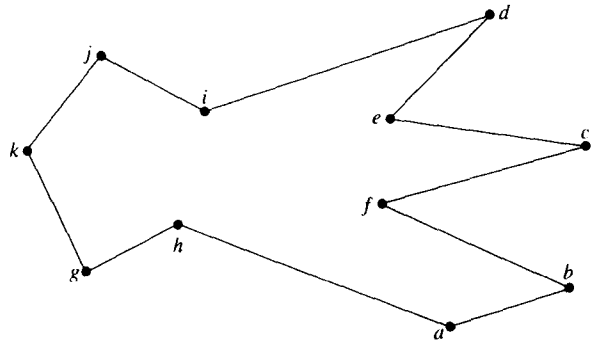


图4-10 Graham扫描

在子集Hull(S_L)和Hull(S_R)构造后，为了进行Graham扫描，可以选择Hull(S_L)的内部点P。显然，在该凸包内的点都不必检查。因为从点P处看，Hull(S_R)位于一个楔内，它的顶角（apex angle）等于或小于π。该楔由Hull(S_R)中两个顶点u和v定义，可在线性时间内由下面的过程找出：构造一个通过点P的水平线，如果该线与Hull(S_R)相交，那么Hull(S_R)在由Hull(S_R)中的两顶点所确定的楔内，两顶点中最大的极角小于$\pi/2$，最小的极角大于$3\pi/2$。如果通过点P的水平线不与Hull(S_R)相交，那么楔由关于P的最大和最小极角对着的点所确定。在图4-9中，它们是a和d。这样，相对于S_L的内部点P有三个递增极角的点序列。它们是

(1) g, h, i, j, k

(2) a, b, c, d

和 (3) f, e。

可以合并这三个点序列为一个序列。在所举的例子中，合并的序列是

g, h, a, b, f, c, e, d, i, j, k

Graham扫描可以应用于此序列以删去不在凸包中的点。对于如图4-9所示的情况，结果的凸包如图4-11所示。

构造凸包的分治算法如下。

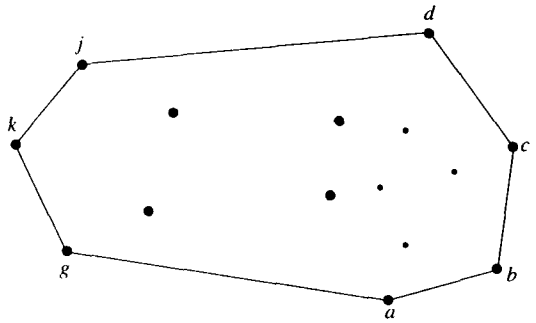


图4-11 图4-9中点的凸包

算法4-3 基于分治策略构造凸包的算法

输入：平面上的点集合S。

输出：集合S的凸包。

步骤1. 如果集合S有不超过5个点，那么使用穷尽搜索找出凸包并返回。

步骤2. 找出垂直于x轴的中线L，将集合S划分成相同大小的子集S_L和S_R，且S_L位于S_R的左边。

步骤3. 递归地对S_L和S_R构造凸包，分别以Hull(SL)和Hull(S_R)表示各自的凸包。

步骤4. 找出S_L的一个内部点P，确定Hull(S_R)的两个点v_1和v_2，划分Hull(S_R)的顶点成两个相对于点P渐增极角的顶点序列。不失一般性，假定v_1的y值大于v_2，那么形成如下三个序列：

(a) 序列1：Hull(S_L)的所有凸包顶点都沿反时针方向。

(b) 序列2：Hull(S_R)中从v_2到v_1的凸包顶点沿反时针方向。

(c) 序列3：Hull(S_R)中从v_2到v_1的凸包顶点沿顺时针方向。

合并这三种序列，并进行Graham扫描。删去折回的点，剩余的点形成凸包。

上面算法的时间复杂度基本上由划分过程和合并过程决定。因为划分过程是一个中值查找过程，所以它的时间复杂度是$O(n)$。因为对于内部点的检索、极值点的确定、合并以及Graham扫描都花费$O(n)$步，所以合并过程的时间复杂度是$O(n)$，这样，$T(n) = 2T(n/2) + O(n) = O(n \log n)$。

4.4 用分治策略构造Voronoi图

Voronoi图如同第3章介绍的最小生成树，是一种非常有趣的数据结构。该数据结构可用于存储有关平面中点的最近邻点的重要信息。

先考虑图4-12中所示的两个点的情况。在图4-12中，直线L_{12}是连接P_1和P_2的垂直平分线(perpendicular bisector)，L_{12}就是P_1和P_2的Voronoi图。位于L_{12}左(右)半平面的每个点X在P_1与P_2间，X的最近邻点就是$P_1(P_2)$。在此意义上，已知任意点X，为找出X的最近邻点，不必计算X与P_1和P_2间的距离，只需确定位于L_{12}的哪一边。这可通过将X的坐标代入L_{12}来做。依赖于此代换结果的符号，可确定X位于何处。

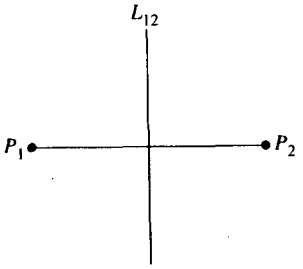


图4-12 两个点的Voronoi图

图4-13中每条L_{ij}是连接P_i与P_j间直线的垂直平分线。三个超平面L_{12}，L_{13}和L_{23}组成点P_1、P_2和P_3的Voronoi图。如果某点位于某个R_i中，那么该点在P_1、P_2和P_3中的最近邻点就是P_i。

为定义平面点集的Voronoi图，首先定义Voronoi多边形。

定义 已知在n个点集合S中的两个点P_i和P_j，令$H(P_i, P_j)$表示包含较接近P_i的点集合的半平面，相对于P_i的Voronoi多边形是不超过$n-1$条边的凸多边形区域，用$V(i)$ =

$\bigcap_{i \neq j} H(P_i, P_j)$表示。

一个Voronoi多边形如图4-14所示。已知n个点的集合，Voronoi图包含这些点的所有Voronoi多边形。Voronoi图中的点称为Voronoi点，它的线段称为Voronoi边。（名字Voronoi起源于一位著名的俄罗斯数学家。）一个具有6个点的Voronoi图表示在图4-15中。

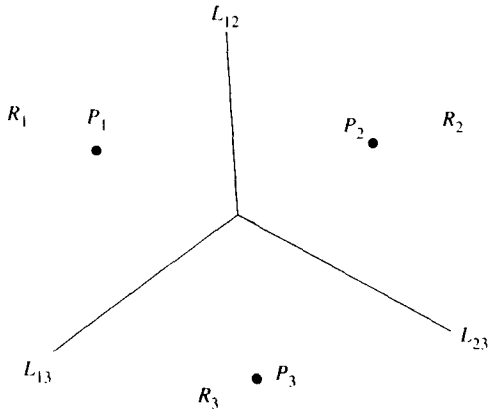


图4-13 三个点的Voronoi图

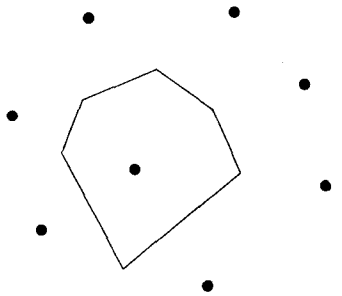


图4-14 Voronoi多边形

Voronoi图的直线对偶称为Delaunay三角剖分（Delaunay triangulation），为纪念著名的法国数学家。在Delaunay三角剖分中有连接P_i和P_j的线段，当且仅当P_i和P_j的Voronoi多边形有共同边。例如，图4-15，它的Delaunay三角剖分如图4-16所示。

Voronoi图有很多应用。例如，可以从Voronoi图中提取信息来解决称为所有最近点对的问题，最小生成树也可从Voronoi图中求解。

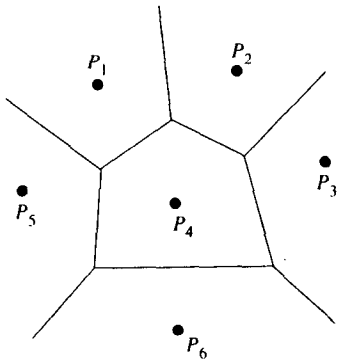


图4-15 6个点的Voronoi图

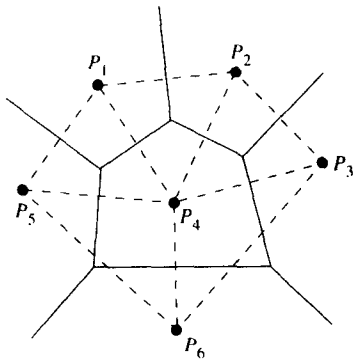


图4-16 Delaunay三角剖分

Voronoi图可以用算法4-4的分治方法形式构造。

算法4-4 构造Voronoi图的分治策略算法

输入：n个平面点集合S。

输出：集合S的Voronoi图。

步骤1. 如果集合S只有1个点，那么返回。

步骤2. 找出垂直于x轴的中线L，将S划分成相同大小的S_L和S_R，使得$S_L(S_R)$位于L的左（右）边。

步骤3. 递归地构造S_L和S_R的Voronoi图，分别以$VD(S_L)$和$VD(S_R)$表示各自的Voronoi图。

步骤4. 构造分段直线超平面HP，它是同时接近S_L中一点和S_R中一点的点轨迹。删除位于HP右边的$VD(S_L)$所有的线段和位于HP左边$VD(S_R)$所有的线段。剩下的图就是S的Voronoi图。

通过考虑图4-17来理解该算法。为了合并$VD(S_L)$和$VD(S_R)$，观察到，只有$VD(S_L)$和$VD(S_R)$接近L的部分会互相干扰。而远离L的点不受合并步骤的影响，并保持原样。

合并$VD(S_L)$和$VD(S_R)$最重要的步骤是构建分段直线超平面HP。此超平面有下述性质：如果点P在HP的左（右）边内，那么点P的最近邻点一定是$S_L(S_R)$中的点。图4-17中$VD(S_L)$和$VD(S_R)$的HP显示在图4-18中。

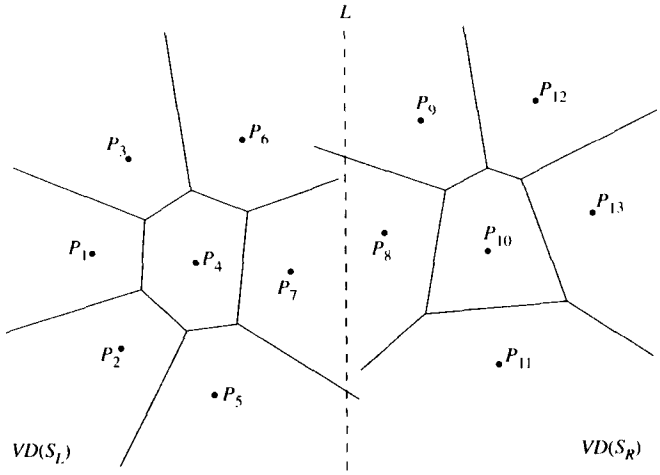


图4-17 步骤2后的两个Voronoi图

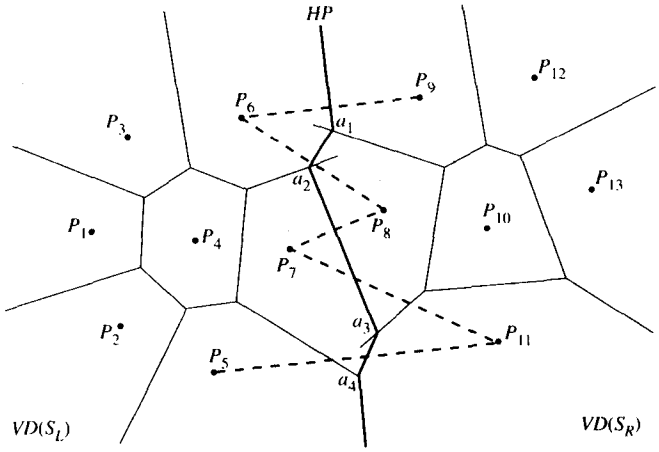


图4-18 图4-17中点集的分段直线超平面

在删除所有$VD(S_L)$到HP右边与所有$VD(S_R)$到HP左边的直线部分后，得到结果Voronoi图，如图4-19所示。

剩下的问题是如何有效地找出HP。构造HP的第一步是找出S_L和S_R的两个顶部元素，它们是P_6和P_9。现在构造直线$\overline{P_6P_9}$的垂直平分线，这是HP的开始线段。如图4-18所示，HP与$\overline{P_9P_8}$的平分线相交于a_1点。这样，知道轨迹将比P_9更接近P_8。换句话说，下一线段将是$\overline{P_6P_8}$的平分线。

继续向下移，HP将与$VD(S_L)$相交于点a_1。$VD(S_L)$与HP相交的线段是$\overline{P_6P_7}$的平分线。这样，新的线段将是$\overline{P_7P_8}$的平分线。

通过另一个例子来说明构造Voronoi图的过程，考虑图4-20中的6个点。

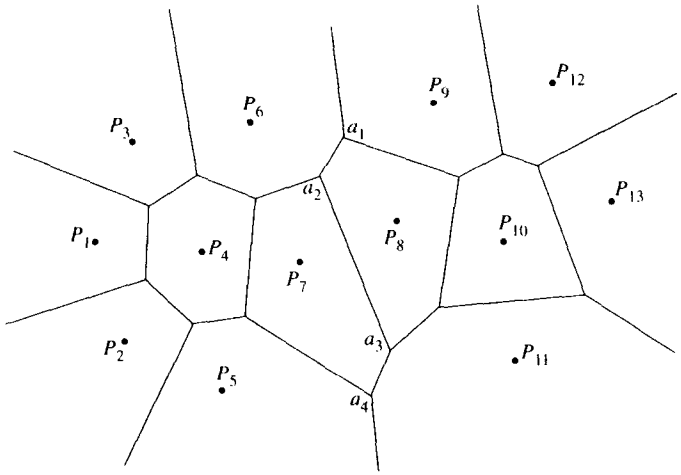


图4-19 图4-17中点集的Voronoi图

图4-20显示了分别对$\{P_1, P_2, P_3\}$和$\{P_4, P_5, P_6\}$构造的两个Voronoi图，都用射线标记。例如，b_{13}是线$\overline{P_1P_3}$的垂直平分线。构造Voronoi图的过程需要构造凸包。两个凸包显示在图4-20中，它们都是三角形。在构造两个凸包后，推断线段$\overline{P_1P_5}$和$\overline{P_2P_6}$是应添加到两个凸包的两条线段。这样，合并步骤从找出$\overline{P_1P_5}$的垂直平分线开始，如图4-21所示。

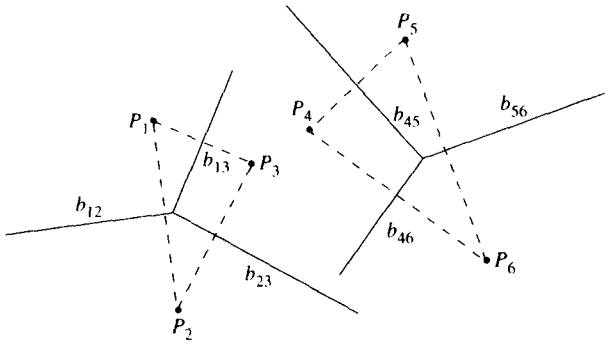


图4-20 说明构造Voronoi图的另一种情形

整个合并过程由下面步骤构成：

(1) $\overline{P_1P_5}$的垂直平分线与线b_{45}相交于b点。现在找出$\overline{P_1P_4}$的垂直平分线，用b_{14}表示。

(2) b_{14}与b_{13}相交于c点，下一条$\overline{P_3P_4}$的垂直平分线将是b_{34}。

(3) b_{34}与b_{46}相交于d点，下一条$\overline{P_3P_6}$的垂直平分线将是b_{36}。

(4) b_{36}与b_{23}相交于e点，下一条$\overline{P_2P_6}$的垂直平分线将是b_{26}。

由于P_2和P_6是两个凸包中最低的点，射线b_{26}无限延伸，就找出分段直线超平面HP。这样，HP由$\overline{ab}$、$\overline{bc}$、$\overline{cd}$、$\overline{de}$和$\overline{ef}$所确定。如果某点落入HP的右（左）边，它的最近邻点一定在$\{P_4, P_5, P_6\}$（$\{P_1, P_2, P_3\}$）中。

图4-21显示了所有构成分段超平面的这些垂直平分线，用虚线标记。这些线段是$\overline{P_5P_1}$、$\overline{P_1P_4}$、$\overline{P_4P_3}$、$\overline{P_3P_6}$和$\overline{P_6P_2}$。

在合并步骤中，最后一步是删除所有向HP右（左）延伸的$VD(S_L)$（$VD(S_R)$）。最终的

Voronoi图如图4-22所示。

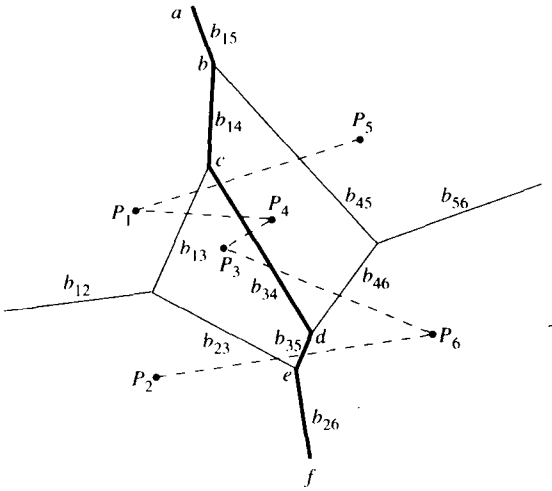


图4-21 构造Voronoi图的合并步骤

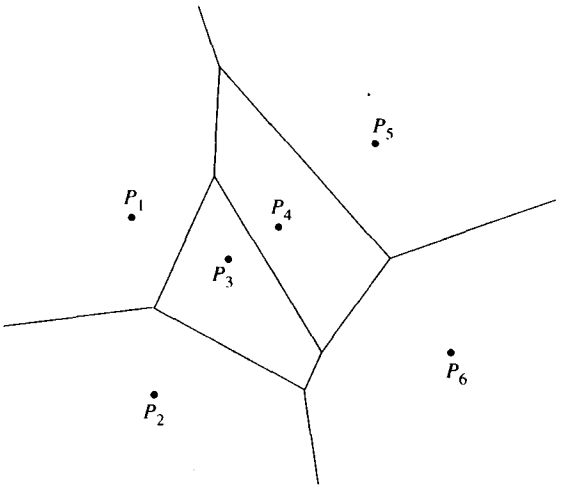


图4-22 最终的Voronoi图

现在介绍合并两个Voronoi图的算法。

算法4-5 将两个Voronoi图合并成一个Voronoi图的算法

输入：(a) S_L和S_R，S_L和S_R是被垂线L所划分。

(b) $VD(S_L)$和$VD(S_R)$。

输出：$VD(S)$，其中$S = S_L \cup S_R$。

步骤1. 找出S_L和S_R的凸包，将它们分别表示为Hull(S_L)和Hull(S_R)。(在此情况下找出凸包的特定算法会在稍后给出。)

步骤2. 找出将Hull(S_L)和Hull(S_R)合并成一个凸包的线段 $\overline{P_aP_b}$ 和 $\overline{P_cP_d}$ （P_a和P_b属于S_L，P_c和P_d属于S_R。）假如 $\overline{P_aP_b}$ 位于 $\overline{P_cP_d}$ 的上面，令$x = a$，$y = b$，$SG = \overline{P_xP_y}$，以及$HP = \phi$。

步骤3. 找出SG的垂直平分线，并用BS表示。令$HP = HP \cup BS$。如果$SG = \overline{P_cP_d}$，那么转向步骤5；否则，转向步骤4。

步骤4. 令BS首先与来自$VD(S_L)$或$VD(S_R)$的射线相交。该射线一定是相对于某z的 $\overline{P_xP_z}$ 或 $\overline{P_yP_z}$ 的垂直平分线。如果该射线是 $\overline{P_yP_z}$ 的垂直平分线，那么令$SG = \overline{P_xP_z}$；否则，令$SG = \overline{P_zP_y}$，并转向步骤3。

步骤5. 删除$VD(S_L)$中向HP右延伸的边，删除$VD(S_R)$中向HP左延伸的边。最终的图就是$S = S_L \cup S_R$的Voronoi图。

接下来详细描述HP的性质。在此之前，先定义一个点与一个点集间的距离。

定义 *已知点P和点P_1，P_2，…，P_n的集合S，令P_i是P的最近邻点，那么P与S间的距离就是P与P_i间的距离。*

根据该定义可以声明：从算法4-5中得到的HP是保持到S_L与S_R距离相等的点的轨迹。也就是，对于HP上的每个点，令P_i和P_j分别是P与S_L和S_R的最近邻点，那么P与P_i间的距离等于P与P_j间的距离。

例如，参见图4-21。设P是线段 $\overline{cd}$ 上的点。对于S_L，P的最近邻点是P_3；对于S_R，P的最近邻点是P_4。由于 $\overline{cd}$ 是 $\overline{P_3P_4}$ 的垂直平分线，所以，在 $\overline{cd}$ 上的每点到P_3和P_4是等距离的。这样，$\overline{cd}$ 上的每一点到S_L和S_R的距离是相等的。

利用上面HP的性质，很容易理解每条水平线H都与HP至少相交一次，这可参见图4-23来理解。

想象沿直线H从左向右移动。起初，某个点P_1的最近邻点一定是S_L中的点。假如从右向左移动，某个点P_2的最近邻点一定是S_R中的点。由于H是连续线，一定有一点P使得到P的左（右），每点都有在$S_L(S_R)$中的最近邻点。这样，P到S_L与S_R等距。或者，P与在S_L中的最近邻点间的距离一定等于P与在S_R中的最近邻点间的距离。因为HP是到S_L和S_R等距的点轨迹，该P也必定在HP上。这样，每条水平线必定与HP至少相交一次。读者可研究图4-21以确信前面解释的正确性。

参见图4-21，可以理解HP还有另一个有趣的性质；它对于y是单调的。接下来，证明每条水平线H与HP至多相交一次。

假如不然，那么有一点P_r使得HP转向上，如图4-24所示。

在图4-24中，$\overline{P_lP_r}$是$\overline{ab}$的垂直平分线，$\overline{P_rP_s}$是$\overline{bc}$的垂直平分线。由于HP是划分S_L和S_R的轨迹，那么或者a和c属于S_L，b属于S_R；或者a和c属于S_R，b属于S_L。这两种情况都是不可能的，因为有一条垂直于x轴的线将S划分成S_L和S_R。也就是P_r不可能存在。

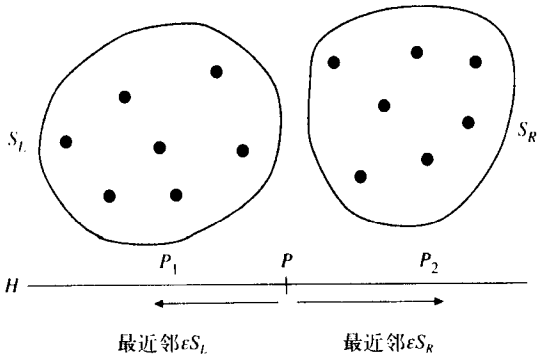


图4-23 水平线H与S_L和S_R间的关系

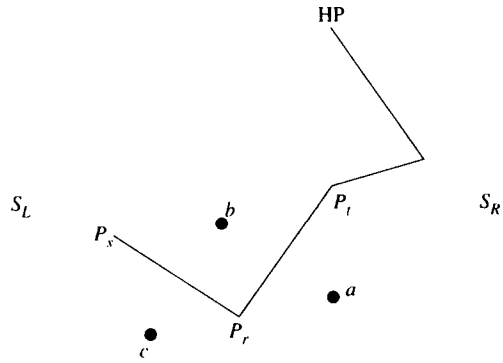


图4-24 HP单调性的举例

由于每条水平线H与HP至少相交一次，至多相交一次。所以推断，每条水平线H与HP相交于一点且仅一点。也就是HP对于y是单调的。

Voronoi图的另一个重要性质是用于推导构造Voronoi图的算法时间复杂度。该性质是：Voronoi图的边数至多是$3n-6$，其中n是点数。注意，每个Voronoi图与一个Delaunay三角剖分对应。由于Delaunay三角剖分是平面图。它至多包含$3n-6$条边。因为在Voronoi图中的边与Delaunay三角剖分的边之间一一对应，所以在Voronoi图中的边数至多是$3n-6$。

在得到Voronoi边的上界后，可推导Voronoi顶点的上界。注意，每个Voronoi顶点都与Delaunay三角剖分中的一个三角形对应。由于Delaunay三角剖分是个平面图，可将剖分看作此平面图的一个面。令F、E和V分别表示Delaunay三角剖分的面数、边数和点数，那么根据欧拉关系式（Euler' relation）：$F = E-V + 2$。在Delaunay三角剖分中，$V = n$，以及E至多为$3n-6$。所以，F至多为$(3n-6)-n + 2 = 2n-4$。也就是，Voronoi顶点数至多为$2n-4$。

现在已准备好推导合并过程的时间复杂度。谨记在合并过程中，必须找出两个凸包。在4.3节所介绍的算法在此不能使用，因为它的时间复杂度是$O(n \log n)$，这距离我们的目标太高了。现在将说明能很容易地找出这些凸包，因为$VD(S_L)$和$VD(S_R)$可用于构造这些凸包。

参见图4-25所示的有四条无限射线的Voronoi图，即b_{12}、b_{15}、b_{56}和b_{26}，与这些无限射线对应的点是P_2、P_1、P_5和P_6。实际上，可以通过连接这些点构造凸包，如图4-25所示的虚线。

在构造了Voronoi图之后，通过检查所有的Voronoi边直到找出一条无限射线，就可以构造凸包了。设P_i是此无限射线左边的一点，那么P_i就是一个凸包顶点。现在检查P_i的Voronoi多边形的Voronoi边，直到发现一条无限的射线。上面的过程重复直至返回到开始的射线为止。由

于每条边一定出现在两个Voronoi多边形上，不存在边被检查多过两次。因此，在构造Voronoi图之后，就可在时间$O(n)$内找出凸包。

构造基于分治策略的Voronoi图的算法的合并步骤的时间复杂度可以推导如下：

(1) 因为$VD(S_L)$和$VD(S_R)$业已构造，所以凸包可在$O(n)$时间内找出，且通过找无限射线找到了凸包。

(2) 将两个凸包合并成一个新凸包的边可在$O(n)$时间内确定，对此的解释在4.3节。

(3) 由于在$VD(S_L)$和$VD(S_R)$中至多有$3n-6$条边，且HP至多包含n条线段（因为HP对于y的单调性），HP的整个构造过程花费至多$O(n)$步。

因此，Voronoi图构造算法的合并过程是$O(n)$，因此

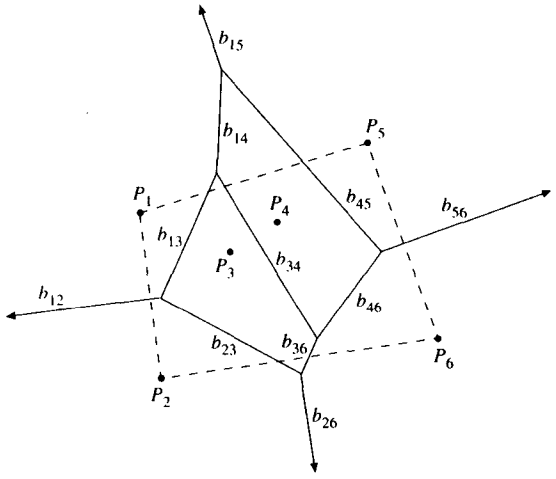


图4-25 从Voronoi图构造凸包

$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

我们得出结论：构造Voronoi图的分治算法时间复杂度是$O(n \log n)$。现在说明该算法是最优的。考虑一条直线上的点集，这样点集的Voronoi图是由如图4-26所示的平分线集所组成。在这些线构成之后，对于这些Voronoi边的线性扫描将完成排序功能。换句话说，Voronoi图问题不比排序问题容易。所以，Voronoi图问题的下界是$\Omega(n \log n)$，因此，该算法是最优的。

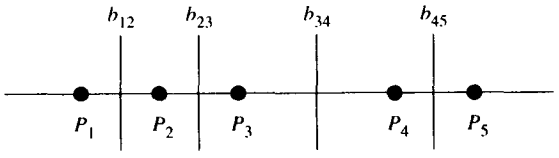


图4-26 直线上点集的Voronoi图

4.5 Voronoi图的应用

我们已经讨论了Voronoi图的概念，并且已经说明了如何将分治策略用于构造Voronoi图。在本节中，将说明Voronoi图有许多有趣的应用。当然，这些应用与分治策略无关。通过介绍Voronoi图的这些应用，希望激发大家对计算几何的兴趣。

欧几里得近邻搜索问题

第一个应用是解决欧几里得近邻搜索问题（Euclidean Nearest Neighbor Searching Problem），该问题定义如下：已知平面上n个点集合：P_1，P_2，…，P_n和一个检测点P，在P_i中找出P的最近邻居，比较的距离是欧几里得距离。

直接的方法是进行穷举搜索，该算法是$O(n)$时间的算法。使用Voronoi图，能减少搜索时间为$O(\log n)$，而预处理时间为$O(n \log n)$。

因为Voronoi图的基本性质，可以使用Voronoi图来解决欧几里得近邻搜索问题。注意，Voronoi图将整个平面划分成R_1，R_2，…，R_n个区域。在每个区域R_i内，有一个点P_i。如果测试点落在区域R_i内，那么在所有点中，它的最近邻点就是P_i。因此，可以通过简单地将该问题转换为一个区域定位问题（region location problem）而避免穷举搜索。也就是，如果能够确定检

测点P_i位于哪个区域R_i，就能确定该检测点的最近邻居。

Voronoi图是个平面图，所以可使用如图4-27所示的平面图的特殊性质。在图4-27中，Voronoi图是图4-22的重画。注意到共有六个Voronoi顶点。第一步是对这些顶点根据其y值排序。如图4-27所示，根据递减的y值，Voronoi顶点标记为V_1，V_2，…，V_6。通过每个Voronoi顶点，画一条水平线，这些水平线将整个空间分割成如图4-27所示的片隙。

在每个片隙内，有可以线性排序的Voronoi边，且这些Voronoi边可再分割片隙为区域，参见图4-27中的SL_5。在SL_5内，有三条Voronoi边：b_{23}、b_{34}和b_{46}。它们可排序成

$$b_{23}<b_{34}<b_{46}$$

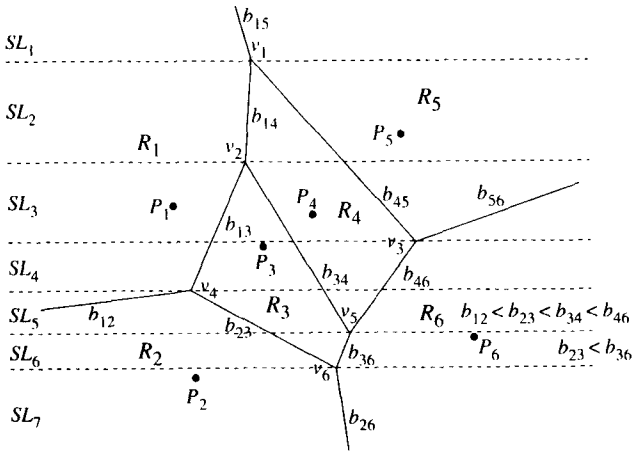


图4-27 Voronoi图解决欧几里得近邻搜索问题的应用

这三条边将片隙分割成四个区域：R_2、R_3、R_4和R_6。如果在边b_{23}和b_{34}间找出一点，那么它一定在区域R_3中。如果发现它在b_{46}的右边，那么它必定在区域R_6中。由于这些边是有序的，只要考虑这些Voronoi边，就可使用二分查找来确定检测点的位置。

欧几里得近邻搜索算法基本上是由两个主要的步骤组成：

(1) 进行二分查找以确定检测点在哪一个片隙内，由于至多有$O(n)$个Voronoi顶点，所以这可在时间$O(\log n)$内完成。

(2) 在每个片隙内，进行二分查找以确定该点在哪一个区域，由于至多有$O(n)$个Voronoi顶点，所以这可在时间$O(\log n)$内完成。

总的搜索时间是$O(\log n)$，很容易看到预处理时间是$O(n \log n)$，这基本上是构造Voronoi图所需的时间。

欧几里得所有近邻问题

接下来的应用是解决欧几里得所有近邻问题（Euclidean all nearest neighbor problem）。已知平面上的n个点集合：P_1，P_2，…，P_n。欧几里得最近点对问题是找出每个P_i的最近邻点。由于Voronoi图下面的性质，该问题可使用Voronoi图很容易地解决：如果P_j是P_i的最近邻点，那么P_i和P_j共享同一条Voronoi边。而且，$\overline{P_iP_j}$的中点正好位于此共享Voronoi边上。我们采用反证法证明该性质。假如P_i和P_j没有共用同一条Voronoi边，根据Voronoi多边形的定义，$\overline{P_iP_j}$的垂直平分线一定在与P_i相关的Voronoi多边形的外部。令$\overline{P_iP_j}$与平分线相交在点M，与某Voronoi边交于点N，如图4-28所示。

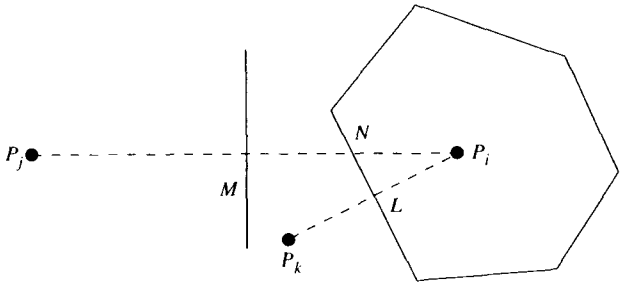


图4-28 Voronoi图的近邻性质的举例

假如Voronoi边在P_i和P_k之间，且$\overline{P_iP_k}$与Voronoi边相交于点L，那么得到

$\overline{P_iN} < \overline{P_iM}$

和 $\overline{P_iL} < \overline{P_iN}$。

这意味着

$$\overline{P_iP_k} = 2\overline{P_iL} < 2\overline{P_iN} < 2\overline{P_iM} = \overline{P_iP_j}$$

这是不可能的，因为P_j是P_i的最近邻居。

已知上面的性质，欧几里得所有近邻问题可通过考查每个Voronoi多边形的每条边来解决。由于每条Voronoi边一定是两个Voronoi多边形共享，没有Voronoi边检验多过两次。也就是，在Voronoi图构造之后，欧几里得所有近邻问题可在线性时间内解决。因此，该问题可在$O(n \log n)$时间内求解。

现在重画图4-22在图4-29中。对于P_4，四条边必须要检验，找出P_4的近邻是P_3。

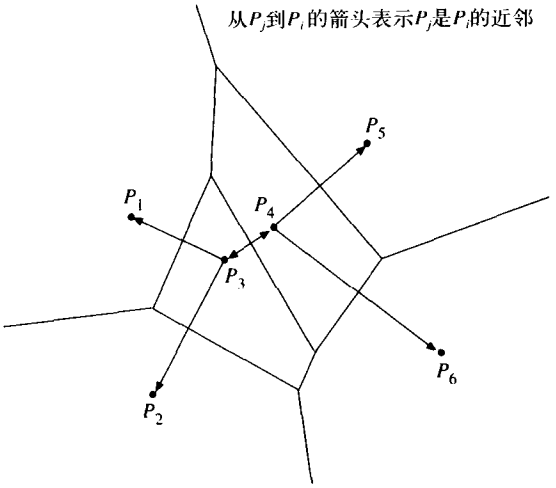


图4-29 所有近邻关系

4.6 快速傅里叶变换

快速傅里叶变换问题（Fast Fourier Transform problem）是计算下面的公式：

$$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}} \qquad 0 \leqslant j \leqslant n-1$$

其中$i = \sqrt{-1}$，且a_0，a_1，…，a_{n-1}是已知数。直接的计算方法将需要$O(n^2)$的步数。如果使用分治策略，那么时间复杂度可减少到$O(n \log n)$。

为了简化讨论，令$w_n = e^{\frac{2\pi i}{n}}$。这样，傅里叶变换就是计算下式：

$$A_j = a_0 + a_1 w_n^j + a_2 w_n^{2j} + \cdots + a_{n-1} w_n^{(n-1)j}$$

令$n = 4$，那么得到

$$A_0 = a_0 + a_1 + a_2 + a_3$$

$$A_1 = a_0 + a_1 w_4 + a_2 w_4^2 + a_3 w_4^3$$

$$A_2 = a_0 + a_1 w_4^2 + a_2 w_4^4 + a_3 w_4^6$$

$$A_3 = a_0 + a_1 w_4^3 + a_2 w_4^6 + a_3 w_4^9$$

可以重新调整上面的公式为下面的形式：

$$A_0 = (a_0 + a_2) + (a_1 + a_3)$$
$$A_1 = (a_0 + a_2 w_4^2) + w_4(a_1 + a_3 w_4^2)$$
$$A_2 = (a_0 + a_2 w_4^4) + w_4^2(a_1 + a_3 w_4^4)$$
$$A_3 = (a_0 + a_2 w_4^6) + w_4^3(a_1 + a_3 w_4^6)$$

由于 $w_n^2 = w_{n/2}$ 及 $w_n^{n+k} = w_n^k$，可得

$$A_0 = (a_0 + a_2) + (a_1 + a_3)$$
$$A_1 = (a_0 + a_2 w_2) + w_4(a_1 + a_3 w_2)$$
$$A_2 = (a_0 + a_2) + w_4^2(a_1 + a_3)$$
$$A_3 = (a_0 + a_2 w_4^2) + w_4^3(a_1 + a_3 w_4^2)$$
$$= (a_0 + a_2 w_2) + w_4^3(a_1 + a_3 w_2)$$

令

$$A_0 = a_0 + a_2$$
$$C_0 = a_1 + a_3$$
$$B_1 = a_0 + a_2 w_2$$
$$C_1 = a_1 + a_3 w_2$$

那么，

$$A_0 = B_0 + w_4^0 C_0$$
$$A_1 = B_1 + w_4^1 C_1$$
$$A_2 = B_0 + w_4^2 C_0$$
$$A_3 = B_1 + w_4^3 C_1$$

上面的公式表明分治策略可以漂亮地应用到解决傅里叶变换问题中。只需计算B_0、C_0、B_1和C_1，那么A_j就可很容易地得到。换句话说，一旦得到A_0，那么A_2就立即得到。类似地，一旦得到A_1，那么A_3也直接可得到。

但是，B_i和C_i是什么？注意到B_i是奇数输入项的傅里叶变换，而C_i是偶数输入项的傅里叶变换，这是应用分治策略解决傅里叶变换问题的基础。将大问题划分为两个子问题，递归地解决这两个子问题，再合并它们的解。

考虑一般情况下的A_j。

$$\begin{aligned} A_j &= a_0 + a_1 w_n^j + a_2 w_n^{2j} + \cdots + a_{n-1} w_n^{(n-1)j} \\ &= \left(a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \cdots + a_{n-2} w_n^{(n-2)j}\right) + w_n^j\left(a_1 + a_3 w_n^{2j} + a_5 w_n^{4j} + \cdots + a_{n-1} w_n^{(n-2)j}\right) \\ &= \left(a_0 + a_2 w_{n/2}^j + a_4 w_{n/2}^{2j} + \cdots + a_{n-2} w_{n/2}^{\frac{(n-2)j}{2}}\right) + w_n^j\left(a_1 + a_3 w_{n/2}^j + a_5 w_{n/2}^{2j} + \cdots + a_{n-1} w_{n/2}^{\frac{(n-2)j}{2}}\right) \end{aligned}$$

定义B_j和C_j如下：

$$B_j = a_0 + a_2 w_{n/2}^j + a_4 w_{n/2}^{2j} + \cdots + a_{n-2} w_{n/2}^{\frac{(n-2)j}{2}}$$

和

$$C_j = a_1 + a_3 w_{n/2}^j + a_5 w_{n/2}^{2j} + \cdots + a_{n-1} w_{n/2}^{\frac{(n-2)j}{2}}$$

那么，

$$A_j = B_j + w_n^j C_j$$

还可证明

$$A_{j+n/2} = B_j + w_n^{j+n/2} C_j$$

对于$n=2$，傅里叶变换如下：

$$A_0 = a_0 + a_1$$
$$A_1 = a_0 - a_1$$

基于分治法的傅里叶算法表示如下：

算法4-6 基于分治策略的快速傅里叶变换

输入：a_0，a_1，…，a_{n-1}，$n=2^k$。

输出：$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}}$，对于$j=0$，1，2，…，$n-1$。

步骤1. 如果$n=2$，

$A_0 = a_0 + a_1$

$A_1 = a_0 - a_1$

返回。

步骤2. 递归地找出a_0，a_1，…，a_{n-2}(a_1，…，a_{n-1})的傅里叶变换系数。令系数表示成B_0，B_1，…，$B_{n/2}$(C_0，C_1，…，$C_{n/2}$)。

步骤3. For $j=0$ to $j=n/2-1$

$A_j = B_j + w_n^j C_j$

$A_{j+n/2} = B_j + w_n^{j+n/2} C_j$。

上面算法的时间复杂度显然是$O(n \log n)$。傅里叶变换逆变换是将A_0，A_1，…，A_{n-1}按如下方法变换回a_0，a_1，…，a_{n-1}。

$$a_j = \frac{1}{n}\sum_{k=0}^{n-1} A_k e^{\frac{-2\pi ijk}{n}} \quad 对于j=0，1，2，\cdots，n-1$$

考虑一个例子，令$a_0=1$，$a_1=0$，$a_2=-1$，$a_3=0$，那么

$B_0 = a_0 + a_2 = 1 + (-1) = 0$

$B_1 = a_0 - a_2 = 1-(-1) = 2$

$C_0 = a_1 + a_3 = 0 + 0 = 0$

$C_1 = a_1 - a_3 = 0-0 = 0$

$w_4 = e^{\frac{2\pi i}{4}} = \mathrm{i}$

$w_4^2 = -1$

$w_4^3 = -i$

因此，

$A_0 = B_0 + w_4^0 C_0 = B_0 + C_0 = 0 + 0 = 0$

$A_1 = B_1 + w_4 C_1 = 2 + 0 = 2$

$A_2 = B_0 + w_4^2 C_0 = 0 - 0 = 0$

$A_3 = B_1 + w_4^3 C_1 = 2 + 0 = 2$

易知傅里叶变换的逆变换就是将A_i正确地变换回a_i。

4.7 实验结果

为了说明分治策略是个很有用的策略，我们实现了基于分治策略的最近点对算法和穷举检验每对顶点的直接算法。两个程序都是在IBM个人计算机上实现的，如图4-30总结了实验结果。如图4-30所示，当n较小时，直接方法实现得较好。但是，随着n的增大，分治算法将更快。例如，当n等于7000时，分治算法几乎比直接方法快了200倍。这样的结果是可以预测的，因为直接算法的时间复杂度是$O(n^2)$，而分治算法的时间复杂度是$O(n \log n)$。

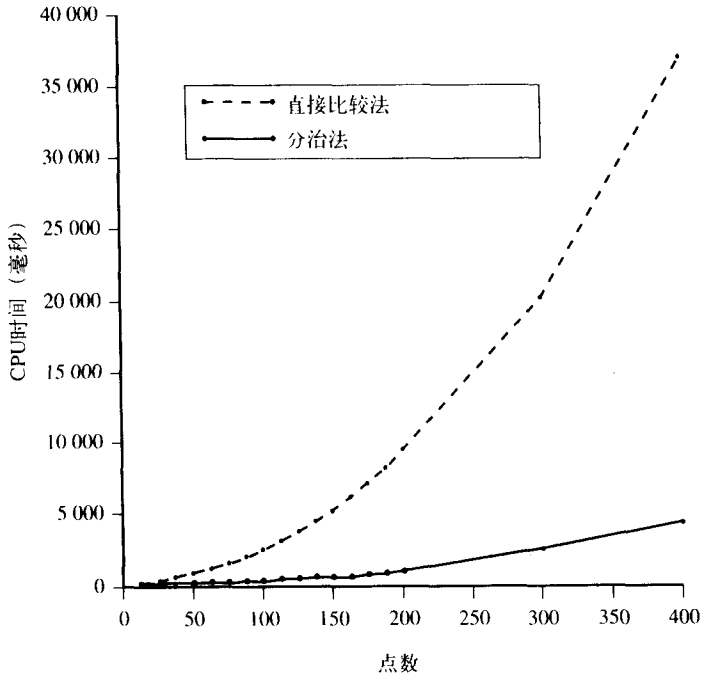


图4-30 求最近点对问题的实验结果

4.8 注释与参考

分治策略经常用于解决计算几何问题，可参阅文献Preparata and Shamos (1985)和Lee and Preparata (1984)。分治策略也在文献Horowitz and Sahni(1978)和Aho，Hopcroft and Ullman (1974)中讨论。

关于极大值问题的完全讨论，参阅文献Pohl(1972)。最近点对问题和求极大值问题也在文献Bentley(1980)中解决。

Voronoi图首先在文献Voronoi(1908)中讨论。另外，关于Voronoi图的更多信息可在文献Preparata and Shamos(1985)中找到。对于Voronoi图的更高阶和更高维的一般性讨论，可参阅文献Lee(1982)和Brown(1979)。

不同的凸包算法可在文献Levcopoulos and Lingas(1987)和Preparata and Shamos(1985)中找到。Graham扫描在文献Graham(1972)中提出。3维凸包也可由分治策略找出，这由文献Preparata and Hong(1977)提出。

通过分治法解决傅里叶变换有文献Cooley and Tukey(1965)。文献Gentleman and Sande (1966)讨论了快速傅里叶变换的应用。Brigham的书（Brigham，1974）是完全地对快速傅里叶变换的介绍。

分治也可用于计算有效的矩阵相乘算法，参见文献Strassen(1969)。

4.9 进一步的阅读资料

对于许多研究者分治策略仍然是有吸引力的主题，它对计算几何特别有效。推荐下面的论文作进一步研究：Aleksandrov and Djidjex (1996)；Bentley (1980)；Bentley and Shamos (1978)；Blankenagel and Gueting (1990)；Bossi，Cocco and Colussi (1983)；Chazelle，Drysdale and Lee (1986)；Dwyer (1987)；Edelsbrunner，Maurer，Preparata，Rosenberg，Welzl and Wood (1982)；Gilbert，Hutchinson and Tarjan (1984)；Gueting (1984)；Gueting and Schilling (1987)；Karp (1994)；Kumar，Kiran and Pandu (1987)；Lopez and Zapata (1994)；Monier (1980)；Oishi and Sugihara (1995)；Reingold and Supowit (1983)；Sykora and Vrto(1993)；Veroy (1988)；Wslsh (1984)；and Yen and Kuo (1997)。

对于一些最新出版的论文感兴趣可参阅Abel (1990)；Derfel and Vogl (2000)；Dietterich (2000)；Even，Naor，Rao and Schieber(2000)；Fu (2001)；Kamidoi，Wakabayashi and Yoshida (2002)；Lee and Sheu (1992) Liu (2002)；Lo，Rajopadhye，Telle and Zhong (1996)；Melnik and Garcia-Molina (2002)；Messinger，Rowe and Henry (1991)；Neogi and Saha (1995)；Rosler (2001)；Rosler and Ruschendorf (2001)；Roura (2001)；Tisseur and Dongarra (1999)；Tsai and Katsaggelos (1999)；Verma (1997)；Wang (1997)；Wang (2000)；Wang，He，Tang and Wee (2003)；Yagle (1998)；and Yoo，Smith and Gopalarkishnan (1997)。

习题

4.1 二分查找可使用分治策略吗？

4.2 直接将两个n比特数u和v相乘需要$O(n^2)$步数。使用分治方法，可将数划分成两个相等的部分，通过下面的方法计算乘积：

$$uv = (a \cdot 2^{n/2} + b) \cdot (c \cdot 2^{n/2} + d) = ac \cdot 2^n + (ad + bc) \cdot 2^{n/2} + bd$$

如果将$ad + bc$计算成$(a + b)(c + d) - ac - bd$，计算时间是多少？

4.3 证明：在快速排序中，需要的最大栈是$O(\log n)$。

4.4 实现基于分治法的快速傅里叶转换算法，并与直接方法做比较。

4.5 实现基于分治法的找秩算法，并与直接方法做比较。

4.6 令 $T\left(\frac{n^2}{2^r}\right) = nT(n) + bn^2$，其中$r$是一个整数，且$r \geqslant 1$，计算$T(n)$。

4.7 参阅文献Horowitz and Sahni(1978)的3.7节，了解基于分治的Strassen矩阵乘法。

4.8 令

$$T(n) = \begin{cases} b & \text{对 } n = 1 \\ aT(n/c) + bn & \text{对 } n > 1 \end{cases}$$

其中a、b和c都是非负常数。

证明：如果n是c的幂时，那么

$$T(n)=\begin{cases}O(n) & \text{如果 } a<c\\ O(n\log_a n) & \text{如果 } a=c\\ O(n^{\log_c a}) & \text{如果 } a>c\end{cases}$$

4.9 证明：如果$T(n)=mT(n/2)+an^2$，那么$T(n)$满足

$$T(n)=\begin{cases}O(n^{\log m}) & \text{如果 } m>4\\ O(n^2\log n) & \text{如果 } m=4\\ O(n^2) & \text{如果 } m<4\end{cases}$$

4.10 一种基于分治的特殊排序算法是奇偶合并排序（odd-even merge sorting），由Batcher(1968)发明。对于该排序算法，参阅文献Liu(1977)的7.4节。该排序算法适合作为顺序算法吗？为什么？（这是个著名的并行排序算法。）

4.11 对于n个数的序列，设计一个$O(n\log n)$时间的算法，找出最长的单调递增的子序列。

第 5 章　树搜索策略

在本章中，将说明有许多问题的解可以用树来表示，因此解决这些问题就变成了树搜索问题（tree searching problem）。我们首先研究将在第8章讨论的可满足性问题（satisfiability problem），已知一个子句集（set of clauses），确定这个子句集是否可满足的一种方法是考查所有可能的指派（assignment）。也就是，如果有n个变量x_1，x_2，…，x_n，那么简单地测试所有可能的2^n个指派。在每个指派中，x_i 可指派为T或者F。假如$n = 3$，需要考查下列指派：

x_1	x_2	x_3
F	F	F
F	F	T
F	T	F
F	T	T
T	F	F
T	F	T
T	T	F
T	T	T

另一方面，上面这8个指派可以用一棵树表示，如图5-1所示。为什么该树能具有这么丰富的信息呢？这些指派的树表示说明在顶层，确实有两类指派：

类1：那些$x_1 = T$的指派。

类2：那些$x_1 = F$的指派。

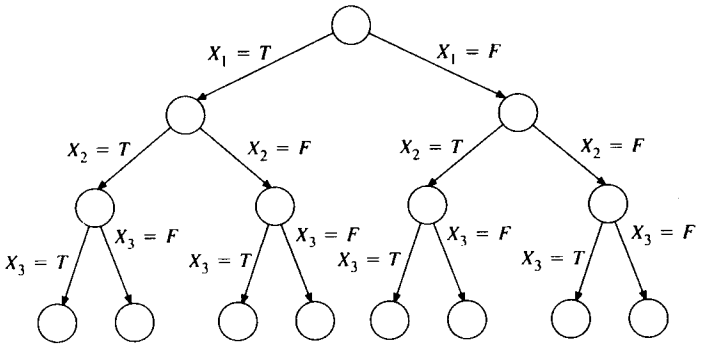


图5-1　八个指派的表示

这样，能递归地对每类指派划分成两个子类。基于这样的观点，只需确定指派的所有分类而不用去考查所有的值，就能确定其可满足性。例如，假定有下面的子句集：

$-x_1$ (1)

x_1 (2)

$x_2 \vee x_5$ (3)

x_3 (4)

$-x_2$ (5)

现在可画出一棵部分树，如图5-2所示。

从图5-2所示的树中，可容易地看到$x_1 = T$的指派将不满足子句(1)，而$x_1 = F$的指派将不满足子句(2)。由于在每个指派中，x_1只能指派为T或F，所以不需要测试每一个指派。那么，现在可确定上面子句集的不可满足性。

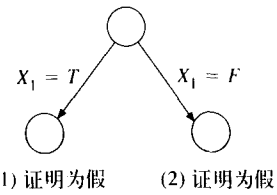


图5-2 确定可满足性问题的部分树

许多其他相似的问题也可通过树的搜索技术解决。考虑有名的8数码问题（8-puzzle problem）。在图5-3中展示了一个正方形结构，它包含有九个格子，仅有八个格子有数字，有一个空格子。这里的问题是移动这些带数字的小方块，使得能到达如图5-4所示的终态。这些带数字的小方块仅能水平或垂直移动到空的方格中。这样，对于如图5-3所示的初始状态，仅有两种可能的移动如图5-5所示。

2	3	
5	1	4
6	8	7

图5-3 一个8数码的初始安排

1	2	3
8		4
7	6	5

图5-4 此8数码问题的目标状态

这样，该8数码问题变成了一个树搜索问题，因为可以逐渐扩展这棵解树。只要表示最终目标的结点出现，那么问题就解决了。在后面几节中将详述这些内容。

最后，将说明哈密顿回路问题（Hamiltonian cycle problem）的解空间也能用一棵树方便地表示。已知图$G = (V, E)$是有n个顶点的连通图。哈密顿回路是一条环形路径，经过图G的n个顶点，并且每个顶点仅访问一次，最终回到最初的位置。参见图5-6所示，通过下列序列表示的路径是一条哈密顿回路：1，2，3，4，7，5，6，1。在图5-7中，没有哈密顿回路。

哈密顿回路问题是确定在一个已知的图中是否包含哈密顿回路。这是一个NP完全问题。它仍然能通过测试所有可能的解来解决，并且这些解可方便地用一棵树来表示。注意，哈密顿回路一定要访问图的每个结点，因此可假定结点1是开始结点。再次参见图5-6，哈密顿回路的搜索可通过一棵树来描述，如图5-8所示。由于回路1，2，3，4，7，5，6，1与回路1，6，5，7，4，3，2，1是相同的，因此，说明存在一条并且仅存在一条哈密顿回路。

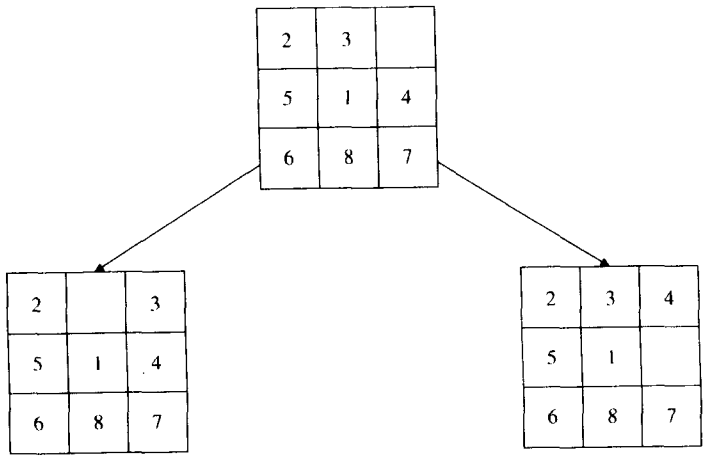


图5-5 8数码问题的最初安排的两个可能移动

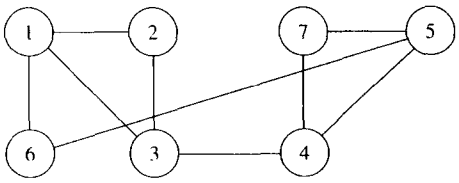


图5-6 一个包含哈密顿回路的图

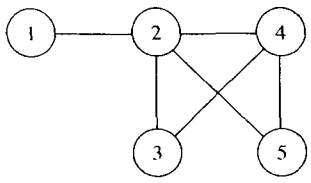


图5-7 一个不包含哈密顿回路的图

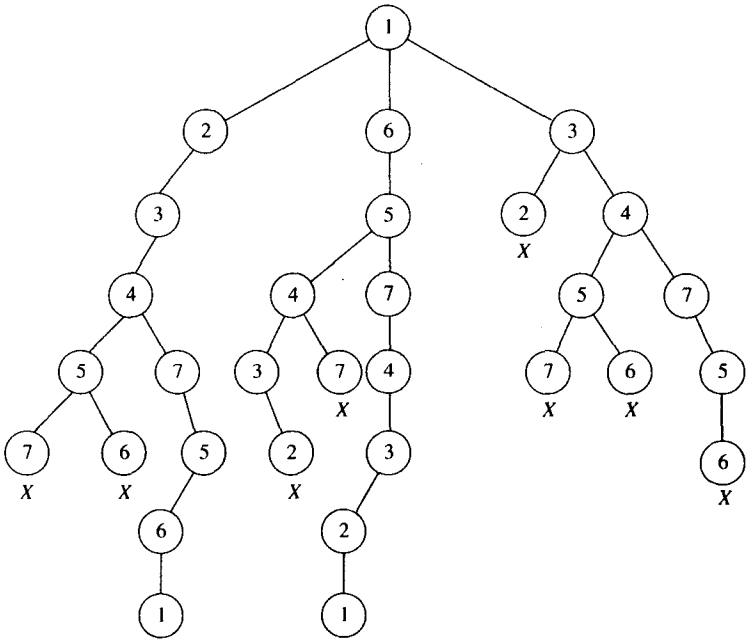


图5-8 在图5-6中的图是否存在哈密顿回路的树表示

考查图5-7，是否存在一棵对应该图的哈密顿回路的树表示在图5-9中。此次，很容易能看出不存在一条哈密顿回路。

业已证明许多问题都能通过树来表示。在本章的剩余部分中，将讨论解决这些问题的剪枝树策略（strategy of pruning tree）。

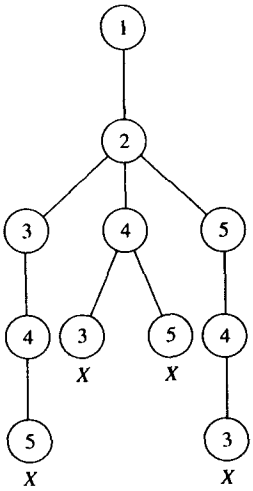


图5-9 不存在任何哈密顿回路的树

5.1 广度优先搜索

广度优先搜索（breadth-first search）也许是剪枝一棵树最直接的方法。在广度优先搜索中，树中某一层的所有结点在考查下一层所有结点以前被考查。图5-10表示一个典型的解决8数码问题的广度优先搜索。

注意到结点6表示一个目标结点。因此，搜索将停止。

广度优先搜索的基本数据结构是一个能容纳所有扩展结点的队列。下列方案说明了广度优先搜索。

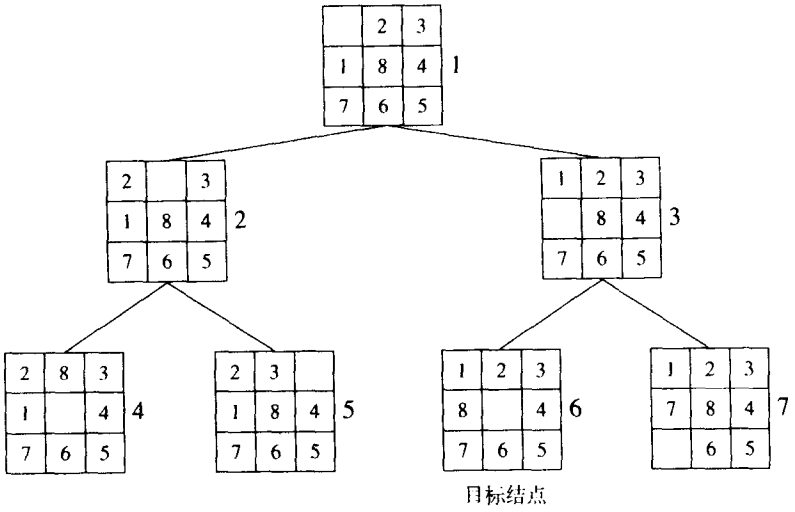


图5-10 通过广度优先搜索产生的搜索树

广度优先搜索

步骤1. 构造由根结点组成的1元队列。
步骤2. 考查该队列的第一个元素是否目标结点。如果是，那么停止；否则，转去步骤3。
步骤3. 从该队列中删除第一个元素，如果有的话，增加第一个元素的后代到该队列的最后。
步骤4. 如果该队列是空的，那么失败；否则，转去步骤2。

5.2 深度优先搜索

深度优先搜索（depth-first search）总是选择最深的结点展开。考虑下面的子集和问题（sum of subset problem）。已知集合$S=\{7, 5, 1, 2, 10\}$，确定是否存在元素和等于9的子集S'。这个问题可通过深度优先搜索容易地解决，如图5-11所示。在图5-11中许多结点被终止，显而易见因为它们不能产生问题的解。图5-11中每个圆圈中的数字代表一个子集和。注意在这个过程中，总是选择最深的结点进行扩展。

现在考虑哈密顿回路问题。对于图5-12，通过深度优先搜索找出一条哈密顿回路，如图5-13所示。

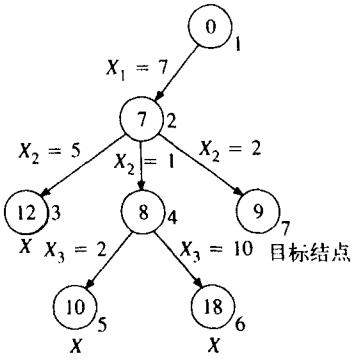


图5-11 通过深度优先搜索解决子集和问题

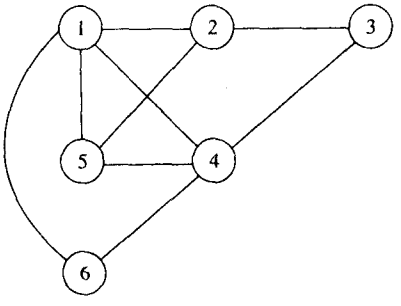


图5-12 一个包含哈密顿回路的图

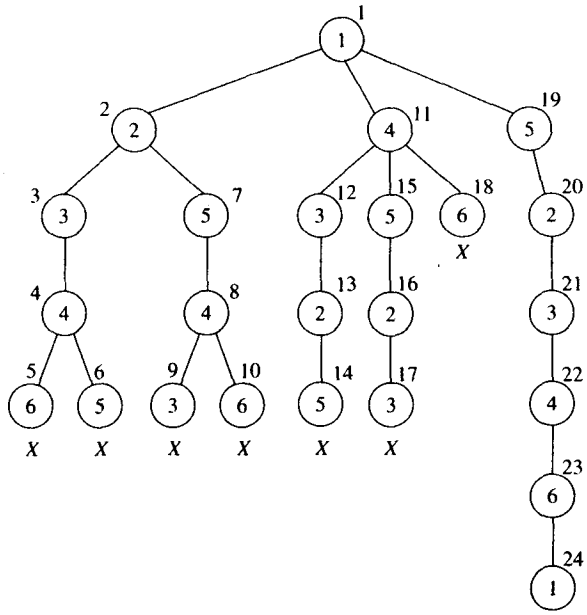


图5-13 通过深度优先搜索产生的哈密顿回路

总结深度优先搜索如下：

深度优先搜索

步骤1. 构造由根结点组成的1元栈。
步骤2. 考查该栈的栈顶元素是否目标结点。如果是，那么停止；否则，转去步骤3。
步骤3. 从该栈中移除顶元素，如果有后代，将其增加到该栈顶。
步骤4. 如果该栈是空的，那么失败；否则，转去步骤2。

5.3 爬山法

在阅读完深度优先搜索这节后，读者也许想知道：在所有的后代结点中，选择哪一个结点扩展？在本节将介绍一种方法，称为爬山法（hill climbing）。爬山法是深度优先搜索的一种变形。在这种方法中，使用某种贪心法来帮助我们决定在搜索空间中向哪一个方向移动。通常，贪心法使用一些启发式的方法对各种可能的选择排序。启发式的方法越好，爬山法也越好。

再次考虑8数码问题。假如贪心法使用下面简单的评价函数（evaluation function）$f(n)$确定选择的次序：

$$f(n) = w(n)$$

其中$w(n)$是结点n错误放置的方格数。这样，假如开始结点如图5-14所示放置，因为1，2和8是错误放置的，所以$f(n)$等于3。

2	8	3
1		4
7	6	5

图5-14 8数码问题的开始结点

图5-15所示的是应用爬山法的结果，该方法使用f作为启发，在结点的所有后代中做出选择的次序，每个结点f的值如图5-15所示。在每条边上，结点扩展的顺序也标注了。注意爬山法仍然采用深度优先搜索，只是在这个结点的所有后代中，爬山法选择局部最优的结点去扩展。

读者不应该得出爬山法是极其有效的错误印象，因为图5-15

所示的例子是这样的。在图5-15中，在最先被扩展的结点中有两个结点有相同的评价值。假如扩展另一个结点，那么需要更长的时间才能得到解。

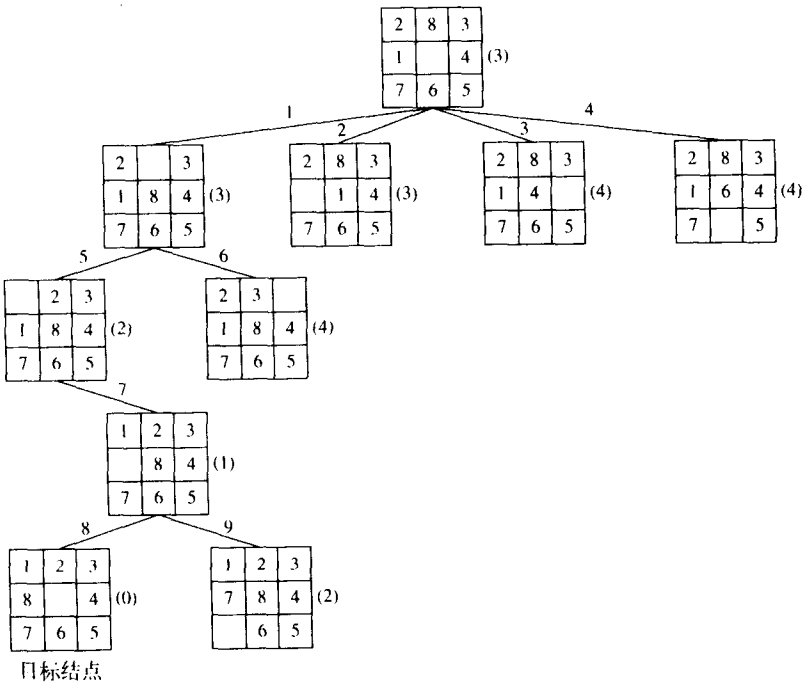


图5-15 通过爬山法解决8数码问题

爬山法的方案

步骤1. 构造由根结点组成的1元栈。

步骤2. 考查该栈的栈顶元素是否目标结点。如果是，那么停止；否则，转向步骤3。

步骤3. 从该栈中删除栈顶元素并扩展该元素，增加该元素的后代到该栈中，进栈的次序依据评价函数值。

步骤4. 如果该栈是空的，那么失败；否则，转向步骤2。

5.4 最佳优先搜索策略

最佳优先搜索策略（best-first search strategy）是将深度优先和广度优先搜索优点进行组合而成的一种简单方法。在最佳优先搜索中，有一个评价函数，总是选择到目前为止产生的所有结点中具有最小代价的结点。可以看出最佳优先搜索不像爬山法，它具有全局观点。

最佳优先搜索方案

步骤1. 使用评价函数构造一个堆，首先构造由根结点组成的1元堆。

步骤2. 考查该堆的根元素是否目标结点。如果是，那么停止；否则，转向步骤3。

步骤3. 从该堆中删除根元素，并扩展该元素，把这个元素的后代添加到堆中。

步骤4. 如果该堆是空的，那么失败；否则，转向步骤2。

如果在最佳优先搜索中，使用与爬山法相同的启发式方法，那么8数码问题的解如图5-16所示。

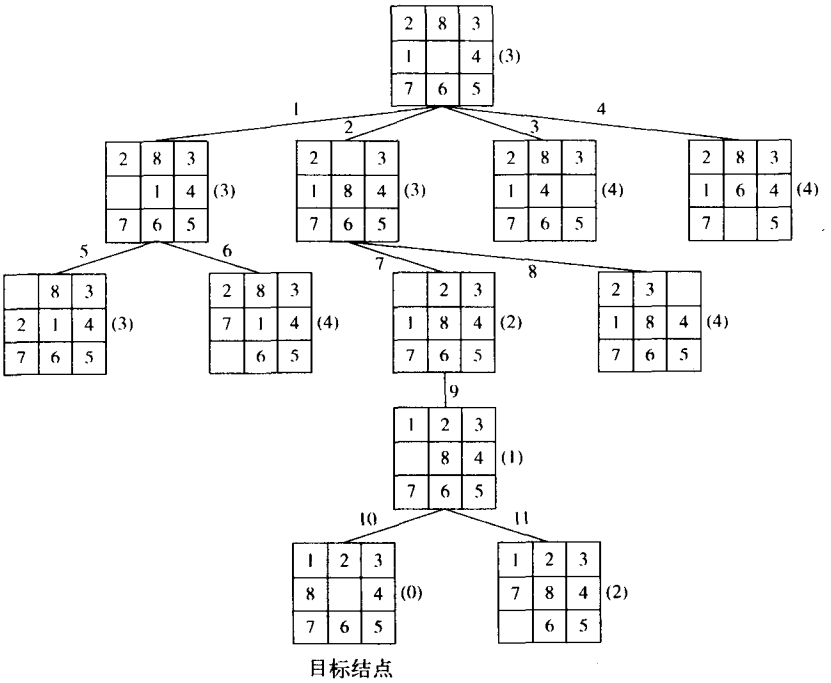


图5-16 用最佳优先搜索方法解决8数码问题

5.5 分支限界策略

在前面几节中，我们使用树搜索技术解决了许多问题。这些问题都不是优化问题。使读者感兴趣的是注意到上面的方法都不能用于解决任何优化问题（optimization problem）。在本节中，将介绍分支限界策略（branch-and-bound strategy），这也许是解决一大类组合问题的最有效策略之一。基本上，该策略表明问题可能有许多可行解（feasible solution）。然而，人们通过发现许多可行解不能成为最优解（optimal solution）来修剪解空间（solution space）。

现在通过图5-17来解释分支限界策略的基本原理。

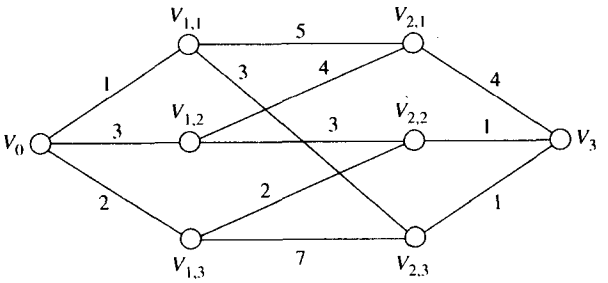


图5-17 一个多级图搜索问题

在图5-17中，问题是找出从V_0到V_3的最短路径。这个问题能先通过转换成一棵树搜索问题而有效地解决，如图5-18所示。

图5-18显示了所有6个可行解。如果不使用穷举搜索，那么分支限界策略是如何帮助我们找到最短路径的呢？图5-19说明了使用某种爬山法的过程。在图5-19a中，搜索树的根结点扩展出来的3个结点。在这3个结点中，必须选择一个结点来扩展。有许多方法选择下一步要扩展的结点。

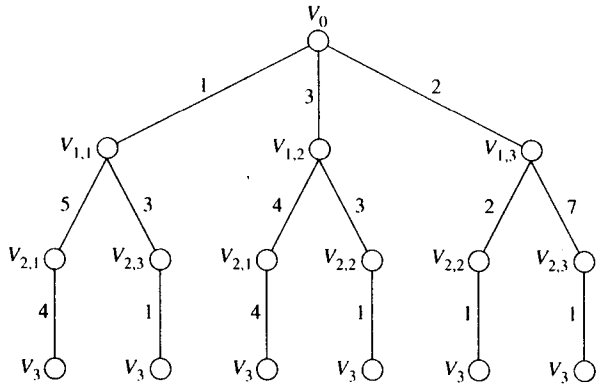


图5-18 图5-17中问题解的树表示

在这个例子中，假定使用爬山法，也就是在所有最近扩展的结点中，总是选择最少代价的结点作为下一个扩展的对象。

使用这个原理，结点b将被扩展，它的两个孩子如图5-19b所示。由于结点f对应于$V_{2,3}$，并且在结点e和f中，它的代价最少，所以f将被扩展。因为结点g是一个目标结点，如图5-19c所示，我们已经发现一个可行解，这个可行解的代价是5。

这个代价等于5的可行解可作为最优解的上界。任何代价大于5的解都不能成为一个最优解。因此，这个上界能终止许多不成熟的分支。例如，结点e将决不会产生任何最优解，由于具有结点e的解代价肯定超过6。

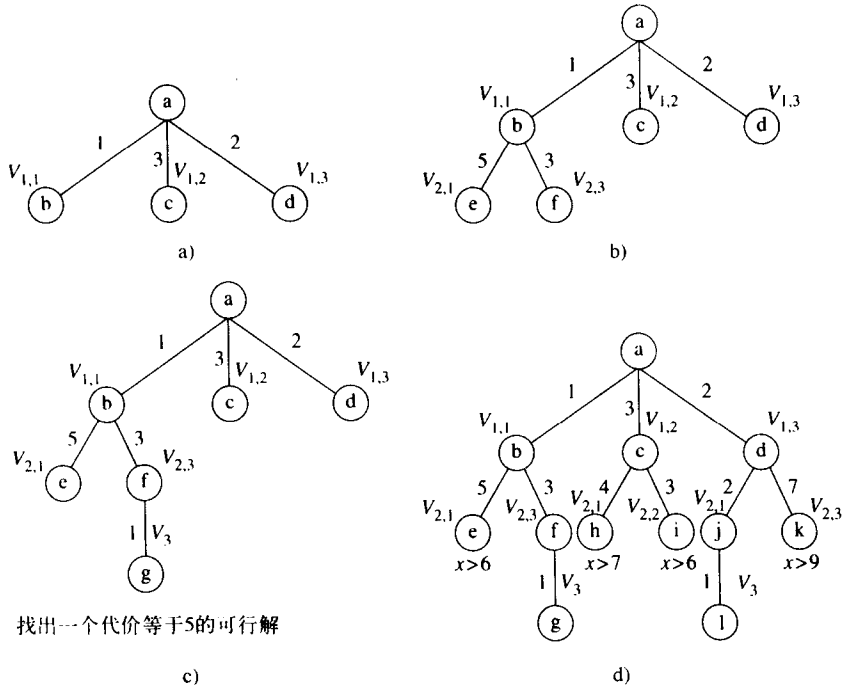


图5-19 分支限界策略的举例

如图5-19所示，穷举搜索整个解空间是可以避免的。当然，必须指出还有另一个最优解存在。上面的例子说明了分支限界策略的基本原理。这个策略由两个重要机制组成：一个机制是

产生分支，另一个机制是产生一个界限以致于能终止许多分支。尽管分支限界策略通常是非常有效的，但在最坏情况下，仍然可能生成一棵规模非常大的树。因此，我们必须认识到分支限界策略在平均情况下是有效的。

5.6 用分支限界策略解决人员分配问题

现在介绍如何使用分支限界策略有效解决人员分配问题（personnel assignment problem），这个问题是NP完全的。有一个人员线序集（linearly ordered set）$P = \{P_1, P_2, \cdots, P_n\}$，其中$P_1<P_2<\cdots<P_n$。可以设想这个人员顺序是通过某个标准确定的，例如身高、年龄、资历等。还有一个工作集$J = \{J_1, J_2, \cdots, J_n\}$，并且假定这些工作是偏序的（partially ordered）。每个人分配一项工作。令P_i和P_j各自分配的工作分别是$f(P_i)$和$f(P_j)$。如果$f(P_i)\leqslant f(P_j)$，那么$P_i\leqslant P_j$。函数f可以解释为一个可行的分配，该分配映射人到适合他们的工作。如果$i\neq j$，那么$f(P_i)\neq f(P_j)$。

考虑下面的例子。$P = \{P_1, P_2, P_3\}$，$J = \{J_1, J_2, J_3\}$，并且J的偏序是$J_1\leqslant J_3$和$J_2\leqslant J_3$。在这种情况下，$P_1\rightarrow J_1$，$P_2\rightarrow J_2$和$P_3\rightarrow J_3$是可行的分配，而$P_1\rightarrow J_1$，$P_2\rightarrow J_3$和$P_3\rightarrow J_2$是不可行的分配。

进一步假设将P_i分配了工作J_j的代价表示为C_{ij}。如果P_i分配了J_j，那么令X_{ij}就为1，否则为0。那么与可行性分配相应的总代价f为

$$\sum_{i,j} C_{ij}X_{ij}$$

对人员分配问题精确的定义如下：已知一个人员线序集$P = \{P_1, P_2, \cdots, P_n\}$，其中$P_1<P_2<\cdots<P_n$，以及一个工作偏序集$J = \{J_1, J_2, \cdots, J_n\}$，某人$P_i$分配到工作$J_j$的代价为$C_{ij}$，每个人分配一项工作且没有两个人分配同样的工作。人员分配问题是要找到一种最优的可行分配，使得下面的代价最小

$$\sum_{i,j} C_{ij}X_{ij}$$

这样，该问题是一个优化问题，并且已被证明是NP难的。此处不讨论NP困难性。

为了解决该问题，将使用“拓扑排序”（topological sorting）的概念。对于一个已知的偏序集S，当在这个偏序中，$S_i\leqslant S_j$表示在此序列中S_i位于S_j之前，那么线序$S_1, S_2, \cdots, S_n$是关于集合S拓扑有序的。例如，对于如图5-20所示的偏序，其相应的拓扑有序序列是1，3，7，4，9，2，5，8，6。

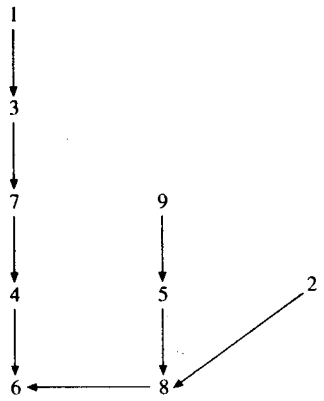


图5-20 一个偏序

令$P_1\rightarrow J_{k_1}$，$P_2\rightarrow J_{k_2}$，$\cdots$，$P_n\rightarrow J_{k_n}$为可行的分配。根据问题定义，工作是偏序的而人员是线序的。因此，相对于工作的偏序，$J_{k_1}, J_{k_2}, \cdots, J_{k_n}$一定是拓扑有序序列。通过一个例子来阐述该思想，考虑$J = \{J_1, J_2, J_3, J_4\}$和$P = \{P_1, P_2, P_3, P_4\}$，$J$的偏序如图5-21所示。

图5-21 工作的一个偏序

下面是所有的拓扑有序序列：

J_1, J_2, J_3, J_4

J_1, J_2, J_4, J_3

J_1, J_3, J_2, J_4

J_2, J_1, J_3, J_4

J_2, J_1, J_4, J_3

每个序列表示一个可行的分配。例如，对于第一个序列，

对应的可行分配是

$$P_1 \rightarrow J_1,\ P_2 \rightarrow J_2,\ P_3 \rightarrow J_3,\ P_4 \rightarrow J_4$$

树搜索技术能容易地用于找出所有的拓扑有序序列。例如，对于图5-21中所示的偏序序列，显示其所有的拓扑有序序列的树如图5-22所示。

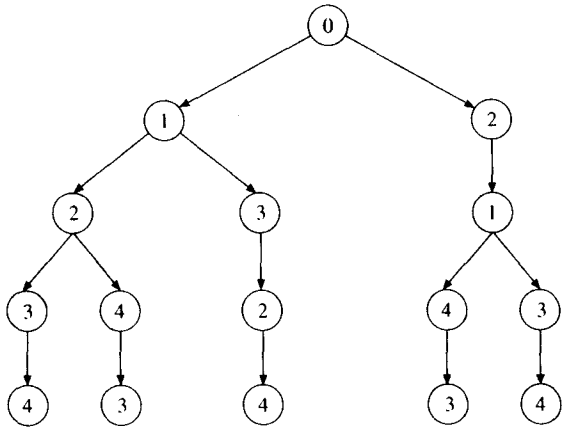


图5-22 对应于图5-21的所有拓扑有序序列的树表示

图5-22中的树通过下面三个基本步骤产生。

(1) 取一个元素，在这个偏序序列中它都不先于任何其他元素。

(2) 选择此元素作为拓扑有序序列中的一个元素。

(3) 把这个刚选择的元素从偏序集中删除，剩余的集合仍然是偏序的。

例如，对于图5-21中所示的偏序。在开始时，J_1和J_2是没有前驱的元素。因此，它们在树的同一层上。考虑相应于1的结点，如果删除1，那么偏序集包含2、3和4。在新的集合中只有2和3没有前驱。因此，生成2和3。

在阐述了如何用一棵树描述问题的解空间之后。现在，继续阐述如何使用分支限界策略找到最优解。

已知代价矩阵，能立即计算出解的一个下界。该下界通过某种方法简化代价矩阵获得，不影响解并且在每行每列至少有一个0，以及代价矩阵中所有其他项的值非负。

注意到，如果从这个代价矩阵的任何行或列中减去一个常数，不会改变最优解，参见表5-1所示的代价矩阵。对于该数据集，可从1、2、3和4行分别减去12、26、3和10，再从第2列减去3，所得到的矩阵是一个简化的代价矩阵，每行每列至少包含一个0并且矩阵中其他的值非负，如表5-2所示。被减去的值的总和为12 + 26 + 3 + 10 + 3 = 54，这是解的一个下界。

表5-1 人员分配问题的代价矩阵

人员＼工作	1	2	3	4
1	29	19	17	12
2	32	30	26	28
3	3	21	7	9
4	18	13	10	15

表5-2 简化的代价矩阵

人员 \ 工作	1	2	3	4	
1	17	4	5	0	
2	6	1	0	2	总计 = 54
3	0	15	4	6	
4	8	0	0	5	

图5-23表示一棵与该简化代价矩阵相关联的枚举树（enumeration tree）。如果使用最小下界，那么不能产生最优解的子解将会在更早的阶段被修剪掉，如图5-24所示。

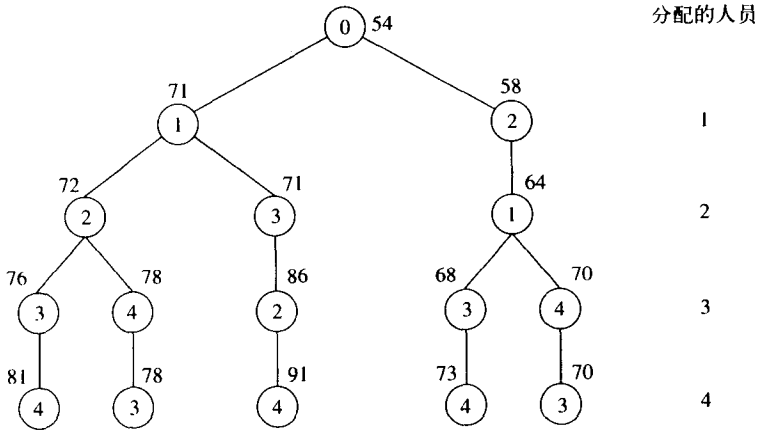


图5-23 与表5-2中简化代价矩阵相关联的枚举树

在图5-24中看到在找出代价为70的解后，可以立即剪去已经给P_1分配J_1开始的所有解，因为其代价71比70大。

为什么可以从代价矩阵中减去一些值呢？假如不这么做，那么考虑对应于分配$P_1 \rightarrow J_1$的结点。与这个结点相关的代价仅是29。想像我们已经找出了代价为70的可行解，也就是$P_1 \rightarrow J_2$，$P_2 \rightarrow J_1$，$P_3 \rightarrow J_4$和$P_4 \rightarrow J_3$。尽管已经找到一个上界，但不能使用它剪去对应于$P_1 \rightarrow J_1$的结点，因为它的代价仅是29，比70低。

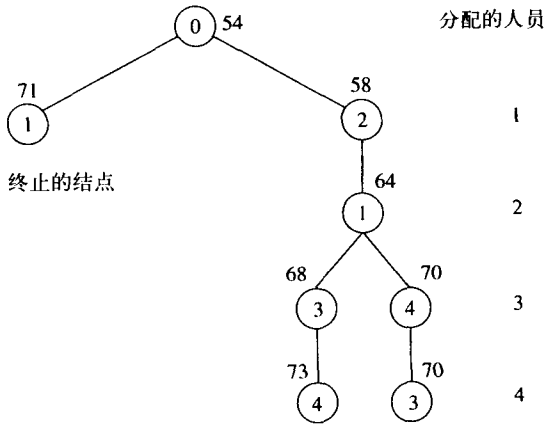


图5-24 子解的界

再看图5-24。现在能看到给P_1分配J_1相关的代价是71，而不是29。这样产生了一个界限。为什么能有这样高的代价呢？因为我们已经从初始的代价矩阵中减去了一些值，使每行每列都包含一个0。这样，在减去后，所有的解有相对比较高的下界，即54。换句话说，没有代价能比54低。使用这个信息我们知道，给P_1分配J_1的下限是54 + 17 = 71，而不是29。当然，一个相对比较高的下限会导致相对较早地终止。

5.7 用分支限界策略解决旅行商优化问题

旅行商决策问题（traveling salesperson decision problem）是一个NP完全问题。因此，在

最坏情况下旅行商问题（traveling salesperson problem）是很难解决的。但是，如将在本节中所介绍的，旅行商问题可通过使用分支限界策略来解决。也就是说，假如足够幸运，可以避免对解空间进行穷举搜索。

使用分支限界策略解决旅行商优化问题的基本原理由两部分组成。

(1) 用一种方法划分解空间。

(2) 对一类解用一种方法预测下界，用另一种方法找到最优解的上界。如果一个解的下界超过了上界，那么这个解不是最优解，所以，应该终止与这个解相关的分支。

旅行商问题可以定义在图或平面的点上。如果将其定义在平面的点集上，那么可以使用许多方法使算法更有效。在本节中，假设问题定义在图上。为了简化讨论，假定在顶点与其本身之间没有弧，并且在每对顶点之间有代价为非负的弧。旅行商问题是要找出一条代价最小的回路，它从任意一个顶点开始，经过每一个顶点，再返回出发顶点。

考虑表5-3所示的代价矩阵。

表5-3 一个旅行商问题的代价矩阵

i \ j	1	2	3	4	5	6	7
1	∞	3	93	13	33	9	57
2	4	∞	77	42	21	16	34
3	45	17	∞	36	16	28	25
4	39	90	80	∞	56	7	91
5	28	46	88	33	∞	25	57
6	3	88	18	46	92	∞	7
7	44	26	33	27	84	39	∞

使用分支限界策略将解分成两组：一组包括某条特定的弧，另一组不包括该弧。每次划分产生一个下界，利用“更低”的下界来遍历搜索树。

首先，如前所述，如果从这个代价矩阵的任一行或列中减去一个常数，不会改变最优解。从表5-3为例，从矩阵中减去每一行的最小代价，那么减去的总数将会是旅行商问题解的下界。因此，我们可从1～7行中分别减去3，4，16，7，25，3和26，总共减去的代价为3 + 4 + 16 + 7 + 25 + 3 + 26 = 84。通过这个方法，可以得到简化的代价矩阵，如表5-4所示。

表5-4 简化的代价矩阵

i \ j	1	2	3	4	5	6	7
1	∞	0	90	10	30	6	54
2	0	∞	73	38	17	12	30
3	29	1	∞	20	0	12	9
4	32	83	73	∞	49	0	84
5	3	21	63	8	∞	0	32
6	0	85	15	43	89	∞	4
7	18	0	7	1	58	13	∞

在表5-4所示的矩阵中，每一行包含一个零。然而，一些列，即3，4和7列仍然不包含零。因此，可再从3，4和7列中分别减去7，1和4。（总的减去代价为7 + 1 + 4 = 12。）最终得到的简化矩阵如表5-5所示。

表5-5 另一个简化的代价矩阵

i \ j	1	2	3	4	5	6	7
1	∞	0	83	9	30	6	50
2	0	∞	66	37	17	12	26
3	29	1	∞	19	0	12	5
4	32	83	66	∞	49	0	80
5	3	21	56	7	∞	0	28
6	0	85	8	42	89	∞	0
7	18	0	0	0	58	13	∞

由于减去的总代价为84 + 12 = 96，我们便知道这个旅行商问题的下界为96。

接下来考虑下面的问题：假设知道一条回路包括弧4-6，其代价是零。那么，这条旅行线路的代价下界是多少呢？答案很简单：下界仍是96。

假设知道回路不包括弧4-6，那么新的下界又会是多少呢？查看表5-5，如果一条回路不包括4-6，那么肯定包括一条从4出发的其他弧。从4出发的最小代价弧是4-1，其代价是32，到终点是6的最小代价弧是5-6，其代价是零。因此，新的下界为96 + (32 + 0) = 128。由此得到的二叉树如图5-25所示。

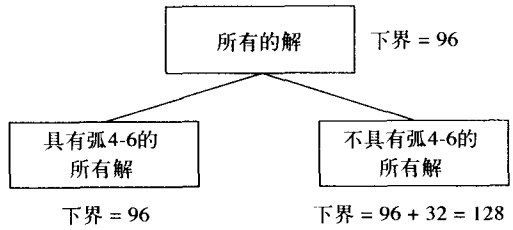


图5-25 决策树的最高层

为什么选择弧4-6来做划分？因为弧4-6能引起下界的最大增加。假如用弧3-5来做划分，那样的话，下界仅增加了1 + 17 = 18。

现在考虑左子树。在这棵子树中包含弧4-6。所以，必须从代价阵中删除第四行和第六列。而且，由于包含了弧4-6，所以弧6-4就不能再包含了，必须置$C_{6\to4}$为∞，这样得到的新矩阵如表5-6所示。

表5-6 不包含弧4-6的简化代价阵

i \ j	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	3	21	56	7	∞	28
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

再次注意到，现在第5行仍然不包含零，所以，可以对第5行减去3，相对于左子树的简化代价阵如表5-7所示，同样必须将左子树的下界也增加3（具有弧4-6的解）。

表5-7 相对于表5-6的简化代价矩阵

i \ j	1	2	3	4	5	7
1	∞	0	83	9	30	50
2	0	∞	66	37	17	26
3	29	1	∞	19	0	5
5	0	18	53	4	∞	25
6	0	85	8	∞	89	0
7	18	0	0	0	58	∞

对于右子树的代价矩阵，解不含弧4-6，只需把$C_{4\text{-}6}$设为∞。划分过程可以继续，产生的树如图5-26所示。如果沿着最小代价的路径，将会得到代价为126的可行性解。将这个代价126作为上界，许多分支可以终止，因为这些分支的下界超过此界。

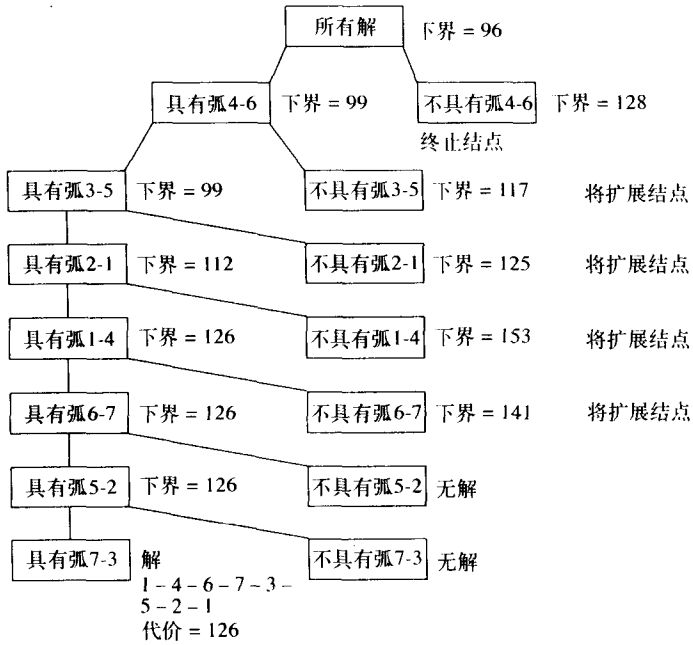


图5-26 旅行商问题的分支限界解

还有一点需要提一下。考虑所有包含弧4-6，3-5和2-1的简化代价矩阵如表5-8所示。

表5-8 简化的代价矩阵

i \ j	2	3	4	7
1	∞	74	0	41
5	14	∞	0	21
6	85	8	∞	0
7	0	0	0	∞

可以用弧1-4来划分，得到的子树如表5-9所示。

表5-9 简化的代价矩阵

i \ j	2	3	7
5	14	∞	21
6	85	8	0
7	0	0	∞

注意，已经确定将弧4-6和2-1包含在解中。再加一条弧1-4，很显然，必须避免使用弧6-2。如果使用弧6-2，就会出现一条回路，这是不允许的。所以必须设置C_{6-2}为∞。从而，将会得到左子树的代价矩阵如表5-10所示。

总之，如果已经包含路径$i_1-i_2-\cdots-i_m$和$j_1-j_2-\cdots-j_n$，并且添加一条从i_m到j_1的路径，那么不允许有从j_n到i_1的路径。

表5-10 简化的代价矩阵

i \ j	2	3	7
5	14	∞	21
6	∞	8	0
7	0	0	∞

5.8 用分支限界策略解决0/1背包问题

0/1背包问题（0/1 knapsack problem）定义如下：已知正整数P_1，P_2，…，P_n，W_1，W_2，…，W_n和M，找出X_1，X_2，…，X_n，$X_i=0$或1，$i=1$，2，…，n。满足条件

$$\sum_{i=1}^{n} W_i X_i \leqslant M$$

使得

$$\sum_{i=1}^{n} P_i X_i$$

最大。

该问题是一个NP难问题。然而，正如后面将看到的，我们仍然可用分支限界策略解决该问题。当然，在最坏情况下，就算是分支限界策略能解决此问题也需要花费指数级步数。

最初的0/1背包问题是求最大值问题，不能使用分支限界策略解决。为了解决0/1背包问题，必须将最初的0/1背包问题修改成如下的求最小值问题：已知正整数P_1，P_2，…，P_n，W_1，W_2，…，W_n和M，找出X_1，X_2，…，X_n，$X_i=0$或1，$i=1$，2，…，n。满足下面的条件

$$\sum_{i=1}^{n} W_i X_i \leqslant M$$

使得

$$-\sum_{i=1}^{n} P_i X_i$$

最小。

任何分支限界策略都需要一个分支机制。在0/1背包问题中，该分支机制如图5-27所示。第一个分支将所有的解划分成两组：一组是$X_1=0$的解，另一组是$X_1=1$的解。对于每一组，继续用X_2来划分。正如所看到的，当列举完n个X_i之后，就可以找到可行解了。

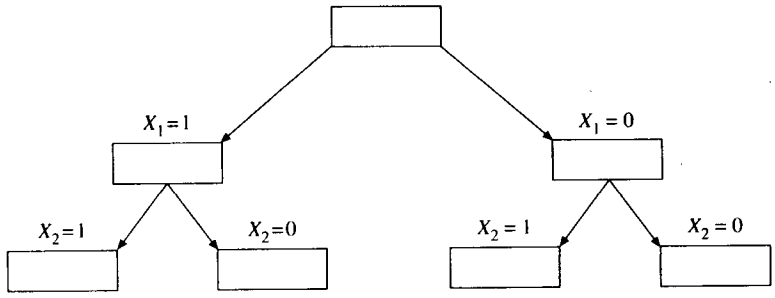


图5-27 在分支限界策略解决0/1背包问题中的分支机制

在解释分支限界策略解决0/1背包问题以前，回忆一下它是怎么解决旅行商问题的。当用分支限界策略解决旅行商问题的时候，使用了下面的基本原则：将解划分成两组，对于每一组找出一个下界。同时，尝试找到一个可行解。一旦发现可行解，都会找到一个上界。当且仅当满足以下条件之一，分支限界策略就会终止一个结点的扩展：

(1) 这个结点本身表示了一个不可行解，那么没有再扩展的意义了。

(2) 这个结点的下界大于或等于当前找出的最小上界。

现在对分支限界策略解决0/1背包问题做进一步改进。仍然将解划分成两组。对于每一组，通过找出一个可行解，不仅找出一个下界，而且还找出一个上界。当扩展一个结点的时候，希望用较小的代价找到一个解。这意味着当扩展一个结点时，希望找出一个较小上界。如果知道找出的这个上界不能再小了时（已等于其下界），就不再需要扩展这个结点。总之，当且仅当一个结点满足了以下条件之一时，就可以终止扩展了：

(1) 这个结点本身表示一个不可行解。

(2) 这个结点的下界大于或等于当前找出的最小上界。

(3) 这个结点的下界等于上界。

如何找出一个结点的上界和下界呢？下界可以认为是所能得到的最优解的值。树中的一个节点对应于构造的部分解，因此，该结点的下界对应于此部分构造解相关可能的最高利益。因为一个结点的上界意味着可行解的代价与这个部分构造解相对应。下面通过一个例子来说明我们的方法。

考虑以下的数据：

i	1	2	3	4	5	6
P_i	6	10	4	5	6	4
W_i	10	19	8	10	12	8

$M = 34$

应当注意到，$P_i/W_i \geqslant P_{i+1}/W_{i+1}$，其中$i = 1, 2, \cdots, 5$。后面将会看到这个顺序是必要的。

如何找到可行解

从最小可用的i开始向最大的i扫描，直到超过M，可行解很容易找出。例如，可以令$X_1 = X_2 = 1$，那么$\sum_{i=1}^{n} W_i X_i = 10 + 19 = 29 < M = 34$。这意味着$X_1 = X_2 = 1$是可行解。（注意，不能进一步令$X_3 = 1$，因为$\sum_{i=1}^{3} W_i X_i > 34$。）

如何找到下界

为了找到下界，回忆下界对应于所取得代价函数的最佳值。0/1背包问题是一个受约束的优化问题，因为X_i限制为是0和1。

如果放宽这个约束条件，将得到较好的结果，将该结果用作下界。可以令X_i为0到1之间的数。如果这样做，那么0/1背包问题变成定义如下的背包问题：已知正整数$P_1, P_2, \cdots, P_n, W_1, W_2, \cdots, W_n$和$M$，找到$X_1, X_2, \cdots, X_n$，$0 \leqslant X_i \leqslant 1$，$i = 1, 2, \cdots, n$。在满足下面的条件时

$$\sum_{i=1}^{n} W_i X_i \leqslant M$$

使得

$$-\sum_{i=1}^{n} P_i X_i$$

最小。

令 $-\sum_{i=1}^{n} P_i X_i$ 是0/1背包问题的最优解，而 $-\sum_{i=1}^{n} P_i X_i'$ 是背包问题的最优解。令 $Y = -\sum_{i=1}^{n} P_i X_i$，$Y' = -\sum_{i=1}^{n} P_i X_i'$。容易证明 $Y' \leqslant Y$。也就是说一个背包问题的解可作为一个0/1背包问题解的下界。

关于背包优化问题，有趣的一点是贪心算法可以找到背包问题的最优解（见习题3.11）。考虑上面提供的数据，假设已经置 $X_1 = X_2 = 1$，就不能再令 X_3 为1，因为 $W_1 + W_2 + W_3$ 会超过 M。然而，如果置 X_3 值在0与1之间，那么将得到背包问题的一个最优解，这个解是0/1背包问题的最优解的下界。X_3 的近似值如下得到：由于 $M = 34$，且 $W_1 + W_2 = 10 + 19 = 29$，那么 X_3 的最佳值应该是 $W_1 + W_2 + W_3X_3 = 10 + 19 + 8X_3 = 34$，即 $X_3 = (34-29)/8 = 5/8$。使用这个值，可以得到下界为 $-(6 + 10 + 5/8 \times 4) = -18.5$。我们使用较高的界限，因此下界为 -18。

我们考虑 $X_1 = 1$，$X_2 = 0$ 和 $X_3 = 0$ 的情况。由于 $W_1 + W_4 + W_5 = 32 < 34$ 且 $W_1 + W_4 + W_5 + W_6 = 40 > 34$，那么，通过解以下方程得到下界：

$$W_1 + W_4 + W_5 + W_6X_6 = 32 + 8X_6 = 34$$

可以得到：$W_6X_6 = 34-32 = 2$，并且

$X_6 = \frac{2}{8} = \frac{1}{4}$。这对应于下式的下界

$$-\left(P_1 + P_4 + P_5 + \frac{1}{4}P_6\right) = -\left(6+5+6+\frac{1}{4}\times 4\right) = -18$$

注意，我们找下界的方法是正确的，因为 $P_i/W_i \geqslant P_{i+1}/W_{i+1}$，且贪心算法能在此情况下正确地找到背包（而非0/1背包）问题的最优解。

如何找到上界

考虑以下情况：

$$X_1 = 1,\ X_2 = 0,\ X_3 = 0,\ X_4 = 0$$

上界对应于

$$X_1 = 1,\ X_2 = 0,\ X_3 = 0,\ X_4 = 0,\ X_5 = 1,\ X_6 = 1$$

该上界是 $-(P_1 + P_5 + P_6) = -(6 + 6 + 4) = -16$。这意味着到目前为止，我们关注这个结点，如果进一步扩展该结点，将会得到一个代价为 -16 的可行解。这就是说 -16 是这个结点上界的原因。

整个问题的解决可以用如图5-28所示的一棵树说明。每个结点上的数字表示扩展的顺序。采用最佳搜索规则，也就是说，用最好的下界扩展结点。如果两个结点有相同的下界，那么扩展具有较小上界的结点。

在图5-28所示的树中，

(1) 结点2终止是由于它的下界等于结点14的上界。

(2) 所有其他结点终止是由于局部的下界等于局部的上界。

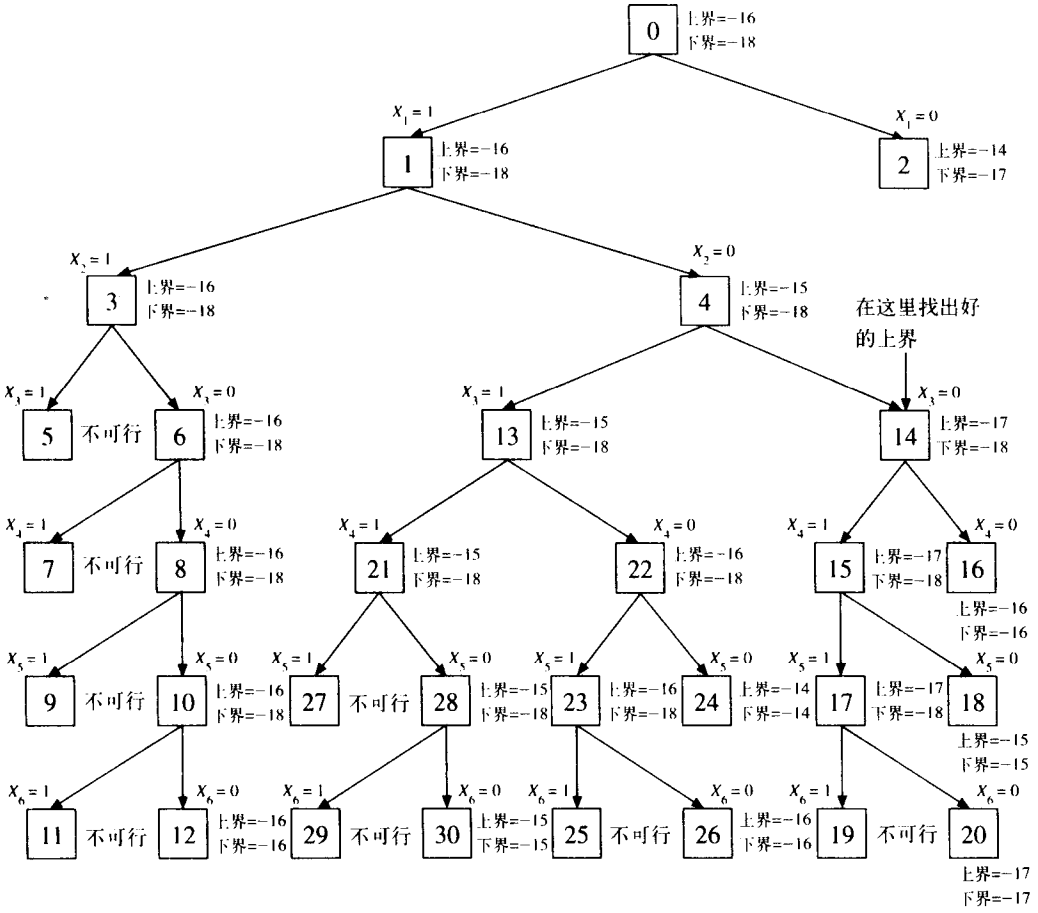


图5-28 用分支限界策略解决0/1背包问题

5.9 用分支限界方法解决作业调度问题

虽然很容易解释清楚分支限界策略的基本原则，但是并不意味着就能很有效地使用这个策略。需要设计好的分支与限界规则。在本节中，将会看到拥有聪明限界规则的重要性。

根据以下假设的作业调度问题（job scheduling problem）：

(1) 所有的处理器都是一样的，任何作业可在任一台处理器上执行。

(2) 有一个作业的偏序集。如果前一项作业存在，但是还没有执行，那么这项作业也不能执行。

(3) 只要一台处理器处于空闲状态，且某项作业是可处理的，那么该处理器必须开始执行这项作业。

(4) 每一项作业所执行的时间相同。

(5) 给定一个时间表，详细说明在每一段时隙内同时使用的处理器数量。

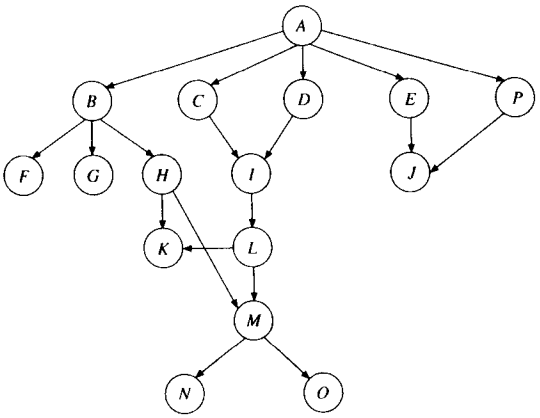


图5-29 一个作业调度问题的偏序

调度的目标是最小化最大完成时间，这个值是完成最后一项作业的时间段。

在图5-29中，已知一个偏序集。正如所看到的，作业I必须等待作业C和D，作业H等待作业B。起初，只有工作A能够立即执行。

现在考虑如表5-11所示的时间表。

表5-11 一个时间表

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
3	2	2	2	4	5	3	2	3

这个时间表说明在t = 1时刻，仅有3台处理器可以使用，在t = 5时刻，4台处理器处于活动状态。

根据图5-29的偏序和上述的时间表，可以得到两个可能的方案：

解1：	T_1	T_2	T_3	T_4	T_5	T_6	
	A	B	C	H	M	J	
	*	D	I	L	E	K	Time = 6
	*				F	N	
					P	O	
						G	

解2：	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
	A	B	D	F	H	L	M	N	Time = 8
	*	C	E	G	I	J	K	O	
	*				P	*	*		
					*	*			
						*			

显然，解1好于解2。作业调度定义如下：已知一个偏序图和一个时间表，找到完成所有作业具有最小时间步数的解。这个问题称为具有优先约束（precedence constraint）和时间表的等执行时间作业调度问题（equal execution-time job scheduling problem），它已被证明是一个NP难问题。

首先将通过树搜索技术解决该作业调度问题。假如作业偏序如图5-29所示，时间表如表5-11所示。那么，部分解树如图5-30所示。

这棵树的树顶是个空结点。从时间表中可以看出能够执行作业的最大数量是3。由于作业A是唯一的可在起始执行的作业，那么解树的第一层仅由一个结点构成。在A执行完后，从时间表中可以看出，有两项工作现在可以执行，在图5-30中解树的第二层显示了所有可能的组合。由于空间所限，我们仅将结点的后代作为解树结点的子集。

在上面的图中，看到如何通过树搜索技术来解决具有时间表的作业调度问题。在下面几节中，将会看到为了使用分支限界策略，需要设计出好的分支与限界规则，以便能够避免搜索整棵解树。

以下四条规则是分支限界方法所采用的，这里不证明这些规则的有效性，仅对它们做非正式地描述。

规则1：共同后继效应

这个规则可再次对图5-30做非正式的解释。在此情况下，解树的根是只有一项作业构成的结点A。该结点将有很多直接的子孙结点，其中的两个是(C，E)和(D，P)。如图5-29所示，作

业C和D共有同一个子孙I。类似地，E和P也有同一个子孙J。在这种情况下，规则1保证了仅有一个结点，要么是(C，E)要么是(D，P)需要在解树中扩展。因为任何一个由结点(C，E)开头的最优解的长度都与从结点(D，P)开头的一样。

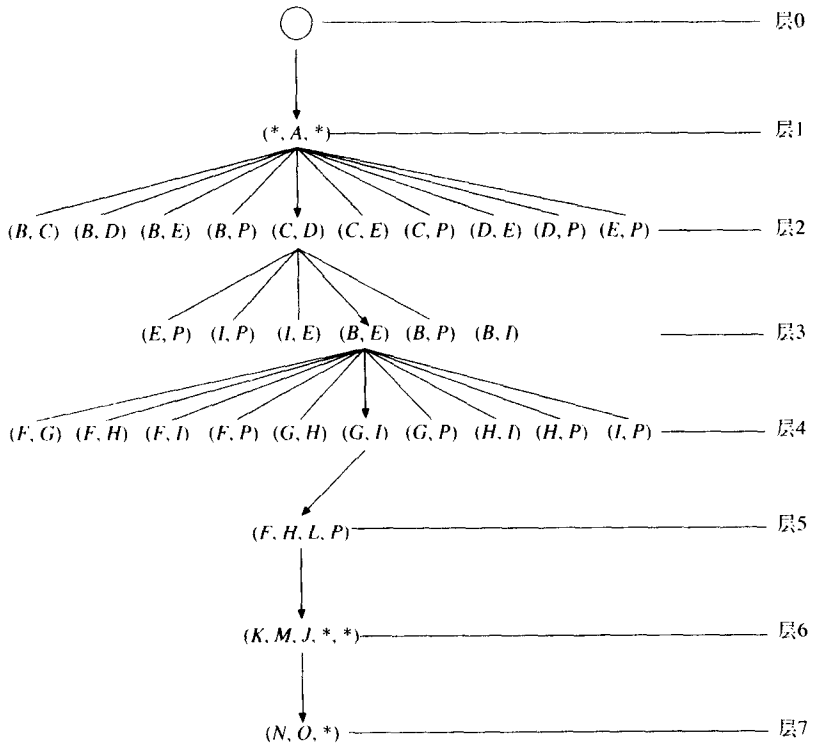


图5-30 部分解树

* 表示一台空闲处理器

为什么能做出这个结论呢？考虑一个从(C，E)出发的可行解，在某处有作业D和P。由于C和D有同样的子孙I，那么可以在不改变解可行性的前提下交换C和D，依据类似的理由，也可以交换P和E。因此，以(C，E)开头的可行解可以转换成以(D，P)为头的可行解，而不会改变解的长度。由此看出，规则1是有效的。

规则2：内部结点优先策略

作业优先图的内部结点将会比叶子结点优先处理。

这个规则可以图5-29的作业优先图和表5-12中的时间表来做非正式地解释。

表5-12 一个时间表

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
1	3	2	3	2	2	3	2	3

在这种情况下，图5-31中显示了解树的一部分。在扩展解树中的结点(B，E，P)之后，得到结点(B，E，P)的许多直接后继孩子，其中两个是(C，D)和(F，J)，如图5-31所示。

在作业优先图中，由于作业C和D是内部结点，而F和J是叶子结点，规则2表明交换结点(C，D)和终端结点(F，J)，类似于用于解释规则1时的理由。

规则2表明候选集应该划分成两个子集。一个是活动的内部结点，另一个是活动的叶子结点。前者具有处理的较高优先级。由于此集合的规模较小，可以减少可能选择的次数。只有当活动的内部结点集合小于活动的处理器数量时，要选择活动的叶子结点集。由于叶子结点没有后继，如何选择它们将没有差别，可以选择任一组，如图5-30所示，在遍历了第三层结点(*B*，*E*)后，在当前候选集中有五项作业(*F*，*G*，*H*，*I*，*P*)，而有两台处理器处于活动状态，所以，可以产生共10种可能的组合。但是，如果仅考虑内部结点，那么只产生3个结点。它们是(*H*，*I*)(*H*，*P*)和(*I*，*P*)，而其他7个结点将不再生成。

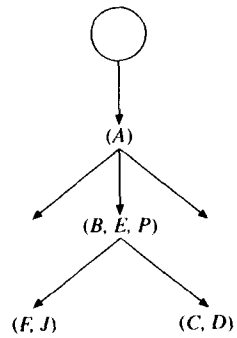
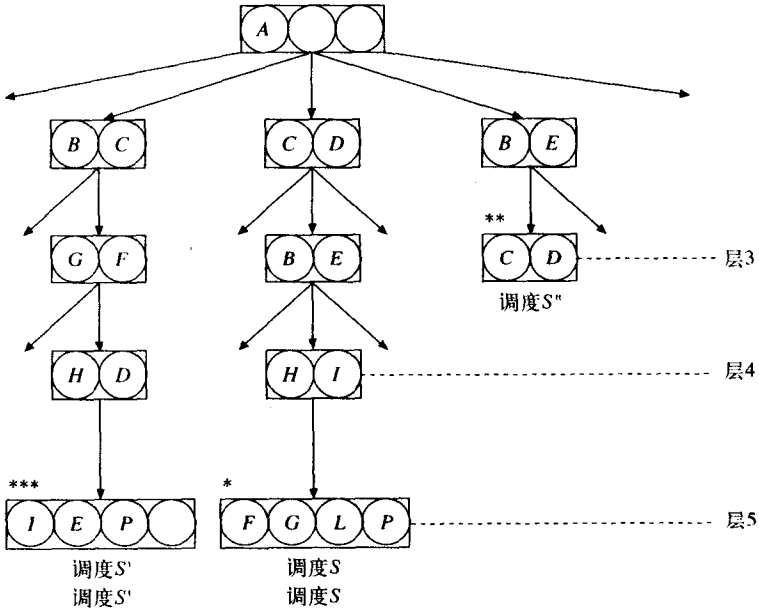


图5-31 部分解树

规则3：最大化处理作业的数量

在从时段1到时段*i*的部分或完全扩展的调度*S*中，令*P*(*S*，*i*)表示已经处理的作业集。规则3说明如下：

对另外的某个调度*S'*，如果*P*(*S*，*i*)包含在*P*(*S'*，*i*)中，那么，这个调度*S*不是最优解。规则3显然是正确的。如图5-32所示，规则3是如何终止解树中的一些结点。对于图5-32，可以断言从结点“***”和从“**”的调度不比从“*”开始得好。这是由于*P*(*S*"，3) = *P*(*S*，3)，而*P*(*S'*，5)包含在*P*(*S*，5)中。



$$\begin{cases} P(S'', 3) = (A, B, C, D, E) \\ P(S, 3) = (A, B, C, D, E) \end{cases}$$

$$P(S'', 3) = P(S, 3)$$

$$\begin{cases} P(S', 5) = (A, B, C, D, E, F, G, H, I, P) \\ P(S, 5) = (A, B, C, D, E, F, G, H, I, P, L) \end{cases}$$

P(*S'*，5)包含在*P*(*S*，5)中）

图5-32 处理作业影响

规则4：累积空闲处理器策略

规则4说明如下：

具有比可行调度解多的累积空闲处理器数的部分调度安排不会比此可行解好，所以可被终止。

规则4说明如果已经找到了可行解，就可以使用所有空闲的处理器终止解树的其他结点。

考虑图5-33，解树中的结点(K，M，J)拥有的累积空闲处理器数是4，这比当前可行解的处理器数还要多。所以，由规则4可知结点(K，M，J，*，*)可被限制。

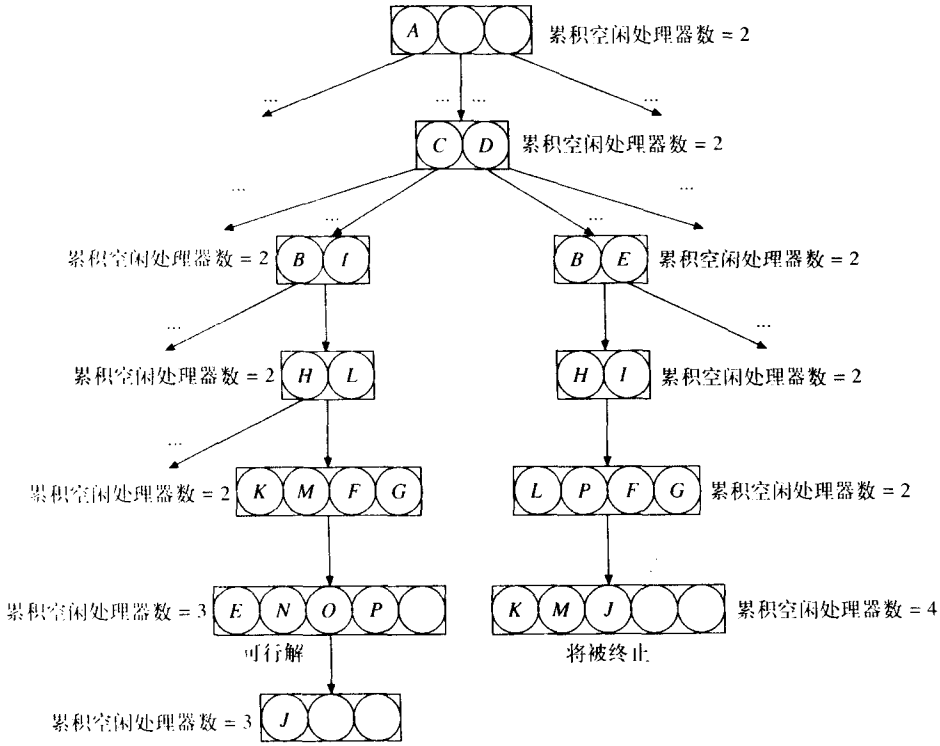


图5-33 积累的空闲处理器效应

5.10 A*算法

A^*算法（A^* algorithm）是一种很好的树搜索策略，很受人工智能（artificial intelligence）研究者的喜欢，但非常遗憾的是它经常被算法研究者所忽视。

先来讨论隐藏在A^*算法背后的哲学，最好的方法是把它与分支限界策略进行比较。注意，在分支限界策略中，主要的工作是确定不必进一步考查许多解，因为它们不会最终生成最优解。因此，在分支限界策略中主要的技巧是在限界。

A^*算法强调另一个点。它将告诉我们在特定情况下，已得到的一个可行解肯定是一个最优解。这样，我们可以停止。当然，我们希望这个终止出现在比较早的阶段。

A^*算法通常用于解决优化问题。它使用最佳优先（最少代价优先）策略（best-first（least-cost-first）strategy）。A^*算法关键的因素是代价函数（cost-function），参见图5-34。要找到从S到T的最短路径，假定使用某种树搜索算法来解决此问题。解树的第一步显示在图5-35中。结点A与选择V_1（边a）的决策有关。通过选择V_1，当前的代价至少是2，因为边a的代价值是2。令$g(n)$表示从决策树的根到结点n的路径长度，$h^*(n)$表示从结点n到目标结点的最优路径长度，那么结点n的代价是$f^*(n) = g(n) + h^*(n)$。对于图5-35中的树，$g(A) = 2$，$g(B) = 4$，$g(C) = 3$。问题是：$h^*(n)$的值是多少？注意从V_1开始到T结束有许多路径。

下面是所有的这些路径：

$V_1 \rightarrow V_4 \rightarrow T$ (路径长度 = 5)
$V_1 \rightarrow V_2 \rightarrow V_4 \rightarrow T$ (路径长度 = 8)
$V_1 \rightarrow V_2 \rightarrow V_5 \rightarrow T$ (路径长度 = 10)
$V_1 \rightarrow V_4 \rightarrow V_5 \rightarrow T$ (路径长度 = 8)

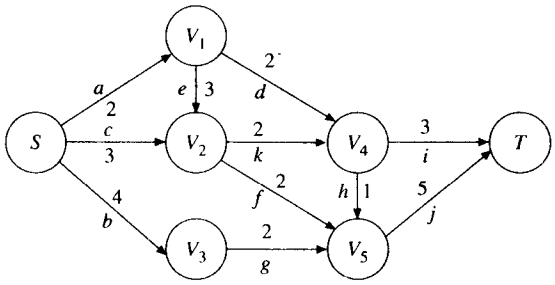


图5-34 说明A^*算法的图

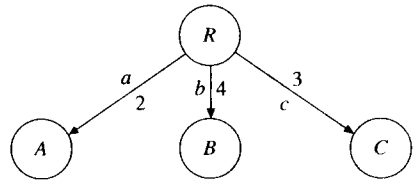


图5-35 解树的第一层

如果从V_1开始，那么最优路径是$V_1 \rightarrow V_4 \rightarrow T$，路径长度是5。因此，$h^*(A)$等于5，这是$V_1 \rightarrow V_4 \rightarrow T$的路径长度。通常情况下，$h^*(n)$对于我们是未知的，所以一般地$f^*(n)$也是未知的。$A^*$算法假定仍然能估计$h^*(n)$。再次参见图5-35，结点$A$对应于图5-34中的$V_1$。从$V_1$开始有两条可能的路线，到$V_2$，或者到$V_4$。比较短的一条路线是访问$V_4$，其代价是2。这意味着，从结点$A$开始，它的最短路径长度至少是2。因此，$h^*(n)$至少是2。现在可以标记$h(n)$是$h^*(n)$的一个估计。

尽管有许多方法估计$h^*(n)$，*但A^*算法保证总是使$h(n) \leqslant h^*(n)$。也就是说，应该使用$h^*(n)$的相当保守的估计。这意味着，在A^*算法中，总是使用$f(n) = g(n) + h(n) \leqslant g(n) + h^*(n) = f^*(n)$作为代价函数。*后面在说明一个完整的例子后这个概念将变得很清楚。

最后介绍A^*算法一个很重要的性质。注意到A^*算法采用最佳优先规则，这意味着在所有将扩展的结点中，选择具有最小代价的结点作为下一个扩展结点。A^*算法具有下列终止规则(termination rule)：*如果被选择的结点也是目标结点，那么这个结点代表一个最优解并且过程可被终止。*

现在，解释为什么上述的规则是正确的。令t被选择为目标结点，n表示一个已经扩展的结点。

(1) 因为A^*算法使用最少代价优先规则，所以，对于所有的n都有$f(t) \leqslant f(n)$。

(2) 因为A^*算法使用$h^*(n)$的保守估计，所以，对于所有n都有$f(t) \leqslant f(n) \leqslant f^*(n)$。

(3) 但是$f^*(n)$之一肯定是一个最优解。事实上，令$f^*(s)$代表一个最优解的值，那么可以得到

$$f(t) \leqslant f^*(s)$$

(4) 第3条陈述表明在任何时间，对于任何选择用于扩展的结点，它的代价函数将不超过最优解的值。因为t是一个目标结点，所以有$h(t) = 0$，并且

$$f(t) = g(t) + h(t) = g(t)$$

因为$h(t)$为零，所以，

$$f(t) = g(t) \leqslant f^*(s)$$

(5) 然而，$f(t) = g(t)$是一个可行解的值。结果，$g(t)$不能比$f^*(s)$小。根据定义，这意味着$g(t) = f^*(s)$。

我们已经阐述了A^*算法。现在，总结A^*算法的基本规则如下：

(1) A^*算法采用最佳优先（或最少代价优先）规则。换句话说，在所有将被扩展的结点中，具有最小代价的结点将是下一个被扩展的结点。

(2) 在A^*算法中，代价函数$f(n)$定义如下：

$$f(n) = g(n) + h(n)$$

其中$g(n)$是从树根到结点n的路径长度。$h(n)$是$h^*(n)$的估计值，也就是从n到某个目标结点的最佳路径长度。

(3) 对于所有的n，都有$h(n) \leqslant h^*(n)$。

(4) 当且仅当被选择的结点也是目标结点时，A^*算法将停止，然后它返回一个最优解。

当然，A^*算法允许使用其他的树搜索规则。例如，参见图5-36，如果找出从S到T的最短路径，那么可以观察到从S到V_3有两条路径：

$S \xrightarrow{2} V_1 \xrightarrow{3} V_3$，

$S \xrightarrow{3} V_2 \xrightarrow{4} V_3$

因为第二条路径的长度比第一条长，所以，第二条路径将被忽略，因为它决不会产生最优解。

这称为优势规则（dominance rule），用于动态规划设计中。在本书的后面将介绍动态规划设计。读者也许注意到，如果使用优势规则，路径

$$S \rightarrow V_1 \rightarrow V_2$$

也将被忽略。

类似地，在分支限界策略中所使用的限界规则也能在A^*算法中使用。

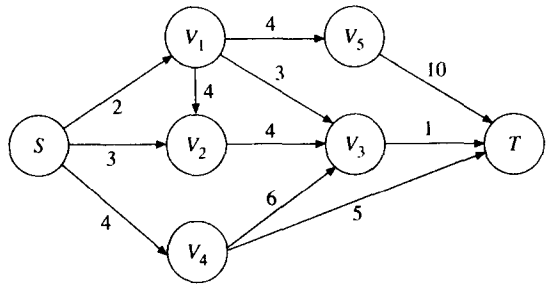
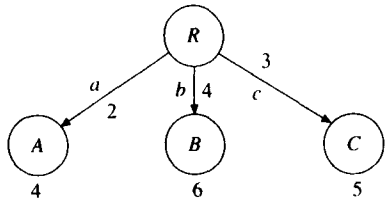


图5-36 阐述优势规则的图

现在说明在图5-34中A^*算法是如何找到一条最短路径的。步骤1到7展示了完整的过程。

步骤1. 扩展结点R

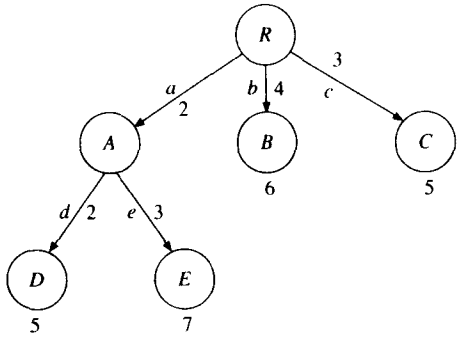


$g(A) = 2$　　$h(A) = \min\{2, 3\} = 2$　　$f(A) = 2 + 2 = 4$

$g(B) = 4$　　$h(B) = \min\{2\} = 2$　　$f(B) = 4 + 2 = 6$

$g(C) = 3$　　$h(C) = \min\{2, 2\} = 2$　　$f(C) = 3 + 2 = 5$

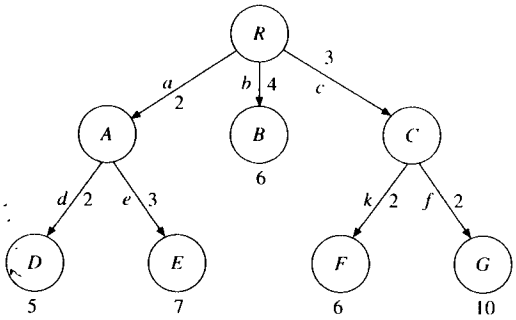
步骤2. 扩展结点A



$g(D) = 2 + 2 = 4$　　$h(D) = \min\{3, 1\} = 1$　　$f(D) = 4 + 1 = 5$

$g(E) = 2 + 3 = 5$　　$h(E) = \min\{2, 2\} = 2$　　$f(E) = 5 + 2 = 7$

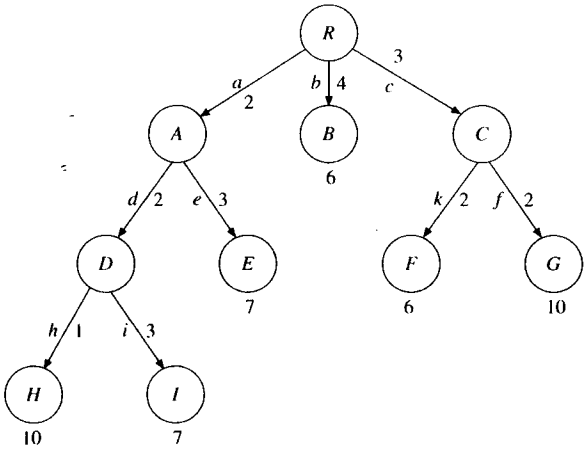
步骤3. 扩展结点C



$g(F) = 3 + 2 = 5$　　$h(F) = \min\{3, 1\} = 1$　　$f(F) = 5 + 1 = 6$

$g(G) = 3 + 2 = 5$　　$h(G) = \min\{5\} = 5$　　$f(G) = 5 + 5 = 10$

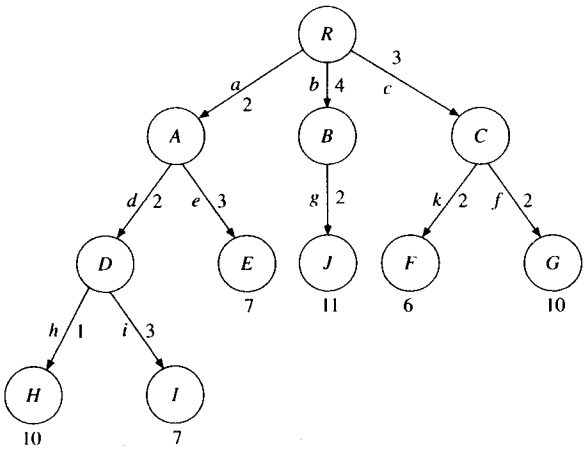
步骤4. 扩展结点D



$g(H) = 2 + 2 + 1 = 5$　　$h(H) = \min\{5\} = 5$　　$f(H) = 5 + 5 = 10$

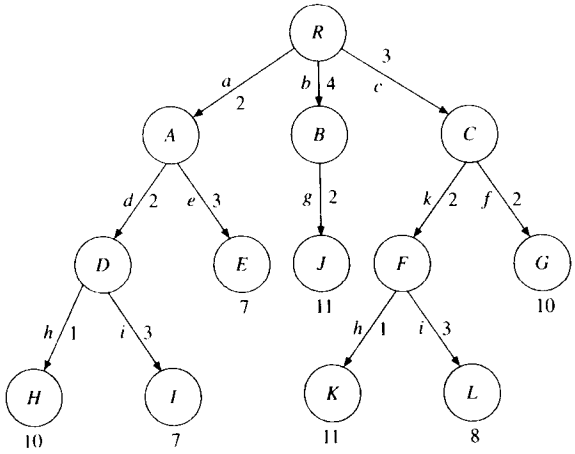
$g(I) = 2 + 2 + 3 = 7$　　$h(I) = 0$　　$f(I) = 7 + 0 = 7$

步骤5. 扩展结点B



$g(J) = 4 + 2 = 6$　　$h(J) = \min\{5\} = 5$　　$f(J) = 6 + 5 = 11$

步骤6. 扩展结点F



$g(K) = 3 + 2 + 1 = 6$　　$h(K) = \min\{5\} = 5$　　$f(K) = 6 + 5 = 11$

$g(L) = 3 + 2 + 3 = 8$　　$h(L) = 0$　　$f(L) = 8 + 0 = 8$

步骤7. 扩展结点I

因为I是目标结点，所以停止并将以下作为一个最优解返回。

$$S \xrightarrow[2]{a} V_1 \xrightarrow[2]{d} V_4 \xrightarrow[3]{i} T$$

也许下面的问题是很重要的：能将A^*算法看作一种代价函数具有聪明设计的分支限界策略吗？回答是肯定的。当A^*算法终止时，本质上我们断言现在所有其他的扩展结点同时被这个可行性解限制。例如，看上例中的步骤6，结点I相对于一个代价为7的可行解。这意味着已经找到了这个问题实例的一个上界，就是7。然而，所有其他的结点的代价都大于或等于7。这些代价也都是下界，这就是为什么能通过使用这个上界剪去所有其余结点来终止这个过程。

分支限界策略是一个通用的策略。它既不指定代价函数也不确定选择结点进行扩展的规则。A^*算法指定代价函数$f(n)$，它确实就是这个结点的下界。A^*算法也指定了结点的选择规则是最少代价规则。只要到达一个目标结点，那么上界就确定了。假如这个目标结点较晚扩展，因为使用的是最少代价优先规则，所以这个上界肯定小于或等于其他结点的代价。又因为每个结点的代价是该结点的下界，那么这个上界小于或等于所有其他下界。因此，这个上界肯定是一个最优解的代价。

5.11 用特殊的A*算法解决通道路线问题

本节将介绍通道路线问题（channel routing problem），这个问题起源于非常大的集成计算机辅助设计（integrated computer aided design）系统。我们将介绍使用特定的A^*算法完美地解决这个问题。在这种特殊的A^*算法中，注意下列重要的问题：只要找出一个目标结点，就可停止并且返回一个最优解。注意这种做法在普通A^*算法中是错误的，因为在普通的A^*算法中，当找出一个目标结点，不能终止算法，只有在一个目标结点被选择用来扩展时才终止算法。

这种特殊的A^*算法可通过图5-37加以解释。在这个算法中，无论什么时候选择结点t，并且产生一个目标结点作为它的直接后继。那么，$h(t) = h^*(t) = g(goal) - g(t)$。在这个条件下，

$$f(goal) = g(goal) + h(goal) = g(t) + h^*(t) + 0 = f^*(t)$$

$$g(goal) = f(goal) = f^*(t)$$

注意，t是为了扩展而选择的结点。因此，假如n是一个扩展结点，因为使用最佳优先策略，那么$f^*(t) \leqslant f(n)$。根据定义，$f(n) \leqslant f^*(n)$，因此，$g(goal) = f^*(t) \leqslant f^*(n)$，首先发现的目标结点肯定是一个最优解。

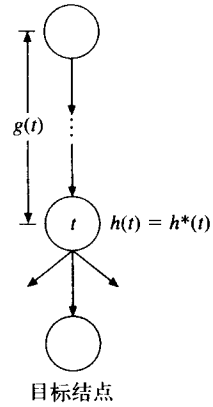


图5-37 A*算法应用的一个特殊情形

正如所期望的，函数$h(n)$必须仔细地设计，否则将不能获得好的结果。将会令读者惊讶的是对于本节定义的通道路线问题，$h(n)$能容易地设计以至于当应用A^*算法首先发现的目标结点，或者用另一种方式表述，第一个叶子结点表示一个最优解。

现在阐述通道路线问题，参见图5-38。有一个通道和两行终端，一行在顶端，另一行在底部还有一套标记从1到8的网络集。例如，网7将连接顶端的终端4与底部的终端3。类似地，网8将连接顶端的终端2与底部的终端8。当连接这些终端时，不允许出现如图5-39所示的非法连接。

有许多连接终端的方法。其中的两种方式分别如图5-40和图5-41所示。对于图5-40中的设计，有7条轨迹，而图5-41所示只有4条轨迹。事实上，图5-41中的轨迹数是最少的轨迹数。

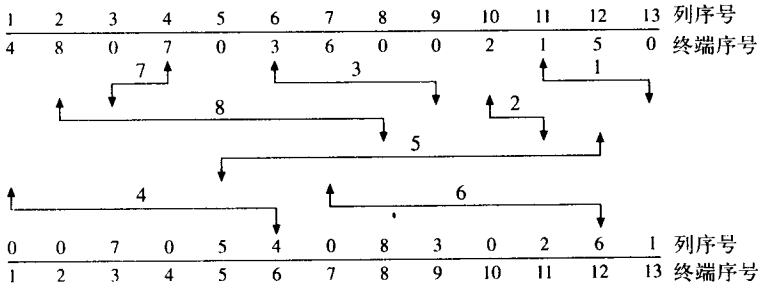


图5-38 一个特殊的通道

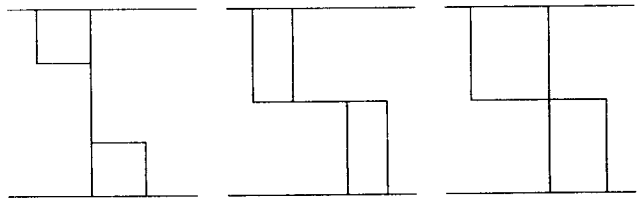


图5-39 非法的配线

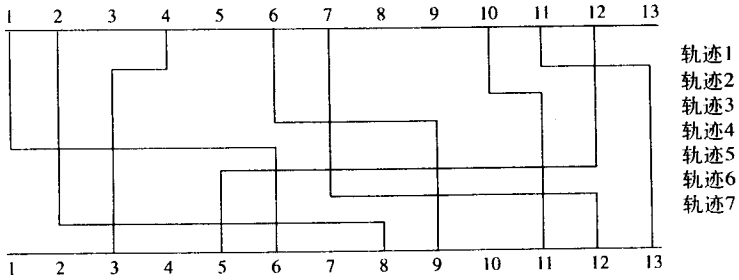


图5-40 一种可行的设计

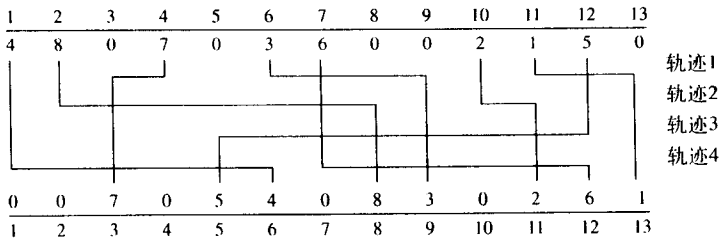


图5-41 一种最优设计

通道路线问题是发现一个有最少数目轨迹的设计，这已经被证明是一个NP难问题。为了设计一个解决此问题的A^*算法，首先观察网络必须服从两个约束条件：水平约束和垂直约束。

水平约束

水平约束（horizontal constrains）可以通过如图5-42所示的水平约束图（horizontal constraint graph）来解释。通过参考图5-38，注意到，如果它们在同一个轨迹中，例如网7，必须配置到网3、网5和网6的左边。类似地，网8必须配置到网1和网2的左边。这些相关联系概括在如图5-42所示的水平约束图中。

垂直约束

垂直约束（vertical constraints）也概括在垂直约束图（vertical constraint graph）中，如图5-43所示。

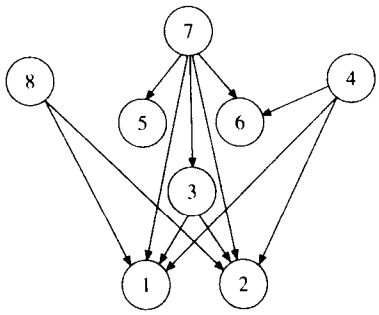


图5-42 水平约束图

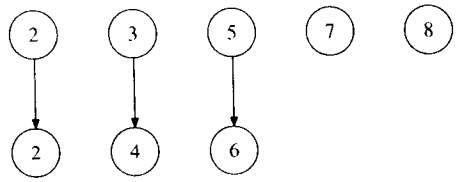


图5-43 垂直约束图

再次参见图5-38，注意到当网1和网2共享相同的终端11时，网1必须在网2之前完成。类似地，网3必须在网4之前连线。

因为在本书中仅关心如何应用A^*算法，所以不打算深入解决通道路线问题的细节。在此有许多步骤将不详细地解释，因为它们对于理解A^*算法是不相关的。

因为垂直约束图的存在，这本质上是定义了一个偏序，仅那些在垂直约束图中没有前驱的网能分派给任何轨迹。例如，最初只有网1、3、5、7和8可以分派一个轨迹。假如1和3已经分派，那么2和4才能分派。

当垂直约束图给出哪些网能分派的信息时，水平约束图也提供给我们哪些网能分派给一个轨迹的信息。因为我们感兴趣的是介绍A^*算法，目的不是去解决这个通道路线问题，所以将仅在概念上而不是原理上介绍一些操作。参见图5-43，注意到网1、3、5、7和8可以分派。参考图5-42，注意到在网1、3、5、7和8中，有三个最大团（clique）：{1，8}、{1，3，7}和{5，7}。（一个图的团是一个子图，在这个子图中每一对顶点连接，一个最大团是指它的规模不能再扩大的团。）可以证明，每个最大团可以分派一个轨迹。读者可以通过参考图5-38来证明这一点。

使用这些规则，我们可以用树搜索方法解决通道路线问题，树的第一层如图5-44所示。

为了进一步扩展这些子树，考虑结点(7，3，1)。参考图5-43所示的垂直约束图，网1、3和7分派后，能被分派的网变成2、4、5和8。类似地，假如网1和8被分派，那么能被分派的网变成2、3、5和7。对于结点(7，3，1)来说，下一个能被分派的网集合是{2，4，5，8}。再一次参考水平约束图5-42，注意到在2、4、5和8中，有三个最大团，即{4，2}、{8，2}和{5}。这样，假如仅扩展结点{7，3，1}，其解如图5-45所示。

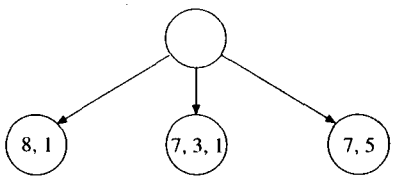


图5-44 解决通道路线问题的树的第一层

关键问题是：用树搜索方法解决通道路线问题的代价函数是什么？注意对于A^*算法，需要两个代价函数：$g(n)$和$h(n)$。$g(n)$可以简单定义，因为树的层次准确对应于轨迹数，$g(n)$能方便地定义成树的层次。对于$h(n)$，能通过必须使用的最少轨迹数来估计，这个最少轨迹数肯定在已经分派的轨迹数目确定后得到。

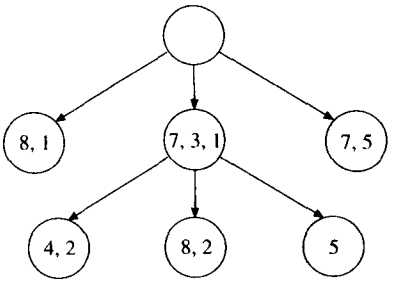


图5-45 图5-44进一步扩展的树

参见图5-38，注意到对于每个终端，能画一条垂线。假如这条垂线与k条水平线相交，那么需要的轨迹最小数目是k。例如，对于终端3，最小轨迹数是3，而对于终端7，最小轨迹数是4。这称为终端i的本地密度函数，整个问题的密度函数是最大的本地密度函数（local density function）。对于图5-38中所示的问题，其密度函数是4。在一些网被分派给轨迹后，密度函数也会改变。例如，网7、3和1被分派后，密度函数变为3。现在使用这个密度函数作为$h(n)$。图5-46所示的是A^*算法如何使用代价函数工作。

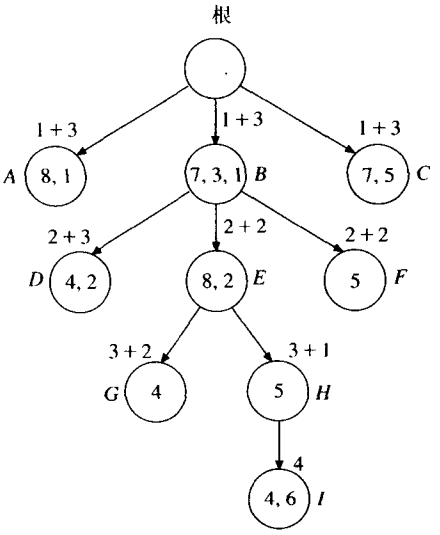


图5-46 使用A^*算法解决通道路线问题的部分解树

容易看到，对于使用的A^*算法，$h(t) = h^*(t) = 1 = g(goal) - g(t)$。因为它需要至少1个轨迹去完成它的任务。因此，结点I肯定代表一个最优解。

这个例子显示如果能设计一个好的代价函数的话，解决一些组合爆炸问题，A^*算法是一个非常好的策略。

5.12 用A*算法解决线性分块编码译码问题

设想用二进制代码发送从0到7这八个数字，每个数字需要3位。例如，0用000发送，4用100发送，而7用111发送。假如有任何错误，接收到的信号将错误译码。例如，对于100来说，如果它被发送并且被接收成000，将造成一个大的错误。

相反地，设想使用6位来代替3位编码这些数字。参见表5-13所示的编码。右边的编码称为码字（code word）。每个数字的二进制形式通过相应的编码形式发送。也就是，100用100110发送，101用101101发送。现在这种方式的优点是显而易见的。把接收到的一个矢量译码成一个码字时，它的汉明距离（Hamming distance）在所有码字中最短。假定码字000000被作为000001发送，那么能容易地看到在000001和000000之间汉明距离在0000001与所有的八个编码中是最短的。由此，译码过程将0000001译码为000000。换句话说，通过增多位数，可以冗余掉更多的错误。

表5-13 码字

000	000000	110	110011
100	100110	101	101101
010	010101	011	011110
001	001011	111	111000

在本书不讨论代码是怎样产生的，关于这些内容在编码理论的书中都有介绍。我们仅假定码字已经存在，我们的工作是译码。在下面的例子中，这个编码方案称为线性分块编码(linear block code)。

实际上，我们不直接发送0和1。在本节中，假定1(0)被作为−1(1)发送。也就是，110110在发送时作为(−1，−1，1，−1，−1，1)发送。接收矢量$r = (r_1, r_2, \cdots, r_n)$和一个码字$(c_1, c_2, \cdots, c_n)$之间的距离是

$$d_E(r, c) = \sum_{i=1}^{n}(r_i - (-1)^{c_i})^2$$

假定有$r = (-2, -2, -2, -1, -1, 0)$和$c = 111000$。那么它们之间的距离是

$$\begin{aligned} d_E(r, c) &= (-2-(-1)^1)^2 + (-2-(-1)^1)^2 + (-2-(-1)^1)^2 + (-1-(-1)^0)^2 \\ &\quad + (-1-(-1)^0)^2 + (0-(-1)^0)^2 \\ &= 12 \end{aligned}$$

为了译码一个接收到的矢量，简单地计算这个矢量和所有码字之间的距离，并且把这个矢量翻译成与其距离最短的特殊码字。实际上，码字的数目超过10^7，这是很大的数目。任何穷举所有码字的搜索都是不可能的。

我们使用一棵码树（code tree）代表所有码字。例如，用图5-47所示的码树表示表5-13中所示的所有码字。

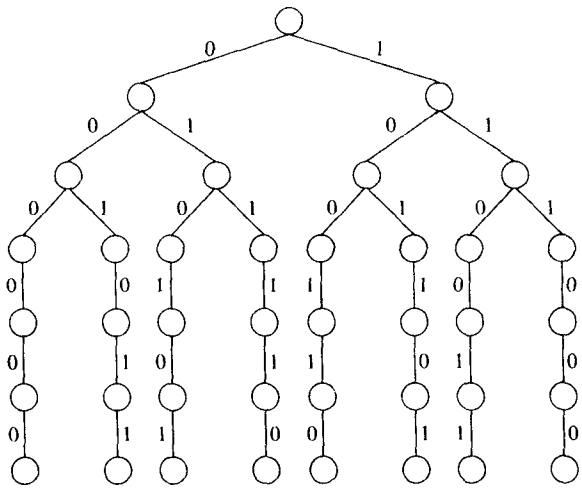


图5-47 一棵码树

译码接收到的矢量，找出一个最接近这个接收字的码字，现在变成一个树搜索问题。这个问题可形式化描述如下：找出从码树的根到目标结点的一条路径，使得这条路径的代价是从根到目标结点的所有路径中距离最短的，而一条路径的代价是在路径中经过的所有分支的代价和。从在$t-1$层的一个结点到t层的一个结点的分支代价是$(r_t - (-1)^{c_t})^2$。我们将说明A^*算法是解决这

个问题的有效方法。

令这棵码树的根层次是−1。n是层t上的一个结点，函数$g(n)$定义如下：

$$g(n)=\sum_{i=0}^{t}(r_i-(-1)^{c_i})^2$$

其中$(c_1, c_2, \cdots, c_t)$是从根到结点n的路径中相关分支的标记。定义启发函数$h(n)$如下

$$h(n)=\sum_{i=t+1}^{n-1}(|r_i|-1)^2$$

容易看到，对于每个结点n都有$h(n) \leqslant h^*(n)$。

参见图5-47中的码树，令接收的矢量是(−2，−2，−2，−1，−1，0)，应用A^*算法的译码过程在图5-48中阐述。

在译码开始，扩展根结点并计算两个新产生的结点2和3。$f(2)$的值计算如下：

$$\begin{aligned} f(2) &= (-2-(-1)^0)^2+h(2) \\ &= 9+\sum_{i=1}^{5}(|r_i|-1)^2 \\ &= 9+1+1+0+0+1 \\ &= 12 \end{aligned}$$

$f(3)$的值可以用同样的方法计算。结点10是目标结点，当它被选择扩展时，过程将中止，找出的最接近的码字是111000。

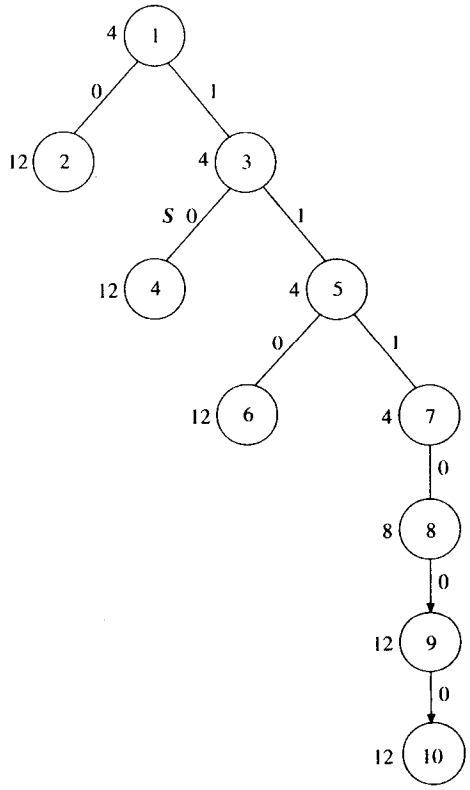


图5-48 一棵解树的扩展

5.13 实验结果

在平均情况下，通过树搜索技术，实验中的许多NP完全问题能有效地解决。考虑0/1背包问题，这是一个NP完全问题。在一台IBM PC机上，用分支限界方法解决了这个问题。测试的目的是在平均情况下性能是否指数级的，表5-14总结了测试结果。对于每个元素的数目n，使用一个随机数生成器产生5套数据。在每一套数据中，P_i和W_i的值都在1到50之中。整数M按如下方法确定。令W是所有权值的总和，那么M设置成$W/4$，$(W/4)+20$，$(W/4)+40$，…。当然M不应该超过W。那么，对于每个n，找出解决所有问题实例集的平均时间。

很明显，在分支限界策略基础上用这个算法解决0/1背包问题的平均性能远低于指数。实际上如图5-49所示，它的性能几乎是$O(n^2)$。通过这些试验结果读者应该受到鼓舞，不要害怕NP完全问题，有许多有效的算法解决这样的问题，当然是在平均情况下。

表5-14 使用分支限界策略解决0/1背包问题的测试结果

n	平均时间	n	平均时间
10	0.025	50	0.120
20	0.050	60	0.169
30	0.090	70	0.198
40	0.110	80	0.239

（续）

n	平均时间	n	平均时间
90	0.306	200	1.167
100	0.415	210	1.679
110	0.441	220	1.712
120	0.522	230	1.972
130	0.531	240	2.167
140	0.625	250	2.302
150	0.762	260	2.495
160	0.902	270	2.354
170	0.910	280	2.492
180	1.037	290	2.572
190	1.157	300	3.145

5.14 注释与参考

深度优先、广度优先和最小代价优先搜索技术可在文献Horowitz and Sahni(1976)和Knuth (1973)中找到。对于A^*算法，可参阅文献Nilsson (1980)和Perl (1984)。分支限界策略的完美回顾可在文献Lawler and Wood (1966)和Mitten (1970)中找到。

人员分配问题首先在文献Ramanan, Deogun and Liu(1984)中讨论。文献Liang (1985)说明了通过分支限界方法解决这个问题。使用分支限界方法解决旅行商问题，可参阅文献Lawler、Lenstra、Rinnooy Kan and Shmoys (1985)和Little, Murty, Sweeney and Karel (1963)。使用分支限界方法解决旅行商问题也出现在文献Yang, Wang and Lee(1989)中。文献Wang and Lee (1990)提出的A^*算法是解决通道路线问题的一项好技术。A^*算法应用于线性分块编码问题可在文献Ekroot and Dolinar (1996)；Han, Hartmann and Chen(1993)和Han, Hartmann and Mehrotra (1998)中找到，相关的工作也可文献Hu and Tucker(1971)中找到。

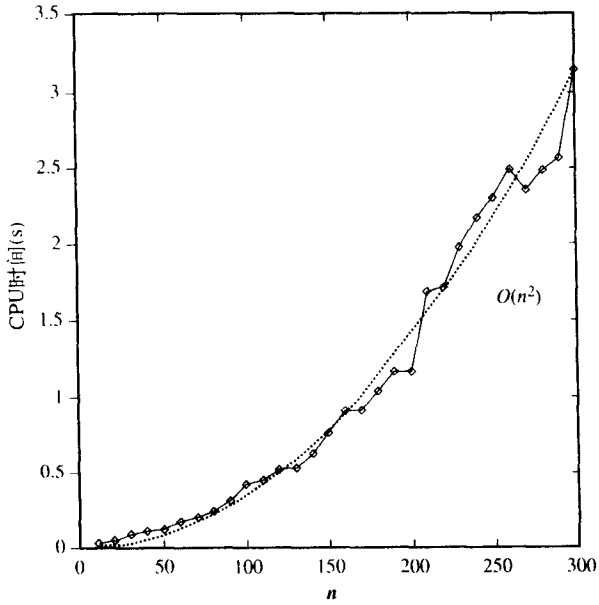


图5-49 使用分支限界策略解决0/1背包问题的试验结果

5.15 进一步的阅读资料

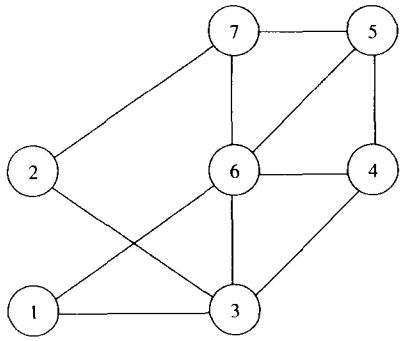
树搜索策略是十分自然并且便于应用。对于树搜索算法的平均情况分析，我们推荐文献Brown and Purdom (1981)；Huyn, Dechter and Pearl (1980)；Karp and Pearl (1983)；还有Purdom and Brown (1985)。对于分支限界算法，我们推荐文献Boffey and Green (1983)；Hariri and Potts (1983)；Ibaraki (1977)；Sen and Sherali (1985)；Smith (1984)；还有Wah and Yu(1985)。对于A^*算法，我们推荐文献Bagchi and Mahanti (1983)；Dechter and Pearl (1985)；Nau, Kumar and Kanal (1984)；Pearl (1983)；以及Srimani (1989)。

关于最近的研究成果，可参阅文献Ben-Asher, Farchi and Newman (1999)；Devroye

(2002)；Devroye and Robson (1995)；Gallant，Marier and Storer (1980)；Giancarlo and Grossi (1997)；Kirschenhofer，Prodinger and Szpankowski (1994)；Kou，Markowsky and Berman (1981)；Lai and Wood (1998)；Lew and Mahmoud (1992)；Louchard，Szpankowski and Tang (1999)；Lovasz，Naor，Newman and Wigderson (1995)；and Meleis (2001)。

习题

5.1 在下面的图中，使用某种树搜索技术找出一条哈密顿回路。



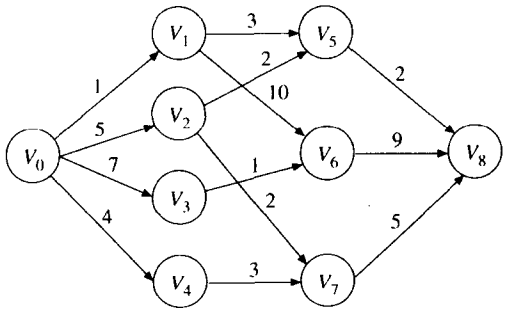
5.2 从下面的初始状态开始测试，解决8数码问题。

2	3	
8	1	4
7	5	6

注意最终的目标是

1	2	3
8		4
7	6	5

5.3 用分支限界策略找出下图从v_0到v_8的最短路径。

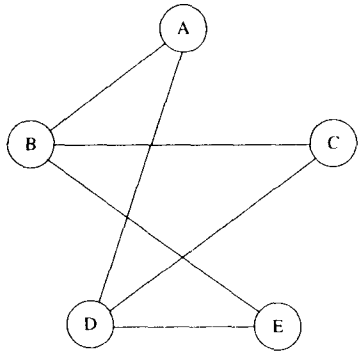


5.4 用分支限界策略解决由下列距离矩阵所刻画的旅行商问题。

i \ j	1	2	3	4	5
1	∞	5	61	34	12
2	57	∞	43	20	7
3	39	42	∞	8	21
4	6	50	42	∞	8
5	41	26	10	35	∞

5.5 求下列子集和问题。$S = \{7, 1, 4, 6, 14, 25, 5, 8\}$，并且$M = 18$。使用分支限界策略，找到一个其元素和为$M$的子集。

5.6 使用某项树搜索技术解决下面图的顶点覆盖问题。



5.7 使用树搜索技术确定下列布尔公式的可满足性。

(a) $-X_1 \vee X_2 \vee X_3$

$X_1 \vee X_3$

X_2

(b) $-X_1 \vee X_2 \vee X_3$

$X_1 \vee X_2$

$-X_2 \vee X_3$

$-X_3$

(c) $X_1 \vee X_2 \vee X_3$

$-X_1 \vee -X_2 \vee -X_3$

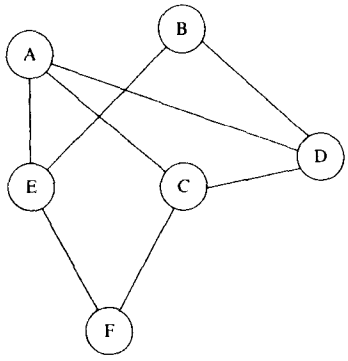
(d) $X_1 \vee X_2$

$-X_2 \vee X_3$

$-X_3$

5.8 考虑在习题5.6中的图，这个图是2可着色吗？用树的某种搜索技术回答这个问题。

5.9 参见下图，通过树搜索证明这个图包含一个3团。



5.10 设计一个试验测试在分支限界策略基础上解决旅行商问题算法的平均情况下的性能。

第 6 章　剪枝搜索方法

在本章中，将讨论一种称为剪枝搜索（prune-and-search）的很好的算法设计策略。该方法可以解决许多问题，尤其是优化问题，且通常能给出有效的算法。例如，当维数固定时，可以使用剪枝搜索策略在线性时间内解决具有n个约束的线性规划问题（linear programming problem）。在接下来的几节中，先介绍剪枝搜索策略的一般步骤，然后讨论一些可用该策略有效解决的问题。

6.1　方法概述

剪枝搜索策略通常由几次迭代组成。在每次迭代中，它剪除输入数据的一部分，比如f比例的数据，然后递归地调用相同的算法处理剩余的数据来解决问题。p次迭代之后，输入数据的规模变为q，此时q很小，可以在常数时间c'内直接解决。此类算法的时间复杂度分析如下：设每次迭代执行剪枝搜索的时间为$O(n^k)$，其中k为常数，且剪枝搜索算法的最坏情况下执行时间为$T(n)$，那么

$$T(n) = T((1-f)n) + O(n^k)$$

可以得到

$$\begin{aligned}
T(n) &\leqslant T((1-f)n) + cn^k \qquad n\text{足够大时} \\
&\leqslant T((1-f)^2 n) + cn^k + c(1-f)^k n^k \\
&\vdots \\
&\leqslant c' + cn^k + c(1-f)^k n^k + c(1-f)^{2k} n^k + \cdots + c(1-f)^{pk} n^k \\
&= c' + cn^k[1 + (1-f)^k + (1-f)^{2k} + \cdots + (1-f)^{pk}]
\end{aligned}$$

由于$1-f<1$，当$n\to\infty$时，

$$T(n) = O(n^k)$$

上面的公式表明整个剪枝搜索过程的时间复杂度与每次迭代的时间复杂度是同级的。

6.2　选择问题

给定n个元素，要求确定其中的第k小元素。解决该问题的一种方法是先将n个元素排序，然后从有序序列中确定第k小元素。因为排序算法的时间复杂度是$O(n \log n)$，所以这也决定了该方法在最坏情况下的时间复杂度是$O(n \log n)$。在本节中，将看到应用剪枝搜索策略可以在线性时间内解决选择问题（select problem）。中位数问题（median problem），即找出第$\lceil n/2 \rceil$小元素，是选择问题的特例。这样，中位数问题也可以在线性时间内完成。

在线性时间内用剪枝搜索解决选择问题算法的基本思想是确定不包含第k小元素的那部分元素，在每次迭代时将该部分剪除。从6.1节中我们知道，要得到$O(n)$时间的算法，必须在每次迭代时用$O(n)$时间剪去部分元素。下面将看到，每次迭代时用于搜索的时间是可以忽略的。

令S表示输入数据集合，p为S中某元素。可以把S分成S_1、S_2和S_3三个子集，其中S_1中包含所有小于p的元素，S_2中包含所有等于p的元素，S_3中包含所有大于p的元素。如果S_1集合中的元素个数大于k，那么可以确定S的第k小元素就在S_1中，并且在接下来的迭代中可以剪去S_2和S_3；

否则，如果S_1和S_2元素个数之和大于k，那么p就是S的第k小元素；如果上面情况均不满足，那么S的第k小元素必大于p。这样，可以丢弃S_1和S_2，在下次迭代时，重新开始从S_3中找出第（$k-|S_1|-|S_2|$）小元素。

关键点是如何选择p，使在每一次迭代时都能丢弃S的一部分，不管该部分是S_1、S_2还是S_3。可以按如下方法选择p：首先，将n个元素分成每5个元素形成一个集合的$\lceil n/5 \rceil$个子集。如果需要，可以在最后的子集中加入虚拟元素∞。然后，对每个5元素子集排序，从每个子集中选出中位数形成新序列$M = \{m_1, m_2, \cdots, m_{\lceil n/5 \rceil}\}$，并令$p$为$M$的中位数。如图6-1所示，$S$中至少有1/4元素小于或等于$p$，至少1/4的元素大于或等于$p$。

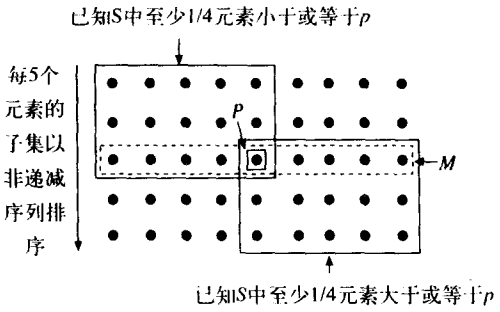


图6-1 选择过程中点的剪除

这样，如果按照这种方法选择p，那么在每次迭代时总能剪去S中至少$n/4$的元素。现在叙述算法如下。

算法6-1 找出第k小元素的剪枝搜索算法

输入：n元素集合S。

输出：S中第k小元素。

步骤1. 如果$|S| \leqslant 5$，那么用任意蛮力方法解决该问题。

步骤2. 将S分成$\lceil n/5 \rceil$个子集，每个子集含有5个元素。如果n不是5的倍数，那么在最后的子集中加入一些虚拟∞元素。

步骤3. 对每个子集排序。

步骤4. 递归地找出$\lceil n/5 \rceil$个子集中位数的中位数p。

步骤5. 将S分成S_1、S_2和S_3，使各自包含的元素分别小于、等于和大于p。

步骤6. 如果$|S_1| \geqslant k$，那么丢弃S_2和S_3，在下次迭代时，解决从S_1中找出第k小元素的问题；否则，如果$|S_1| + |S_2| \geqslant k$，那么$p$为$S$中第$k$小元素；否则，令$k' = k-|S_1|-|S_2|$，在下次迭代时求$S_3$中第$k'$小元素。

令$T(n)$为上面叙述的从S中选择第k小元素算法的时间复杂度。由于每个子集包含常数数目的元素，所以每个子集排序用常数时间。这样，步骤2、3和5可以在$O(n)$时间内完成。如果递归地使用相同的算法，从$\lceil n/5 \rceil$元素中找到中位数，那么步骤4需要$T(\lceil n/5 \rceil)$时间。因为在每次迭代时都剪去至少$n/4$个元素，步骤6中最多包含$3n/4$个元素，因此可以在$T(\lceil 3n/4 \rceil)$时间内完成。所以，

$$T(n) = T(3n/4) + T(n/5) + O(n)$$

令$T(n) = a_0 + a_1 n + a_2 n^2 + \cdots$，$a_1 \neq 0$。

得到

$$T\left(\frac{3}{4}n\right) = a_0 + \frac{3}{4}a_1 n + \frac{9}{16}a_2 n^2 + \cdots$$

$$T\left(\frac{1}{5}n\right) = a_0 + \frac{1}{5}a_1 n + \frac{1}{25}a_2 n^2 + \cdots$$

$$T\left(\frac{3}{4}n + \frac{1}{5}n\right) = T\left(\frac{19}{20}n\right) = a_0 + \frac{19}{20}a_1 n + \frac{361}{400}a_2 n^2 + \cdots$$

这样，

$$T\left(\frac{3}{4}n\right)+T\left(\frac{1}{5}n\right)\leqslant a_0+T\left(\frac{19}{20}n\right)$$

所以，

$$\begin{aligned}T(n)&=T\left(\frac{3}{4}n\right)+T\left(\frac{1}{5}n\right)+O(n)\\&\leqslant T\left(\frac{19}{20}n\right)+cn\end{aligned}$$

将6.1节中得到的公式用于该不等式，得到

$$T(n)=O(n)$$

这样，基于剪枝搜索策略得到在最坏情况下为线性时间的解决选择问题的算法。

6.3 两变量线性规划

很长时间以来，线性规划的计算复杂度一直是引起计算机领域科学家极大兴趣的主题。尽管Khachian提出了一种聪明算法，指出线性规划问题可以在多项式时间内解决，然而，由于其中涉及的常数太大，因此该算法仅具有理论价值。Megiddo和Dyer各自独立地提出了在$O(n)$时间内，解决固定变量数的线性规划问题的剪枝搜索策略，其中的n为约束的数目。

在本节中，我们将描述他们提出的解决两变量线性规划问题的技术。特殊的两变量线性规划问题定义如下：

最小化 $ax+by$

约束为 $a_ix+b_iy\geqslant c_i$，$i=1, 2, \cdots, n$

用剪枝搜索策略解决两变量线性规划问题的基本思想是总有些与解无关的约束，因此可以剪去它们。在剪枝搜索方法中，每次迭代之后会有一部分约束被剪去。数次迭代之后，约束的数目变得很小，使线性规划问题可以在常数时间内解决。

为了简化讨论，描述用于解决简化的两变量线性规划问题的剪枝搜索方法：

最小化 y

约束为 $y\geqslant a_ix+b_i$，$i=1, 2, \cdots, n$

考虑图6-2，总共有8个约束，(x_0, y_0)为最优解。

由于对所有i，$y\geqslant a_ix+b_i$。我们知道最优解必位于围绕可行区域的边界上，如图6-2所示。对于每个x，边界$F(x)$必定在所有n个约束中有最大值。也就是，

$$F(x)=\max_{1\leqslant i\leqslant n}\{a_ix+b_i\}$$

其中a_ix+b_i为输入约束。最优解x_0满足以下方程：

$$F(x_0)=\max_{-\infty\leqslant x\leqslant\infty}F(x)$$

做下面的假定：

(1) 选择一个点x_m，如图6-3所示。

(2) 由于某些原因，可知$x_0\leqslant x_m$。

在这种情况下，考虑

$y=a_1x+b_1$

与 $y=a_2x+b_2$

这两条直线的交点位于x_m的右方。在这两条直线中，当$x<x_m$时，其中一条小于另一条。这

约束可以删去，因为它不会是边界的一部分，如图6-4所示。

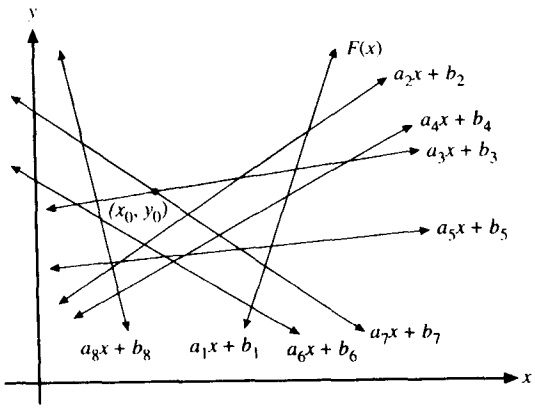


图6-2 特殊两变量线性规划问题的例子

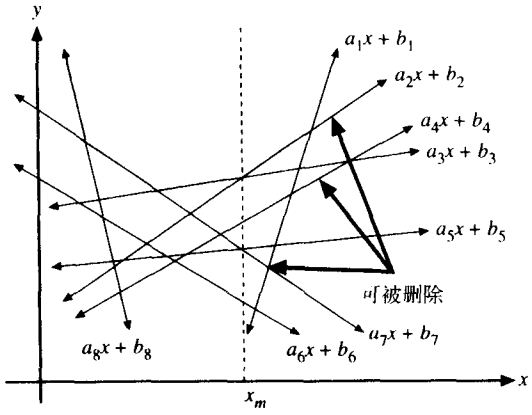


图6-3 两变量线性规划问题中可以消去的约束

这样，如果选择了x_m，且已知$x_0<x_m$，那么可删除$a_1x + b_1$。

类似地，由于$a_3x + b_3$与$a_4x + b_4$的交点在x_m的右方，$x_0<x_m$，且当$x \leqslant x_m$时，$a_4x + b_4<a_3x + b_3$，所以可以将约束$a_4x + b_4$删除。

一般来说，如果有一个x_m，使得最优解$x_0<x_m$，且两个输入约束$a_1x + b_1$和$a_2x + b_2$的交点位于x_m右方，那么当$x<x_m$时，其中的一个约束总是小于另一个。所以，总是可以删除这个约束，因为它不影响最优解的搜索。

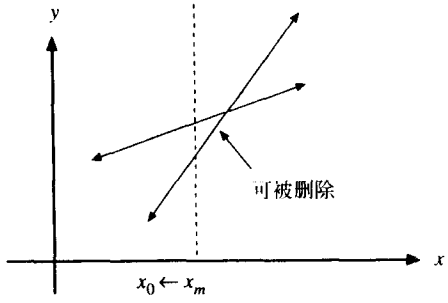


图6-4 为什么约束可以消去的说明

读者一定注意到只能在知道$x_0<x_m$的情况下，才可删除约束$a_1x + b_1$。换言之，假设正在搜索过程中，沿着边界从x_m到x_0方向搜索，才可以消除$a_1x + b_1$，因为当$x<x_m$时，$a_1x + b_1$对结果的搜索没有帮助。必须注意$a_1x + b_1$是可行区域边界的一部分，但这种情况只有当$x>x_m$时才成立。

如果$x_0>x_m$，那么对于交点位于x_m之左的所有约束对，当$x>x_m$时，必有一个约束小于另一个，可以删去此约束而不会影响解。

我们仍需回答如下两个问题：

(1) 假设已知x_m，如何得知搜索的方向？也就是说，如何知道是$x_0<x_m$还是$x_0>x_m$？

(2) 如何选择x_m？

现在回答第一个问题。假设已选定x_m。令

$$y_m = F(x_m) = \max_{1 \leqslant i \leqslant n}\{a_i x_m + b_i\}$$

显然，(x_m, y_m)是可行区域边界上的点，有两种可能：

情况1. y_m只在一个约束上，

情况2. y_m在若干约束的交点上。

对于情况1，简单查看该约束的斜率g。如果$g>0$，那么$x_0<x_m$；如果$g>0$，那么$x_0>x_m$。如图6-5所示。

对于情况2，计算下列各式：

$$g_{\max} = \max_{1 \leqslant i \leqslant n}\{a_i : a_i x_m + b_i = F(x_m)\}$$

$$g_{\min} = \min_{1 \le i \le n}\{a_i : a_i x_m + b_i = F(x_m)\}$$

换言之，在(x_m, y_m)彼此相交的所有约束中，设$g_{\min}$和$g_{\max}$分别为这些约束的最小和最大斜率，那么，有三种可能性：

(1) 情况2a：$g_{\min}>0$且$g_{\max}>0$。此时，$x_0<x_m$。

(2) 情况2b：$g_{\min}<0$且$g_{\max}<0$。此时，$x_0>x_m$。

(3) 情况2c：$g_{\min}<0$且$g_{\max}>0$。此时，(x_m, y_m)为最优解。

三种情况如图6-6所示。

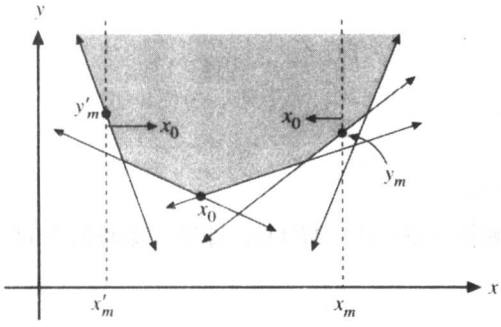


图6-5 x_m只在一个约束上的情况

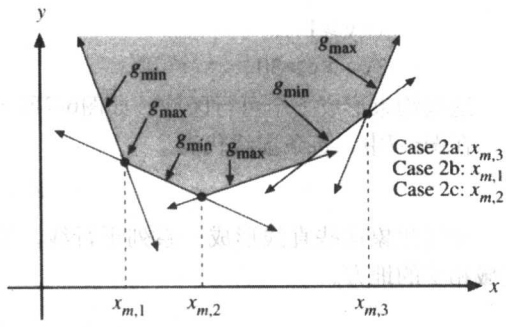


图6-6 x_m在数个约束的交点上的情况

我们已经回答了第一个问题。现在来回答第二个问题：如何选择x_m？应按如下方法选择x_m：按选定的线对，使得两两直线的交点中一半交点位于x_m之左，一半位于x_m之右。这意味着对n个约束，可以把它们任意分为$n/2$组。对每一对，找到它们的交点。在所有$n/2$个交点中，让x_m为其x坐标的中点。

在算法6-2中，给出基于剪枝搜索策略寻找特殊线性规划问题的最优解的详细算法。

算法6-2 解决特殊线性规划问题的剪枝搜索算法

输入：约束：S：$a_i x + b_i$，$i = 1, 2, \cdots, n$。

输出：在约束$y \ge a_i x + b_i$（$i = 1, 2, \cdots, n$）下，使得在x_0值处y最小。

步骤1. 如果S包含不多于两个约束。那么用某种蛮力算法解决该问题。

步骤2. 将S分成$n/2$对约束，对每对约束$a_i x + b_i$与$a_j x + b_j$，求出它们的交点p_{ij}，将其x坐标值记为x_{ij}。

步骤3. 在所有x_{ij}中（最多有$n/2$个），找到中点x_m。

步骤4. 确定$y_m = F(x_m) = \max_{1 \le i \le n}\{a_i x_m + b_i\}$

$$g_{\max} = \max_{1 \le i \le n}\{a_i : a_i x_m + b_i = F(x_m)\}$$

$$g_{\min} = \min_{1 \le i \le n}\{a_i : a_i x_m + b_i = F(x_m)\}$$

步骤5. 情况5a：如果$g_{\min}$与$g_{\max}$符号不同，那么y_m为所求解，退出。

情况5b：否则，如果$g_{\min}>0$，那么$x_0<x_m$；如果$g_{\min}<0$，那么$x_0>x_m$。

步骤6. 情况6a：如果$x_0<x_m$，那么对交点的x坐标大于x_m的点对，对于$x \le x_m$，剪去两约束中的小者。

情况6b：如果$x_0>x_m$，那么对交点的x坐标小于x_m的点对，对于$x \ge x_m$，剪去两约束中的小者。

令S表示剩余的约束，转至步骤2。

对上面算法的复杂性分析如下：由于两直线的交点可在常数时间内找到，因此步骤2可在$O(n)$时间内完成。由6.2节可知中点可在$O(n)$时间内找到。步骤4可通过线性扫描所有约束完成。这样，步骤4和5可在$O(n)$时间内完成。步骤6也可通过扫描所有相交直线对在$O(n)$时间内完成。

由于使用选择的交点的中点，它们中的一半位于x_m的右方，总共有$\lfloor n/2 \rfloor$个交点。对每个交点剪去一个约束。这样，在每次迭代后剪去$\lfloor n/4 \rfloor$个约束。基于6.1节的结果，上面算法的时

间复杂度为$O(n)$。

现在回到最初的两变量线性规划问题。该问题定义如下：

最小化 $Z = ax + by$

约束为 $a_ix + b_iy \geqslant c_i$，$i = 1, 2, \cdots, n$

现在举一个典型例子：

最小化 $Z = 2x + 3y$

约束为 $x \leqslant 10$

$y \leqslant 6$

$x - y \geqslant 1$

$3x + 8y \geqslant 30$

这些约束形成一个可行区域，如图6-7所示。

在图6-7中，每条虚线代表

$$Z = 2x + 3y$$

可以想象这些直线形成一系列平行线。最优解位于(38/11，27/11)，即第一条虚线与可行区域相交的地方。

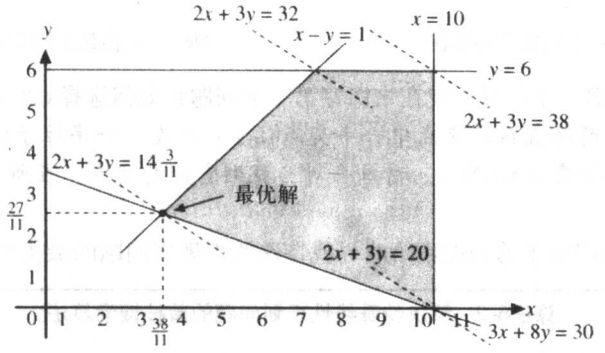


图6-7 一般的两变量线性规划问题

可以将上面一般的两变量线性规划问题转变成一个仅最小化y值的问题。我们的原始问题为：

最小化 $Z = ax + by$

约束为 $a_ix + b_iy \geqslant c_i$，$i = 1, 2, \cdots, n$

现令 $x' = x$

$y' = ax + by$

那么原问题变为

最小化 y'

约束为 $a_i'x' + b_i'y' \geqslant c_i'$ $i = 1, 2, \cdots, n$

其中 $a_i' = a_i - b_ia/b$

$b_i' = b_i/b$

$c_i' = c_i$

因此，可以说一般的两变量问题可以经过n步后转化成下面的问题：

最小化 y

约束为 $a_ix + b_iy \geqslant c_i$，$i = 1, 2, \cdots, n$

读者应当注意到上面的问题与特殊两变量线性规划问题有一点不同。在特殊两变量线性规

划问题中，定义问题为

最小化 y

约束为 $y \geqslant a_i x + b_i$，$i = 1, 2, \cdots, n$

但是，在一般情况下有三种约束：

$$y \geqslant a_i x + b_i$$
$$y \leqslant a_i x + b_i$$

和 $a \leqslant x \leqslant b$

考虑图6-7中的四种约束，它们是

$$x \leqslant 10$$
$$y \leqslant 6$$
$$x - y \geqslant 1$$
$$3x + 8y \geqslant 30$$

可以将上面的公式改写成

$$x \leqslant 10$$
$$y \leqslant 6$$
$$y \leqslant x - 1$$
$$y \geqslant \frac{-3}{8}x + \frac{30}{8}$$

可以将变量y的正（负）系数的分组约束在集合$I_1(I_2)$中，那么原始问题变为

最小化 y

约束为 $y \geqslant a_i x + b_i \ (i \in I_1)$

$y \leqslant a_i x + b_i \ (i \in I_2)$

$a \leqslant x \leqslant b$

定义两个函数：

$F_1(x) = \max\{ a_i x + b_i : i \in I_1\}$

$F_2(x) = \min\{ a_i x + b_i : i \in I_2\}$。

$F_1(x)$是一个分段线性凸函数，$F_2(x)$是一个分段线性凹函数。两个函数及约束$a \leqslant x \leqslant b$确定了线性规划问题的可行区域，如图6-8所示。

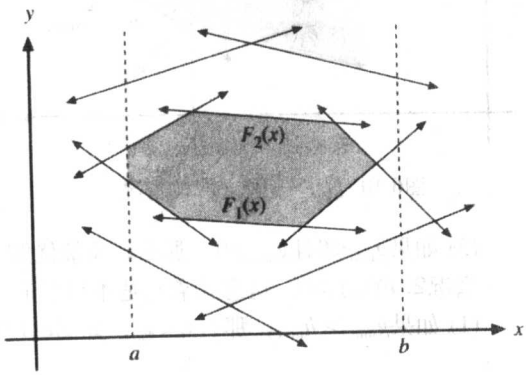


图6-8 两变量线性规划问题的可行区域

这样，原始2变量线性规划问题可以进一步转化成

最小化 $F_1(x)$

约束为 $F_1(x) \leqslant F_2(x)$

$a \leqslant x \leqslant b$

在此，再次指出如果知道搜索方向，那么一些约束可以被剪掉。其原因与特殊2变量线性规划问题中讨论的一样。考虑图6-9，如果知道最优解x_0位于x_m的左方，那么可以删除$a_1x + b_1$而不影响解，因为当$x<x_m$时，$a_1x + b_1 < a_2x + b_2$。我们知道这种情况是因为$a_1x + b_1$与$a_2x + b_2$的交点位于x_m的右方。同样，可以消去$a_4x + b_4$，因为当$x<x_m$时，$a_4x + b_4 > a_5x + b_5$。

读者现在注意到，解决2变量线性规划问题的剪枝搜索算法中最重要的问题是确定x_0（最优解）位于x_m的左方或右方。x_m的解能以类似在特殊2变量线性规划问题中的方法解决。

通常，已知一点x_m，$a \leqslant x_m \leqslant b$，需要确定下面的问题：

(1) x_m可行吗？

(2) 如果x_m可行，那么必须确定最优解x_0位于其左，还是其右。也可能出现x_m本身即最优解的情况。

(3) 如果x_m不可行，那么必须确定是否存在可行解。如果可行解存在，那么必须确定该最优解位于x_m的哪一方。

接下来，将描述该决策过程。考虑$F(x) = F_1(x) - F_2(x)$，显然，当且仅当$F(x_m) \leqslant 0$时，x_m可行。为了确定最优解位于哪一边，我们定义如下：

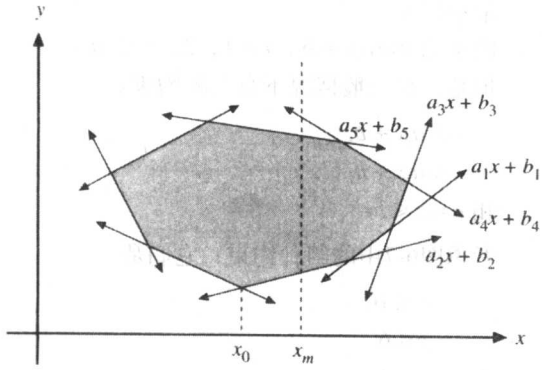


图6-9 一般两变量线性规划问题中约束的剪除

$$g_{\min} = \min\{a_i : i \in I_1,\ a_i x_m + b_i = F_1(x_m)\}$$
$$g_{\max} = \max\{a_i : i \in I_1,\ a_i x_m + b_i = F_1(x_m)\}$$
$$h_{\min} = \min\{a_i : i \in I_2,\ a_i x_m + b_i = F_2(x_m)\}$$
$$h_{\max} = \max\{a_i : i \in I_2,\ a_i x_m + b_i = F_2(x_m)\}$$

考虑下面的情况：

情况1. $F(x_m) \leqslant 0$。这意味着x_m是可行的。

(1) 如果$g_{\min} > 0$且$g_{\max} > 0$，那么$x_0 < x_m$，如图6-10所示。

(2) 如果$g_{\max} < 0$且$g_{\min} < 0$，那么$x_0 > x_m$，如图6-11所示。

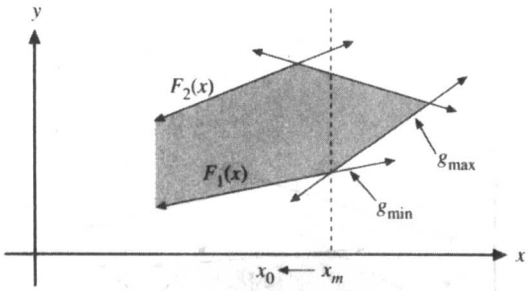


图6-10 $g_{\min} > 0$且$g_{\max} > 0$的情况

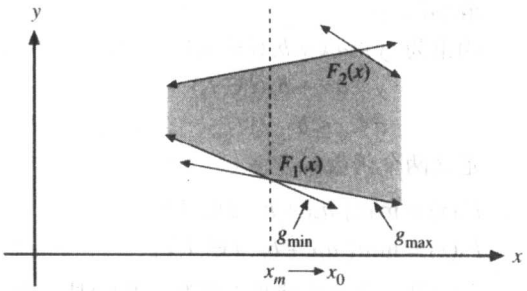


图6-11 $g_{\min} < 0$且$g_{\max} < 0$的情况

(3) 如果$g_{\min} < 0$且$g_{\max} > 0$，那么x_m为最优解，如图6-12所示。

情况2. $F(x_m) > 0$。这意味着x_m是不可行的。

(1) 如果$g_{\min} > h_{\max}$，那么$x_0 < x_m$，如图6-13所示。

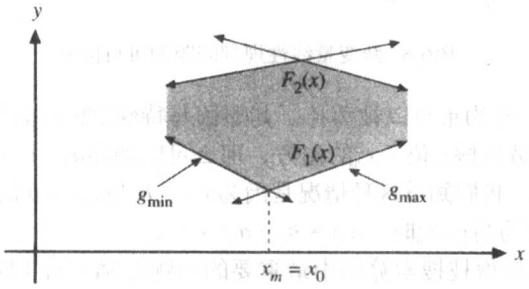


图6-12 $g_{\min} < 0$且$g_{\max} > 0$的情况

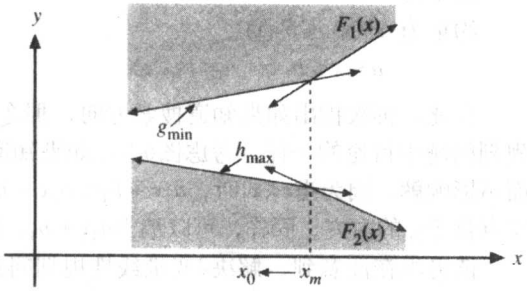


图6-13 $g_{\min} > h_{\max}$的情况

(2) 如果$g_{\max} < h_{\min}$，那么$x_0 > x_m$，如图6-14所示。

(3) 如果$g_{\min} \leqslant h_{\max}$且$g_{\max} \geqslant h_{\min}$，那么不存在可行解，因为$F(x)$在$x_m$处取得最小值，如图6-15所示。

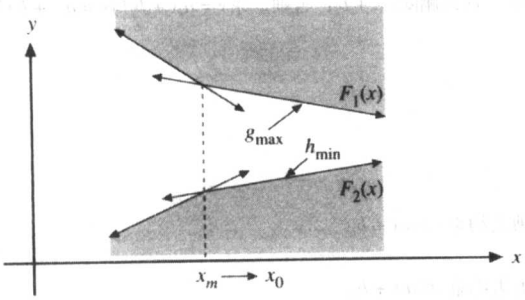


图6-14 $g_{\max} < h_{\min}$的情况

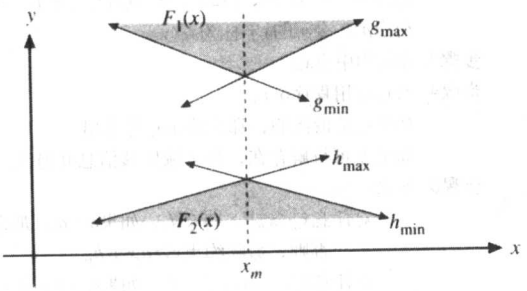


图6-15 $g_{\min} \leqslant h_{\max}$且$g_{\max} \geqslant h_{\min}$的情况

上面的决策过程总结为如下的程序。

程序6-1 算法6-3中使用的程序

输入：某点的x坐标值x_m。

I_1：$y \geqslant a_i x + b_i$，$i = 1, 2, \cdots, n_1'$

I_2：$y \leqslant a_i x + b_i$，$i = n_1' + 1, n_1' + 2, \cdots, n'$

$a \leqslant x \leqslant b$

输出：从x_m继续搜索是否有意义。如果有意义，那么输出搜索的方向。

步骤1. $F_1(x) = \max\{a_i x + b_i : i \in I_1\}$

$F_2(x) = \mathrm{mix}\{a_i x + b_i : i \in I_2\}$

$F(x) = F_1(x) - F_2(x)$

步骤2. $g_{\min} = \min\{a_i : i \in I_1,\ a_i x_m + b_i = F_1(x_m)\}$

$g_{\max} = \max\{a_i : i \in I_1,\ a_i x_m + b_i = F_1(x_m)\}$

$h_{\min} = \min\{a_i : i \in I_2,\ a_i x_m + b_i = F_2(x_m)\}$

$h_{\max} = \max\{a_i : i \in I_2,\ a_i x_m + b_i = F_2(x_m)\}$

步骤3. 情况1：$F(x_m) \leqslant 0$。

(a) 如果$g_{\min} > 0$且$g_{\max} > 0$，那么输出“$x_0 < x_m$”，并退出。

(b) 如果$g_{\min} < 0$且$g_{\max} < 0$，那么输出“$x_0 > x_m$”，并退出。

(c) 如果$g_{\min} < 0$且$g_{\max} > 0$，那么输出“x_m是最优解”，并退出。

情况2：$F(x_m) > 0$。

(a) 如果$g_{\min} > h_{\max}$，那么输出“$x_0 < x_m$”，并退出。

(b) 如果$g_{\max} < h_{\min}$，那么输出“$x_0 > x_m$”，并退出。

(c) 如果$g_{\min} \leqslant h_{\max}$且$g_{\max} \geqslant h_{\min}$，那么输出“不存在可行解”，并退出。

解决2变量线性规划问题的剪枝搜索算法如下。

算法6-3 解决2变量线性规划问题的剪枝搜索算法

输入：I_1：$y \geqslant a_i x + b_i$，$i = 1, 2, \cdots, n_1$

I_2：$y \leqslant a_i x + b_i$，$i = n_1 + 1, n_1 + 2, \cdots, n$

$a \leqslant x \leqslant b$

输出：在以下约束下，y取得最小值时的x_0值

$y \geqslant a_i x + b_i$，$i = 1, 2, \cdots, n_1$

$y \leqslant a_i x + b_i$，$i = n_1 + 1, n_1 + 2, \cdots, n$

$a \leqslant x \leqslant b$

（续）

步骤1. 如果I_1与I_2中的约束不多于两个，那么用蛮力方法解决该问题。

步骤2. 将I_1中的约束分成不相交的对，同样，将I_2中的约束也分成不相交的对。对每一对，如果$a_ix + b_i$平行于$a_jx + b_j$，若对于i，$j \in I_1$，$b_i<b_j$或i，$j \in I_2$，$b_i>b_j$，那么删除$a_ix + b_i$；否则，求$y = a_ix + b_i$与$y = a_jx + b_j$的交点p_{ij}。令p_{ij}的x坐标为x_{ij}。

步骤3. 求x_{ij}的中点x_m。

步骤4. 对x_m应用程序6-1。

如果x_m是最优的，那么输出x_m并退出。

如果无可行解存在，那么输出该信息并退出。

步骤5. 如果$x_0>x_m$

对任意$x_{ij}<x_m$且i，$j \in I_1$，如果$a_i<a_j$，那么剪去约束$y \geqslant a_ix + b_i$；

否则，剪除约束$y \geqslant a_jx + b_j$。

对任意$x_{ij}<x_m$且i，$j \in I_2$，如果$a_i>a_j$，那么剪去约束$y \leqslant a_ix + b_i$；

否则，剪除约束$y \leqslant a_jx + b_j$。

如果$x_0<x_m$

对任意$x_{ij}>x_m$且i，$j \in I_1$，如果$a_i>a_j$，那么剪去约束$y \geqslant a_ix + b_i$；

否则，剪除约束$y \geqslant a_jx + b_j$。

对任意$x_{ij}>x_m$且i，$j \in I_2$，如果$a_i<a_j$，那么剪去约束$y \leqslant a_ix + b_i$；

否则，剪除约束$y \leqslant a_jx + b_j$。

步骤6. 转到步骤1。

因为每次迭代都可以剪去$\lfloor n/4 \rfloor$个约束，而在算法6-3中的每一步需花费$O(n)$的时间，所以很容易得出此算法是$O(n)$阶的。

6.4 1圆心问题

1圆心问题（1-center problem）定义如下：给定n个平面点的集合，要找到一个覆盖所有几个点的最小圆。图6-16给出了一个典型示例。

在本节中，将给出一个基于剪枝搜索策略解决1圆心问题的方法。该问的输入数据为n个点。在每一步里，剪除不影响最终解的1/16的点，基于剪枝搜索解决1圆心问题的方法只需线性时间，因为在每一步中它所需要的时间都是线性的。

图6-17给出剪枝搜索方法的例子。L_{12}和L_{34}分别是连接p_1与p_2，以及p_3与p_4线段的垂直平分线。假设还知道最优解的圆心位于阴影区域，现在考虑L_{12}。因为L_{12}没有与阴影区域相交，两点中必有其一离最优解圆心近。在我们的例子中，p_1比p_2接近于圆心。这样，p_1可以删除掉，因为它不影响结果。

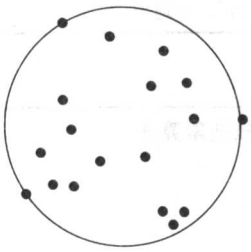


图6-16 1圆心问题

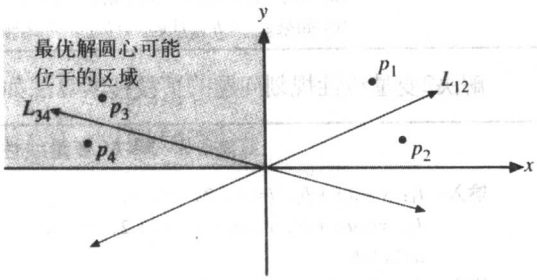


图6-17 1圆心问题中剪除点的可能情况

重要的是，需要知道最优解圆心位于哪一部分，还需知道哪些点比其他点离圆心近，这些较近的点可以删除。

在给出1圆心问题通用算法前，先介绍一种有约束的1圆心问题的算法，约束圆心在一条直线上。这一方法将作为一个子程序被通用1圆心方法调用。不失一般性，假定直线为$y = y'$。

算法6-4 解决有约束1圆心问题的算法

输入：n个点和一条直线$y = y'$。

输出：在直线$y = y'$上有约束的1圆心。

步骤1. 如果n不大于2，那么用蛮力方法解决该问题。

步骤2. 构造不相交点对(p_1, p_2)，(p_3, p_4)，…，(p_{n-1}, p_n)。如果有奇数个点，那么只需要令最后点对为(p_n, p_1)。

步骤3. 对每个点对(p_i, p_{i+1})，求出在直线$y = y'$上的点$x_{i,i+1}$，使得$d(p_i, x_{i,i+1}) = d(p_{i+1}, x_{i,i+1})$。

步骤4. 求$\lfloor n/2 \rfloor$个$x_{i,i+1}$的中点，记为x_m。

步骤5. 对所有的i，计算p_i与x_m的距离。令p_j为离x_m最远的点，x_j表示p_j在$y = y'$上的投影。如果x_j在x_m的左（右）方，那么最优解x^*必位于x_m的左（右）方。

步骤6. 如果$x^* < x_m$（如图6-18所示）

对每个$x_{i,i+1} > x_m$，如果p_i比p_{i+1}距x_m近，那么剪去点p_i；否则，剪去点p_{i+1}。

如果$x^* > x_m$

对每个$x_{i,i+1} < x_m$，如果p_i比p_{i+1}距x_m近，那么剪去p_i点；否则，剪去点p_{i+1}。

步骤7. 转向步骤1。

由于在x_m左（右）边有$\lfloor n/4 \rfloor$个$x_{i,i+1}$点，对于每一次迭代或整个算法而言，可以剪去$\lfloor n/4 \rfloor$个点，每次迭代花费$O(n)$时间。这样，该算法的时间复杂度为

$$T(n) = T(3n/4) + O(n) = O(n)，随着n \to \infty$$

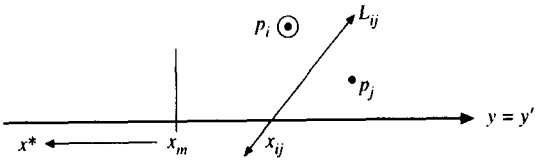


图6-18 有约束的1圆心问题中点的剪除

在此需强调，有约束1圆心问题算法将用于主算法中以求得最优方案。有约束1圆心问题算法中剪除的点仍被主算法使用。

现在考虑一个较为复杂的问题。设想有一个点集和一条直线$y = 0$，如图6-19所示。使用有约束1圆心算法可以确定x^*在直线上的确切位置。实际上，利用该信息，可以做更多的事情。令(x_s, y_s)为包括所有点的最优解圆心，可以确定是否$y_s > 0$、$y_s < 0$或$y_s = 0$。出于同样原因，也可以确定是否$x_s > 0$、$x_s < 0$或$x_s = 0$。

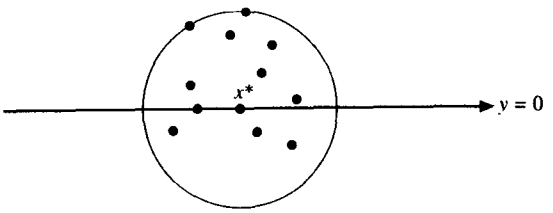


图6-19 对1圆心问题中有约束1圆心问题的解决

令I表示距离点$(x^*, 0)$最远点组成的集合，有两种可能的情况。

情况1. I包括一个点，记为p。

这种情况，p的x坐标必定等于x^*。若非如此，那么可以将x^*沿直线$y = 0$向p移动，这与$(x^*, 0)$为最优解的假设相矛盾。这样，如果p为距$(x^*, 0)$最远的唯一点，那么它的x值必定等于x^*，如图6-20所示。可以推断y_s与p的y坐标有相同的正负号。

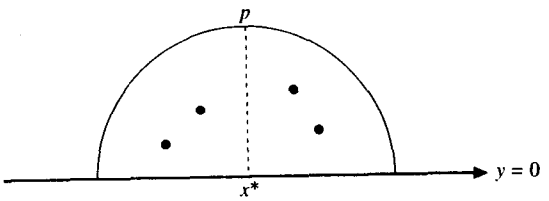


图6-20 I只包含一点的情况

情况2. I包含多于一点。

在集合I所有的点中，找出覆盖I中所有点的最小弧。令弧的两个端点为p_1和p_2。如果该弧的度数大于或等于180°，那么$y_s = 0$；否则，令y_i为p_i的y坐标，$y_c = (y_1 + y_2)/2$。这

样，y_s的正负号与y_c的相同。这些情况分别如图6-21a、b所示。下面对其进行讨论。

现在考虑第一种情况，也就是覆盖所有最远点的最小弧，其度数大于等于180°的情况。注意包含某点集的最小圆，该点集由这些点中的两个或三个确定，如图6-22a、b所示。三个点确定了包括它们三个点在内的最小圆的边界，当且仅当它们不形成钝角三角形（如图6-22b所示）；否则，可以用三边中的最长边为直径的圆来代替该圆（如图6-22c所示）。在此情况下，因为覆盖所有最远点的弧，其度数大于等于180°，那么最少有三个这样的最远点，而且这三个最远点不形成钝角三角形。换句话说，当前的最小圆已是最优的。进而可以推断$y_s = 0$。

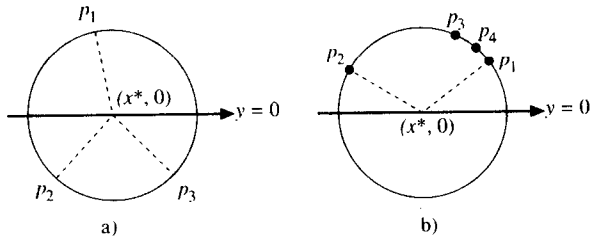


图6-21 I包含多于一点的情况

对于覆盖所有最远点的弧，其度数小于180°的第二种情况，首先看到端点p_1和p_2的x坐标正负号必定相反。若非如此，如图6-23所示，可以将x^*朝p_1和p_2位置的方向移动。这是不可能的，因为x^*是在$y = 0$的最优圆心上。

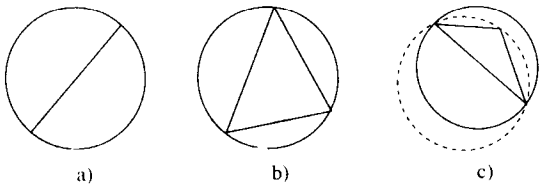


图6-22 两点或三点确定覆盖所有点的最小圆

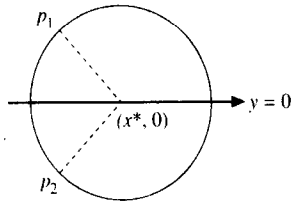


图6-23 度数小于180°时x^*的方向

令$p_1 = (a, b)$，$p_2 = (c, d)$，不失一般性，可以假定$a>x^*$，$b>0$和$c<x^*$，$d<0$，如图6-24所示。在图6-24中有三个圆，圆心分别位于$(x^*, 0)$，(a, b)和(c, d)。这三个圆形成四个区域：R_1、R_2、R_3和R_4。令当前圆的半径为r，有下面三个结论：

(1) 在R_2中，在该区域中任意点x与p_1或p_2之间的距离大于r。因此，最优圆的圆心位置不在R_2区域内。

(2) 在R_1中，在该区域中任意点x与p_2的距离大于r。因此，最优圆的圆心位置不在R_1区域内。

(3) 同理，很容易看出最优圆的圆心也不在R_4区域内。

基于上面三个结论，可以推断最优圆心必落在区域R_3内，进而，y_s必定有与$(b + d)/2 = (y_1 + y_2)/2$相同的正负号。

上面的程序可以总结如下。

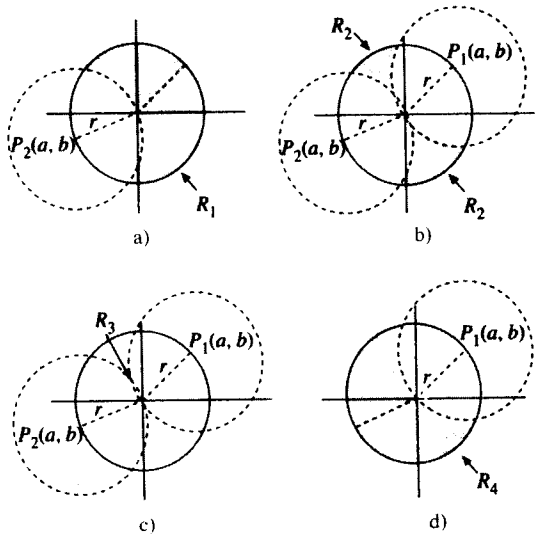


图6-24 度数大于180°时x^*的方向

程序6-2 算法6-5中使用的程序

输入：点集合S。

直线$y = y^*$。

(x^*, y^*)：对于S，在$y = y^*$上有约束1圆心问题的解。

输出：是否$y_s > y^*$，$y_s < y^*$或$y_s = y^*$，其中(xs, ys)是S的1圆心问题的最优解。

步骤1. 找到距离(x^*, y^*)最远点的集合I。

步骤2. **情况1**：I只包含一点$p = (x_p, y_p)$。

如果$y_p > y^*$，那么输出“$y_s > y^*$”，并退出。

如果$y_p < y^*$，那么输出“$y_s < y^*$”，并退出。

情况2：I包含多于一个的点。在I中找到形成覆盖所有I中点的最小弧的两个端点$p_1 = (x_1, y_1)$与$p_2 = (x_2, y_2)$。

情况2.1：p_1和p_2形成的弧的度数大于等于180°。

输出“$y_s = y^*$”，并退出。

情况2.2：p_1和p_2形成的弧的度数小于180°。令$y_c = (y_1 + y_2)/2$。

如果$y_c > y^*$，那么输出“$y_s > y^*$”，并退出。

如果$y_c < y^*$，那么输出“$y_s < y^*$”，并退出。

有约束的1圆心算法可以用来剪除点。在给出精确的算法前，考虑图6-25。对于点对(p_1, p_2)和(p_3, p_4)，分别画出线段$\overline{p_1p_2}$和$\overline{p_3p_4}$的垂直平分线L_{12}和L_{34}。令L_{12}和L_{34}相交于一点p。旋转x轴以使L_{34}有负斜率，L_{12}有正斜率。将坐标系的原点移至p，先应用有约束的1圆心算法求出在$y = 0$的圆心。找到该有约束的1圆心后，使用程序6-2找出最优解，确定最优解的y方向，在例子中应该向上。重复该过程将有约束的1圆心算法应用于$x = 0$，然后找到x方向，在例子中应该向左移动。因此，最优位置必定在阴影区域内，如图6-25所示。因为有一条垂直平分线与阴影区域不相交，总是可以利用该信息移除所考虑的一个点。

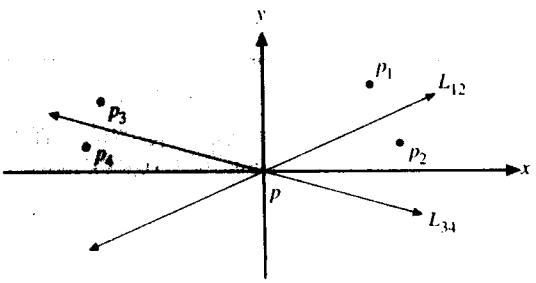


图6-25 坐标系的适当旋转

在例子里，L_{12}不与阴影区域相交，点p_1位于阴影区域一方，这意味着只要最优圆心限定于阴影区域内，点p_1离最优圆心的距离总比p_2小。因此，可以删去点p_1，因为它被点p_2所“支配”。也就是说，只需考虑p_2即可。

下面是用剪枝搜索解决1圆心问题的算法。

算法6-5 解决1圆心问题的剪枝搜索算法

输入：n个点的集合$S = \{p_1, p_2, \cdots, p_n\}$。

输出：包括S中所有点的最小封闭圆。

步骤1. 如果S包含不多于16点，那么用蛮力方法解决该问题。

步骤2. 形成不相交的点对，(p_1, p_2)，(p_3, p_4)，$\cdots$，(p_{n-1}, p_n)。对每个点对(p_i, p_{i+1})，找出线段$\overline{p_ip_{i+1}}$的垂直平分线，记为$L_{i/2}$，$i = 2, 4, \cdots, n$，并计算它们的斜率，将L_k的斜率表示为s_k，$k = 1, 2, \cdots, n/2$。

步骤3. 计算s_k的中点，表示为s_m。

步骤4. 旋转坐标系使x坐标与$y = s_m x$重合。令L_k中斜率为正(负)组成的集合为$I^+(I^-)$。(两者数目均为$n/4$)。

步骤5. 构建不相交的直线对(L_{i+}, L_{i-})，$i = 1, 2, \cdots, n/4$，其中$L_{i+} \in I^+$，$L_{i-} \in I^-$。找到每对直线的交点，表示为(a_i, b_i)，$i = 1, 2, \cdots, n/4$。

步骤6. 找出b_i的中点，表示为y^*。对S应用有约束的1圆心子程序求得在$y = y^*$上的圆心。令有约束的1圆心问题的解为(x^*, y^*)。

（续）

步骤7. 以S和(x', y^*)作为参数，应用程序6-2。

如果$y_s = y^*$，那么输出“步骤6中找到的以(x', y^*)作圆心的圆为最优解”，并退出；

否则，输出$y_s > y^*$或$y_s < y^*$。

步骤8. 求a_i的中点，表示为x^*。对S应用有约束的1圆心子程序求得在$x = x^*$上的圆心。令有约束的1圆心问题的解为(x^*, y')。

步骤9. 以S和(x', y^*)作为参数，应用程序6-2。

如果$x_s = x^*$，那么输出“步骤8中找到的以(x^*, y')作圆心的圆为最优解”，并退出；

否则，输出$x_s > x^*$或$x_s < x^*$。

步骤10. 情况1： $x_s > x^*$且$y_s > y^*$。

找到所有$a_i < x^*$且$b_i < y^*$的(a_i, b_i)。令(a_i, b_i)为L_{i^+}和L_{i^-}的交点（见步骤5），L_{i-}为p_j和p_k的平分线。如果$p_j(p_k)$比$p_k(p_j)$离(x^*, y^*)近，那么剪去$p_j(p_k)$。

情况2： $x_s < x^*$且$y_s > y^*$。

找到所有$a_i > x^*$且$b_i < y^*$的(a_i, b_i)。令(a_i, b_i)为L_{i^+}和L_{i^-}的交点，L_{i^+}为p_j和p_k的平分线。如果$p_j(p_k)$比$p_k(p_j)$离(x^*, y^*)近，那么剪去$p_j(p_k)$。

情况3： $x_s < x^*$且$y_s < y^*$。

找到所有$a_i > x^*$且$b_i > y^*$的(a_i, b_i)。令(a_i, b_i)为L_{i^+}，L_{i^-}的交点，L_{i-}为p_j和p_k的平分线。如果$p_j(p_k)$比$p_k(p_j)$离(x^*, y^*)近，那么剪去$p_j(p_k)$。

情况4： $x_s > x^*$且$y_s < y^*$。

找到所有$a_i > x^*$且$b_i < y^*$的(a_i, b_i)。令(a_i, b_i)为L_{i^+}，L_{i^-}的交点，L_{i^+}为p_j和p_k的平分线。如果$p_j(p_k)$比$p_k(p_j)$离(x^*, y^*)近，那么剪去$p_j(p_k)$。

步骤11. 设S为剩余点。转向步骤1。

对算法6-5分析如下。首先假定对于某个k'，有$n = 16^k$个点，k为常数，在步骤2中形成了$n/2$条垂直平分线。在步骤4之后，有$n/4$的垂直平分线为负斜率，$n/4$的为正斜率。这样，在步骤5中总共有$n/4$个交点。因为$x^*(y^*)$是a_i (b_i)的中点，对步骤10的每种情况，有$(n/4)/4 = n/16$个(a_i, b_i)，如图6-26所示，假定最优圆心位于阴影区域内。然后，对$x > x_m$和$y > y_m$区域中的每个交点对，位于负斜率直线下方的点可以剪去。对每个这样的(a_i, b_i)，都剪除一个点。因此，在每个交点中，有$n/16$个点被剪除。

容易看出算法6-5的每一步花费$O(n)$时间。这样，算法6-5总的时间复杂度为$O(n)$。

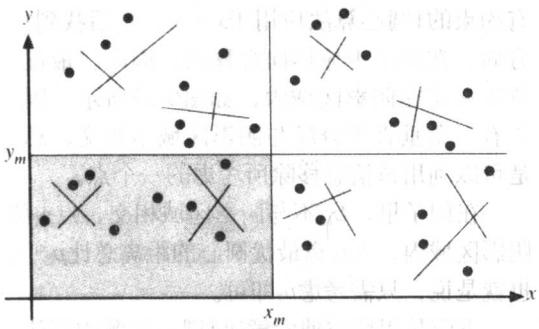


图6-26 不相交点对及其斜率

6.5 实验结果

为了说明剪枝搜索方法的能力，我们实现了对1圆心问题的剪枝搜索算法。该问题还有另一种解决方法。注意到1圆心问题的解必须经过至少三个点。因此，可以尝试每三个点的组合。也就是选择三个点，构造一个圆，看它是否覆盖了所有的点。如果不能，那么丢弃它。如果可以，那么记下它的半径。当检查了所有这些圆后，可以找到最小圆。这样，这是一种$O(n^3)$的方法，而剪枝搜索方法则是一种$O(n)$的方法。

我们在IBM PC上实现了上述两种方法。直接方法用穷举搜索标明。所有数据都是随机产生的。实验结果如表6-1所示。

显然在该情况下，剪枝搜索非常有效。当n为150时，穷举搜索已经比剪枝搜索差多了。当n为300时，穷举搜索用的时间太长以致无法完成。而用剪枝搜索则只需257秒就可解决1圆心问题。有如此结果并不奇怪，因为一种方法的复杂度为$O(n^3)$，而另一种方法的复杂度为$O(n)$。

表6-1 实验结果

样本点数	穷举搜索s	剪枝搜索s
17	1	1
30	8	11
50	28	32
100	296	83
150	1080	130
200	2408	185
250	2713	232
300	—	257

6.6 注释与参考

解决选择问题的方法传统上归入分治策略，我们认为归入剪枝搜索更为恰当。更详尽的讨论参阅文献Blum，Floyd，Pratt，Rivest及Tarjan (1972)和Floyd与Rivest (1975)。选择问题下界的讨论参阅文献Hyafil (1976)。许多教科书也提供了解决选择问题的聪明方法（Brassard和Bratley，1988；Horowitz和Sahni，1978；Kronsjo，1987）。

6.3节中的聪明方法是由Dyer (1984)与Megiddo (1983)分别独立发现的。Megiddo (1984)将结论推广到更高维数。Megiddo (1983)同样提到1圆心问题可以采用剪枝搜索方法实现，这推翻了Shamos和Hoey (1975)的推测。

6.7 进一步的阅读资料

剪枝搜索方法相对较新，与其他领域相比，在这一领域中的文献较少。我们建议把以下材料作为进一步研究的资料：Avis，Bose，Shermer，Snoeyink，Toussaint and Zhu (1996)；Bhattacharya，Jadhav，Mukhopadhyay and Rober (1994)；Chou and Chung (1992)；Imai (1993)；Imai，Kato and Yamamoto (1989)；Imai，Lee and Yang (1992)；Jadhav and Mukhopadhyay (1993)；Megiddo (1983)；Megiddo (1984)；Megiddo (1985)；and Shreesh，Asish and Binay (1996)。

最新出版的有意义的论文可参阅：Eiter and Veith (2002)；ElGindy，Everett and Toussaint (1993)；Kirpatrick and Snoeyink (1993)；Kirpatrick and Snoeyink (1995)；and Santis and Persiano (1994)。

习题

6.1 确信自己弄清楚了分治法与剪枝搜索法的相同点与不同点。注意递归公式看起来相当相似。

6.2 写一个程序实现本章中介绍的选择算法。找出第k大元素的另一种方法是先应用快速排序，然后从中挑出第k大元素。同样实现该方法并比较两段程序，给出测试结果的解释。

6.3 R^d上的两个点集合A和B称为线性可分（linearly separable），如果存在一个$(d-1)$维超平面使A和B分别位于其两侧。试证明线性可分问题是一种线性规划问题。

6.4 根据习题6.3中的结论，证明两维和三维的线性可分问题可在$O(n)$时间内完成。

6.5 阅读文献Horowitz与Sahni(1978)中的定理3.3。它是基于剪枝搜索方法的对选择算法的平均情况的分析。

第7章 动态规划方法

动态规划策略（dynamic programming strategy）是一种用于解决许多组合优化问题（combinatorial optimization problem）的有效方法。在介绍动态规划方法之前，重新考虑使用贪心法解决的典型实例也许是合适的。在此将给出贪心法失效但动态规划方法却起作用的实例。

图7-1包含一个多阶段图（multi-stage graph）。假设要找到从S到T的最短路径。在这种情况下，可以通过下列推导顺序找到最短路径：

(1) 由于知道最短路径一定经过顶点A，所以要找到从S到A的最短路径，最短路径的代价是1。

(2) 由于知道最短路径一定经过顶点B，所以要找到从A到B的最短路径，最短路径的代价是2。

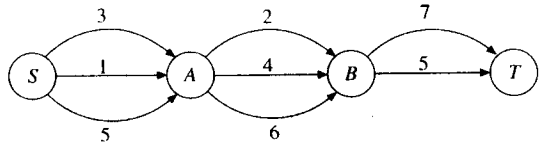


图7-1 贪心法有效工作的实例

(3) 类似地，找到从B到T的最短路径，其代价为5。

因此，从S到T的最短路径的总代价是$1 + 2 + 5 = 8$。总之，使用贪心法的确解决了这个最短路径问题。

为什么可以使用贪心法解决该问题呢？能这样做是因为我们明确地知道问题的解必定由从S到A的子路径，从A到B的子路径等构成。所以，解决问题的策略是先找到从S到A的最短路径，再找到从A到B的最短路径等。

现在给出一种贪心法无效的实例，见图7-2。同样，要找到从S到T的最短路径。然而，这次并不知道最短路径要经过顶点A、B和C中的哪一个。如果还使用贪心法解决该问题，选择顶点A，因为从S到A的边的代价最小。在选择A之后，接着选择D。此条路径的代价是$1 + 4 + 18 = 23$。这不是最短路径，最短路径应为$S \to C \to F \to T$，其总代价为$5 + 2 + 2 = 9$。

如上所示，应选C而不是A。动态规划方法会通过下面的推导方式找出此问题的解。

(1) 我们知道，必须从S开始，经过A、B或C，这如图7-3所示。因此，从S到T的最短路径的长度取决于下面的公式：

$$d(S,\ T) = \min\{1 + d(A,\ T),\ 2 + d(B,\ T),\ 5 + d(C,\ T)\} \tag{7-1}$$

其中，$d(X,\ T)$表示从X到T的最短路径的长度。当分别找到从A、B和C到T的最短路径后，从S到T的最短路径也就找到了。

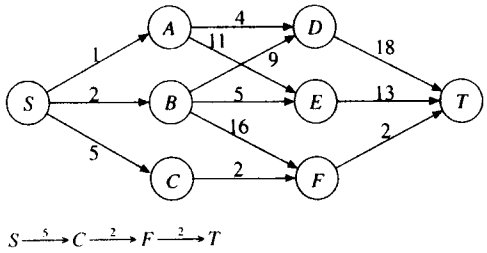


图7-2 贪心法无效的实例

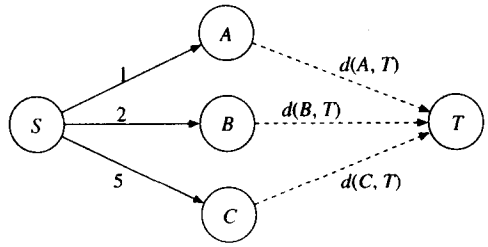


图7-3 使用动态规划方法过程中的一步

(2) 为了找到从A、B和C到T的最短路径，可以再次使用上面的方法。顶点A所得到的子图如图7-4所示。也就是说，$d(A, T) = \min\{4 + d(D, T), 11 + d(E, T)\}$。由于$d(D, T) = 18$，$d(E, T) = 13$，可得

$$d(A, T) = \min\{4 + 18, 11 + 13\} = \min\{22, 24\} = 22$$

当然，必须记录下从A到D再到T的最短路径。

类似地，对于B，该步骤如图7-5所示。

$$\begin{aligned} d(B, T) &= \min\{9 + d(D, T), 5 + d(E, T), 16 + d(F, T)\} \\ &= \min\{9 + 18, 5 + 13, 16 + 2\} \\ &= \min\{27, 18, 18\} \\ &= 18 \end{aligned}$$

最后，很容易得到$d(C, T)$为4。

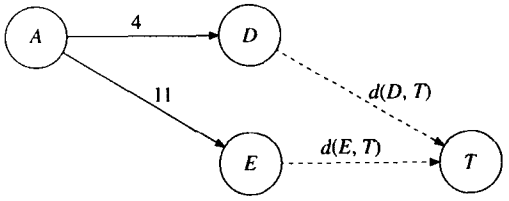


图7-4 使用动态规划方法过程中的一步

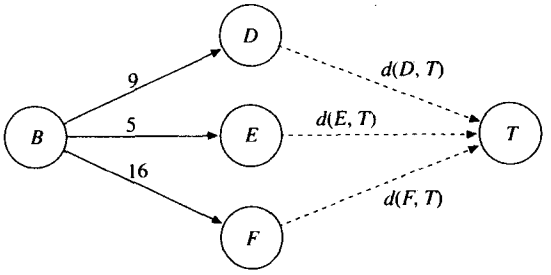


图7-5 使用动态规划方法过程中的一步

(3) 在找到$d(A, T)$、$d(B, T)$和$d(C, T)$}后，可以通过公式(7-1)得到$d(S, T)$的值。

$$\begin{aligned} d(S, T) &= \min\{1 + 22, 2 + 18, 5 + 4\} \\ &= \min\{23, 20, 9\} \\ &= 9 \end{aligned}$$

动态规划的基本原理是把原问题分解到各个子问题，对各个子问题分别用同样的方法递归地解决。将该方法总结如下：

(1) 为了找到从S到T的最短路径，先找从A、B和C到T的最短路径。

(2) (a) 为了找到从A到T的最短路径，要找出从D和E到T的最短路径。

(b) 为了找到从B到T的最短路径，要找出从D、E和F到T的最短路径。

(c) 为了找到从C到T的最短路径，要找出从F到T的最短路径。

(3) 从D、E和F到T的最短路径可以很容易地得到。

(4) 在已知从D、E和F到T的最短路径之后，可以得到从A、B和C到T的最短路径。

(5) 在已知从A、B和C到T的最短路径之后，可以得到从S到T的最短路径。

上面的推导方法实际上是一种反向推导（backward reasoning）方法。接下来，也可以用前向推导（forward reasoning）方法解决此问题。最短路径的求解过程如下：

(1) 首先找到$d(S, A)$、$d(S, B)$和$d(S, C)$。

$d(S, A) = 1$

$d(S, B) = 2$

$d(S, C) = 5$

(2) 然后，确定$d(S, D)$、$d(S, E)$和$d(S, F)$如下：

$$
\begin{aligned}
d(S,\ D) &= \min\{d(A,\ D)+d(S,\ A),\ d(B,\ D)+d(S,\ B)\} \\
&= \min\{4+1,\ 9+2\} \\
&= \min\{5,\ 11\} = 5
\end{aligned}
$$

$$
\begin{aligned}
d(S,\ E) &= \min\{d(A,\ E)+d(S,\ A),\ d(B,\ E)+d(S,\ B)\} \\
&= \min\{11+1,\ 5+2\} \\
&= \min\{12,\ 7\} = 7
\end{aligned}
$$

$$
\begin{aligned}
d(S,\ F) &= \min\{d(B,\ F)+d(S,\ B),\ d(C,\ F)+d(S,\ C)\} \\
&= \min\{16+2,\ 2+5\} \\
&= \min\{18,\ 7\} \\
&= 7
\end{aligned}
$$

(3) 从S到T的最短路径可如下得到：

$$
\begin{aligned}
d(S,\ T) &= \min\{d(D,\ T)+d(S,\ D),\ d(E,\ T)+d(S,\ E),\ d(F,\ T)+d(S,\ F)\} \\
&= \min\{18+5,\ 13+7,\ 2+7\} \\
&= \min\{23,\ 20,\ 9\} \\
&= 9
\end{aligned}
$$

动态规划方法节省计算可以再次通过图7-2进行解释。在图7-2中，有一个解如下：$S\rightarrow B\rightarrow D\rightarrow T$。

此解的长度即$d(S,\ B)+d(B,\ D)+d(D,\ T)$在使用反向推导的动态规划方法时是从不计算的。当使用反向的推导方法时，发现$d(B,\ E)+d(E,\ T)<d(B,\ D)+d(D,\ T)$。也就是说，不需要考虑解：$S\rightarrow B\rightarrow D\rightarrow T$。因为知道$B\rightarrow E\rightarrow T$的长度比$B\rightarrow D\rightarrow T$的长度短。

因此，像分枝限界方法一样，动态规划方法也能帮助我们避免穷尽全部的解空间。

可以说动态规划方法是通过分阶段途径进行的一种排除方法，因为在排除某一个阶段后，许多子解也同时排除了。例如，参见图7-6a。最初有八个解。如果使用动态规划方法，解的数量减少到四个，如图7-6b所示。

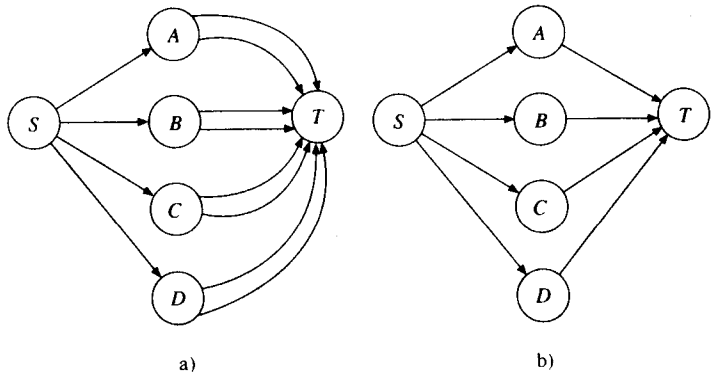


图7-6 用来说明在动态规划方法中排除解的例子

动态规划是基于称为最优化原理（principle of optimality)这个概念的。假设在解决问题时，需要作出一个决策序列D_1，D_2，…，D_n。如果这个序列是最优的，那么最终的$k(1\leqslant k\leqslant n)$个决策也一定是最优的。在后面各节中将对此原理进行多次说明。

应用动态规划方法有两个优点。如前所述，第一个优点是可以排除一些解，减少计算量；另一个优点是可以帮助我们系统化地逐步解决问题。

如果一个解决问题的人员不具备任何动态规划的知识，那么他可能会检查全部可能的组合解来求解一个问题，这极其耗费时间。如果使用动态规划方法，问题会立即变成一个多阶段的问题，因此，可以用非常系统化的方法来解决。随后这将变得更清晰。

7.1 资源配置问题

资源配置问题（resource allocation problem）定义如下：已知m个资源和n个项目。将j $(0\leqslant j\leqslant m)$个资源配置给项目$i$时，可得到利润$P(i, j)$。问题是要找到资源配置的最大总利润。

假设有四个项目和三个资源，利润矩阵（profit matrix）P如表7-1所示，对于每个i，$P(i, 0)=0$。

表7-1 利润矩阵

项目＼资源	1	2	3
1	2	8	9
2	5	6	7
3	4	4	4
4	2	4	5

为了求解该问题，做出一个决策序列。在每次决策时，确定配置给项目i的资源数。不失一般性，必须决定：

(1) 应当给项目1配置多少资源？

(2) 应当给项目2配置多少资源？

⋮

显然，贪心法在这种情况下是无效的。然而，可以用动态规划方法来求解。如果使用动态规划方法，可以设想每次决策将产生一个新的状态。令(i, j)表示i个资源已配置给项目1，2，…，j所得到的状态。那么，最初只有四个描述的状态，如图7-7所示。

在将i个资源配置给项目1之后，至多可以配置$(3-i)$个资源给项目2。因此，第二阶段决策与第一阶段是相关的，如图7-8所示。

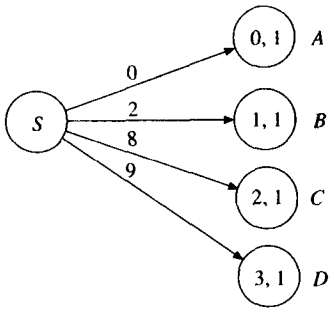


图7-7 资源配置问题的初始阶段决策

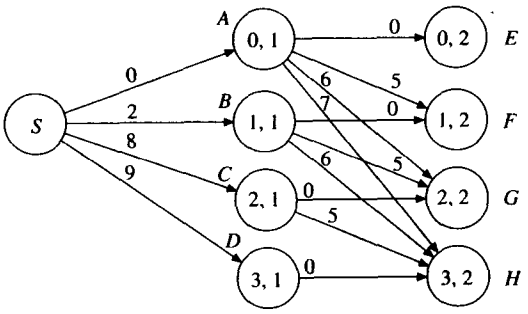


图7-8 资源配置问题的前两阶段的决策

最后，图7-9显示如何表述整个问题。

为了解决资源配置问题，只需要找到从S到T的最长路径，使用反向推导的方法如下：

(1) 从I、J、K和L到T的最长路径如图7-10所示。

(2) 如果已经获得从I、J、K和L到T的最长路径，那么很容易得到从E、F、G和H到T的最长路径。例如，从E到T的最长路径确定如下：

$$
\begin{aligned}
d(E,\ T) &= \max\{d(E,\ I)+d(I,\ T),\ d(E,\ J)+d(J,\ T),\\
&\qquad d(E,\ K)+d(K,\ T),\ d(E,\ L)+d(L,\ T)\}\\
&= \max\{0+5,\ 4+4,\ 4+2,\ 4+0\}\\
&= \max\{5,\ 8,\ 6,\ 4\}\\
&= 8
\end{aligned}
$$

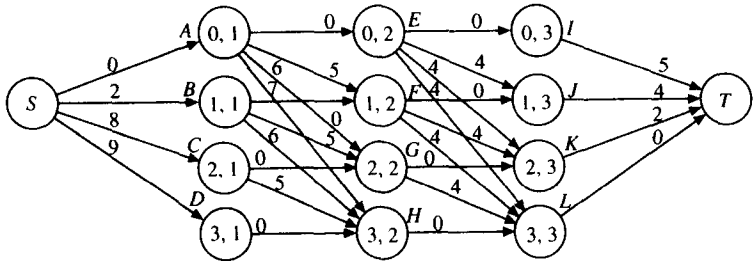


图7-9 资源配置问题表示为多阶段图

图7-11总结了此阶段的结果。

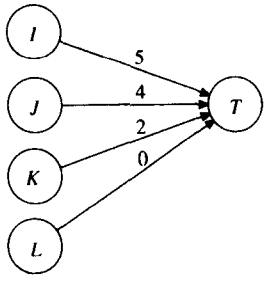


图7-10 从I、J、K和L到T的最长路径

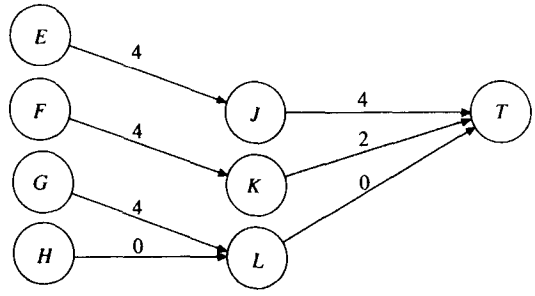


图7-11 从E、F、G和H到T的最长路径

(3) 从A、B、C和D到T的最长路径可以用同样的方法分别得到，如图7-12所示。

(4) 最后，从S到T的最长路径可以按如下方法得到：

$$
\begin{aligned}
d(S,\ T) &= \max\{d(S,\ A)+d(A,\ T),\ d(S,\ B)+d(B,\ T),\\
&\qquad d(S,\ C)+d(C,\ T),\ d(S,\ D)+d(D,\ T)\}\\
&= \max\{0+11,\ 2+9,\ 8+5,\ 9+0\}\\
&= \max\{11,\ 11,\ 13,\ 9\}\\
&= 13
\end{aligned}
$$

最长路径是$S \to C \to H \to L \to T$。

对应的资源配置是

2个资源配置给项目1，

1个资源配置给项目2，

0个资源配置给项目3，

以及0个资源配置给项目4。

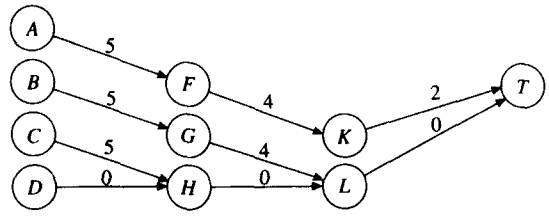


图7-12 从A、B、C和D到T的最长路径

7.2 最长公共子序列问题

考虑一个字符串$A = a\ b\ a\ a\ d\ e$。A的一个子序列是通过从A中删去0个或更多个（不必是连续的）字符得到的。例如，下面字符串都是A的子序列：

a
b
d
$a\ b$
$a\ a$
$b\ d$
$a\ e$
$a\ b\ a$
$a\ a\ a$
$a\ d\ e$
$b\ a\ d\ e$
$a\ a\ a\ d$
$a\ b\ a\ d\ e$

A和B间的公共子序列定义为两个字符串的共同子序列。例如，考虑$A = a\ b\ a\ a\ d\ e\ c$和$B = c\ a\ a\ c\ e\ d\ c$。下面的字符串是A和B的公共子序列：

a
d
c
$a\ d$
$d\ c$
$a\ a$
$a\ c$
$a\ a\ c$
$a\ a\ d$
$a\ a\ e\ c$

最长公共子序列问题（longest common subsequence problem）是要找到两个字符串间最长的公共子序列。许多问题都是最长公共子序列问题的变形。例如，语音识别问题（speech recognition problem）可以看成是最长公共子序列问题。

对于一位初学者，最长公共子序列问题决不是容易解决的。直接的方法是通过穷尽搜索来找到全部的公共子序列。当然，要从可能最长的子序列开始。两个字符串的最长公共子序列的长度不能比原字符串中较短的长。因此，应该从较短的字符串开始。

例如，令$A = a\ b\ a\ a\ d\ e\ c$及$B = b\ a\ f\ c$。可以尝试确定$b\ a\ f\ c$是否公共子序列。可以直观地看出它并不是。接下来从B中选择长度为三的子序列进行尝试，即$a\ f\ c$、$b\ f\ c$、$b\ a\ c$和$b\ a\ f$。因为$b\ a\ c$是一个公共子序列，那么它也一定是最长公共子序列，因为这是从最长的可能子序列开始的。

这种直接方法是非常消耗时间的，因为大量B的子序列数目要与大量A的子序列数目相匹配。因此，这是一个指数增长的过程。

幸运的是，可以使用动态规划方法解决这种最长公共子序列问题。先稍微修改一下初始问题。先不找最长公共子序列，而是尝试确定最长公共子序列的长度。事实上，通过追溯寻找最长公共子序列长度的过程，很容易找到最长公共子序列。

考虑两个字符串$A = a_1a_2\cdots a_m$和$B = b_1b_2\cdots b_n$。注意最后两个字符：a_m和b_n。有两种可能性：

情况1：$a_m = b_n$。在此情况下，最长公共子序列一定包含a_m，所以仅需要找到$a_1a_2\cdots a_{m-1}$和

$b_1b_2\cdots b_{n-1}$的最长公共子序列。

情况2：$a_m \neq b_n$。在此情况下，既可以将$a_1a_2\cdots a_m$与$b_1b_2\cdots b_{n-1}$匹配，也可以将$a_1a_2\cdots a_{m-1}$与$b_1b_2\cdots b_n$匹配。无论哪个产生的较长的最长公共子序列，它都是最长公共子序列。

令$L_{i,j}$表示$a_1a_2\cdots a_i$和$b_1b_2\cdots b_j$的最长公共子序列的长度。$L_{i,j}$可以从下面的递归公式得到：

$$L_{i,j} = \begin{cases} L_{i-1,j-1}+1 & \text{如果 } a_i = b_j \\ \max\{L_{i-1,j}, L_{i,j-1}\} & \text{如果 } a_i \neq b_j \end{cases}$$

$$L_{0,0} = L_{0,j} = L_{i,0} = 0，\text{对于} 1 \leqslant i \leqslant m，1 \leqslant j \leqslant n$$

上面的公式表明可以首先找到$L_{1,1}$。在找到$L_{1,1}$之后，可以找到$L_{1,2}$和$L_{2,1}$。在获得$L_{1,1}$、$L_{1,2}$和$L_{2,1}$后，可以找到$L_{1,3}$、$L_{2,2}$和$L_{3,1}$。整个过程在图7-13中进行了描述。

为了顺序地实现该算法，可依次计算每个$L_{i,j}$的值：$L_{1,1}$，$L_{1,2}$，$L_{1,3}$，…，$L_{1,n}$，$L_{2,1}$，…，$L_{2,n}$，$L_{3,1}$，…，$L_{m,n}$。由于计算每个$L_{i,j}$需要常数的时间，因此算法的时间复杂度为$O(mn)$。

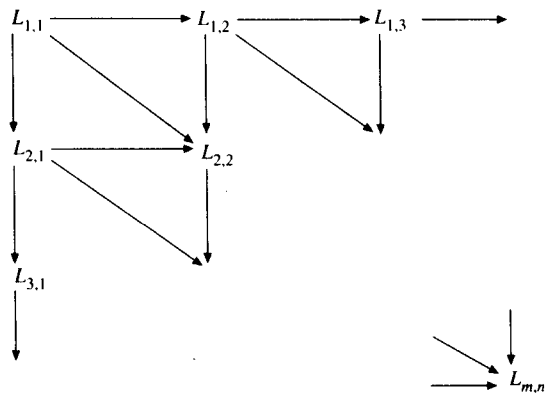


图7-13 用动态规划方法解决最长公共子序列问题

现在用一个例子说明这种动态规划方法。令$A = a\ b\ c\ d$和$B = c\ b\ d$。在这种情况下，我们有

$a_1 = a$

$a_2 = b$

$a_3 = c$

$a_4 = d$

$b_1 = c$

$b_2 = b$

及 $b_3 = d$。

最长公共子序列的长度可以逐步得到，如表7-2所示。

表7-2 $L_{i,j}$的值

b_j \ a_i			a	b	c	d
		0	1	2	3	4
	0	0	0	0	0	0
c	1	0	0	0	1	1
b	2	0	0	1	1	1
d	3	0	0	1	1	2

$L_{i,j}$

7.3 2序列比对问题

令$A = a_1a_2\cdots a_m$和$B = b_1b_2\cdots b_n$为字母集Σ上的两个序列。A与B的序列比对是一个在$\Sigma\cup\{-\}$中的$2\times k$字符矩阵M ($k \geqslant m$，n)，使得M中没有列完全由短横线组成，并且对M中的第一行和第二行的全部短横线进行移动而得到的结果序列分别与A和B相同。例如，如果$A = a\ b\ c\ d$和$B = c\ b\ d$，它们的一个可能的比对是

$a\ b\ c - d$

$- - c\ b\ d$

同样这两个序列的另一个可能的比对是

$a\ b\ c\ d$

$c\ b\ -\ d$

可以看出第二个比对好像比第一个更好。更精确地，需要定义一个可以用来衡量比对性能的得分函数（scoring function）。下面将给出这个函数。可以想像得到，一定还可以定义其他的得分函数。

令$f(x, y)$表示x与y比对的得分。假设x和y都是字符，如果x与y相同，那么$f(x, y) = 2$；如果x与y不同，那么$f(x, y) = 1$；如果x或者y是“−”，那么$f(x, y) = -1$。

一个比对的得分是所有列的字符比对得分的总和，那么下面比对的得分是$-1-1+2-1+2=1$。

$a\ -\ b\ c\ d$

$-\ c\ b\ -\ d$

下面比对的得分是$1+2-1+2=4$。

$a\ b\ c\ d$

$c\ b\ -\ d$

A和B的2序列比对问题（2-sequence alignment problem）是找到具有最多得分的两序列最优比对。与用来找到最长公共序列的递归公式相似，其递归公式也可以公式化。令$A_{i,j}$表示$a_1a_2\cdots a_i$与$b_1b_2\cdots b_j$间的最优比对，其中$1\leqslant i\leqslant m$，$1\leqslant j\leqslant n$。那么，$A_{i,j}$可以表示如下：

$$A_{0,0}=0$$

$$A_{i,0}=i\cdot f(a_i,-)$$

$$A_{0,j}=j\cdot f(-,b_j)$$

$$A_{i,j}=\max\begin{cases}A_{i-1,j}+f(a_i,-)\\A_{i-1,j-1}+f(a_i,b_j)\\A_{i,j-1}+f(-,b_j)\end{cases}$$

有必要将上面的递归公式进行一下解释：

$A_{0,0}$是初始状态。

$A_{i,0}$表示a_1，a_2，…，a_i都与“−”比对。

$A_{0,j}$表示b_1，b_2，…，b_j都与“−”比对。

$A_{i-1,j}+f(a_i, -)$表示a_i与“−”比对，需要在a_1，a_2，…，a_{i-1}与b_1，b_2，…，b_j间找到一个最优比对。

$A_{i-1,j-1}+f(a_i, b_j)$表示a_i与b_j比对，需要在a_1，a_2，…，a_{i-1}与b_1，b_2，…，b_{j-1}间找到一个最优比对。

$A_{i,j-1}+f(-, b_j)$表示b_j与“−”比对，需要在a_1，a_2，…，a_i与b_1，b_2，…，b_{j-1}间找到一个最优比对。

对$A=a\ b\ d\ a\ d$和$B=b\ a\ c\ d$使用上面的递归公式得到的$A_{i,j}$列于表7-3中。

在表7-3中记录了怎样得到每个$A_{i,j}$。从(a_i, b_j)到(a_{i-1}, b_{j-1})的箭头表示a_i与b_j相比对。从(a_i, b_j)到(a_{i-1}, b_j)的箭头表示a_i与“−”相比对。从(a_i, b_j)到(a_i, b_{j-1})的箭头表示b_j与“−”相比对。基于表中的箭头，可以追溯找到最优比对是：

$a\ b\ d\ a\ d$

$-\ b\ a\ c\ d$

现在考虑另一个例子。令两个序列是$A = a\ b\ c\ d$和$B = c\ b\ d$。表7-4表示$A_{i,j}$如何找出。

表7-3 对A = a b d a d和B = b a c d得到的$A_{i,j}$的值

b_j \ a_i			a	b	d	a	d
		0	1	2	3	4	5
	0	0 ↖	−1 ↖	−2	−3	−4	−5
b	1	−1 ↖	1 ↖	1 ←↖	0 ←↖	−1 ←	−2
a	2	−2	1 ↑↖	2 ↖	2 ↖	2 ←↖	1
d	3	−3	0 ↑↖	2 ↑↖	3 ↖	3 ↖	3
d	4	−4	−1	1	4	4	5

表7-4 对A = a b c d和B = c b d得到的$A_{i,j}$的值

b_j \ a_i			a	b	c	d
		0	1	2	3	4
	0	0 ↖	−1 ↖	−2 ↖	−3	−4
c	1	−1 ↖	1 ↑↖ ←	0	0 ←↖	−1
b	2	−2 ↖	0 ↑↖	3 ↑↖ ←	2 ←↖	1
d	3	−3	−1	2	4	4

基于表7-4的箭头，可以追溯并得到其最优比对为

$a\ b\ c\ d$

$c\ b - d$

序列比对可以看成测量两个序列相似性的方法。下面给出一个称为编辑距离的概念，它也经常可以用来测量两个序列的相似性。

考虑两个序列$A = a_1a_2\cdots a_m$和$B = b_1b_2\cdots b_n$。可以通过下面三个编辑操作（edit operation）将A转换成B：从A中删除一个字符，向A中插入一个字符，用另一个字符代替A中的一个字符。例如，令$A = GTAAHTY$，$B = TAHHYC$。A可以通过下面的操作转换成B：

(1) 删除A的第一个字符G，序列A变成$A = TAAHTY$。

(2) 用H替代A的第三个字符A，序列A变成$A = TAHHTY$。

(3) 从A中删除A的第五个字符T，序列A变成$A = TAHHY$。

(4) 在A的最后一个字符后插入C，序列A变成$A = TAHHYC$，它与B相同。

可以给每个操作分配一个代价。编辑距离（edit distance）是把序列A转换成序列B所需的编辑操作的最小代价。如果对每个操作的代价是1，则编辑距离是将A转换成B的编辑操作的最小数目。在上面的例子中，如果每个操作的代价是1，A和B间的编辑距离为4，那么至少需要4次编辑操作。

显然，编辑距离可以用动态规划方法得到。与寻找最大公共子序列或两个序列间的最优比对的递归公式相似，其递归公式可以轻松得到。令α、β和γ分别表示插入、删除和替代的代价，$A_{i,j}$表示$a_1a_2\cdots a_i$与$b_1b_2\cdots b_j$间的编辑距离，那么$A_{i,j}$可以表示如下：

$$A_{0,0} = 0$$
$$A_{i,0} = i\beta$$
$$A_{0,j} = j\alpha$$

$$A_{i,j}=\begin{cases} A_{i-1,j-1} & \text{如果 } a_i=b_j \\ \min\begin{cases} A_{i-1,j}+\beta \\ A_{i-1,j-1}+\gamma \\ a_{i,j-1}+\alpha \end{cases} & \text{其他} \end{cases}$$

实际上，可以很容易地看出，寻找编辑距离问题与最优比对问题是等价的。在这里并不想给出一个形式化的证明，因为这可以从对应的递归公式的相似性中得到。相反，使用上面的例子来说明这一观点。

再次考虑$A = GTAAHTY$和$B = TAHHYC$，产生最优比对如下：

$G\ T\ A\ A\ H\ T\ Y -$

$-\ T\ A\ H\ H - Y\ C$

对上面比对的验证表明编辑操作与比对操作的等价性如下：

(1) 在比对寻找中的(a_i, b_j)与在编辑距离的寻找中的替代操作是等价的。在这种情况下用b_j代替a_i。

(2) 在比对寻找中的$(a_i, -)$与在编辑距离的寻找中删除A中的a_i是等价的。

(3) 在比对寻找中的$(-, b_j)$与在编辑距离的寻找中将b_j插入A中是等价的。

读者可以用上面得到的规则和最优比对来产生这三种编辑操作。对于两个已知的序列$A = a_1a_2\cdots a_m$和$B = b_1b_2\cdots b_n$，需要一个具有$(n+1)(m+1)$项的表格来记录$A_{i,j}$。就是说，对于两个序列$A = a_1a_2\cdots a_m$和$B = b_1b_2\cdots b_n$，找到最优比对需要花费的时间是$O(nm)$。

7.4 RNA最大碱基对匹配问题

核糖核酸（RNA）是一种由核苷酸（碱基）腺嘌呤（A）、鸟嘌呤（G）、胞嘧啶（C）和尿嘧啶（U）构成的单线结构。碱基A、G、C和U的序列称为RNA的基本结构（primary structure）。在RNA中，G和C能够通过一个三氢键构成一个碱基对$G \equiv C$，A和U能够通过一个二氢键构成一个碱基对$A = U$，G和U能够通过一个单氢键构成一个碱基对$G-U$。由于这些氢键，RNA的基本结构可以在自身折叠形成其次级结构（secondary structure）。例如，假设有下面的RNA序列。

$$A-G-G-C-C-U-U-C-C-U$$

那么，这个序列可以在自身折叠形成许多可能的次级结构。在图7-14中显示了该序列可能的六个次级结构。然而，本质上只存在一个次级结构与RNA的序列相对应。那么RNA序列真正的次级结构是什么呢？

根据热力学假说，RNA序列实际的次级结构具有最小自由能量，这将在后面解释。本质上，只有稳定的结构才能够存在，稳定的结构必定具有最小自由能量。在RNA的次级结构中，碱基对会提高结构的稳定性，未配对的碱基将降低结构的稳定性。$G \equiv C$和$A = U$类型的碱基对（称为Watson-Crick碱基对）比$G-U$类型的碱基对（称为wobble碱基对）更稳定。根据这些事实，可以发现图7-14f所示的次级结构是序列$A-G-G-C-C-U-U-C-C-U$实际的次级结构。

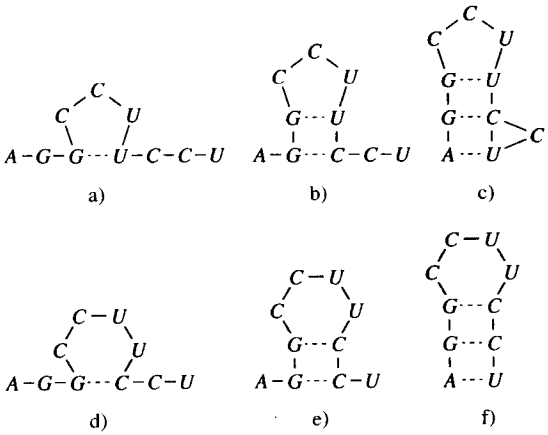


图7-14 RNA序列A-G-G-C-C-U-U-C-C-U的六个可能的次级结构（虚线表示氢键）

一个RNA序列可以用由n个字符的字符串$R = r_1r_2\cdots r_n$表示，其中$r_i \in \{A, C, G, U\}$。n的典型范围是从20到2000。R的次级结构（secondary structure）是碱基对(r_i, r_j)的集合S，其中$1 \leqslant i<j \leqslant n$，满足下面的条件：

(1) $j-i>t$，其中t是小的正常量，典型的值$t=3$。

(2) 如果(r_i, r_j)和(r_k, r_l)是S中的两个碱基对，且$i \neq k$，那么

(a) $i=k$且$j=l$，也就是(r_i, r_j)和(r_k, r_l)是相同的碱基对，

(b) $i<j<k<l$，也就是(r_i, r_j)先于(r_k, r_l)，或者

(c) $i<k<l<j$，也就是(r_i, r_j)包含(r_k, r_l)。

第一种情况表示RNA序列并不是太剧烈地自身折叠。第二种情况表示每一个核苷至多可以参与一个碱基对，而且确保次级结构不包含虚结。如果$i<k<l<j$，那么两个碱基对(r_i, r_j)和(r_k, r_l)称为虚结（pseudoknot）。虚结确实在RNA分子中存在，但排除它们则简化了问题。

对次级结构预测的目的是找到一个有最小自由能量的次级结构。因此，必须有一个方法来计算次级结构S的自由能量。因为碱基对的形式会对结构的自由能量给出稳定的影响，测量S的自由能量最简单方法是将能量分配到S的每个碱基对，这时S的自由能量是全部碱基对能量的总和。由于不同的氢键，碱基对的能量通常分配不同的值。例如，$G \equiv C$，$A = U$和$G-U$的合理值分别为-3，-2和-1。其他可能的值是碱基对的能量全都相等。在这种情况下，问题变成一个寻找具有最大数量碱基对的次级结构。次级结构预测问题的这种说法也称为RNA最大碱基对匹配问题（RNA maximum base pair matching problem），由于可以把次级结构看成是一种匹配问题。接下来，引入动态规划来解决RNA最大碱基对匹配问题，定义如下：已知一个RNA序列$R = r_1r_2\cdots r_n$，找到具有最大数目碱基对的RNA次级结构。

令$S_{i,j}$表示在子串$R_{i,j} = r_ir_{i+1}\cdots r_j$上最大数目碱基对的次级结构，用$M_{i,j}$表示在$S_{i,j}$中已匹配的碱基对。注意并不是任何两个碱基$r_i$和$r_j$都能相互配对，其中$1 \leqslant i<j \leqslant n$。在这里考虑可接受的碱基对是Watson-Crick碱基对（也就是$G \equiv C$和$A = U$）和wobble碱基对（也就是$G-U$）。令$WW = \{(A, U), (U, A), (G, C), (C, G), (G, U), (U, G)\}$，那么用函数$\rho(r_i, r_j)$表示任意两个碱基$r_i$和$r_j$是否合理的碱基对：

$$\rho(r_i, r_j) = \begin{cases} 1 & 若\ (r_i, r_j) \in WW \\ 0 & 否则 \end{cases}$$

根据定义，知道RNA序列在自身上不能太剧烈地折叠。就是说，如果$j-i \leqslant 3$，那么r_i和r_j就不是$S_{i,j}$的碱基对。因此，如果$j-i \leqslant 3$，那么令$M_{i,j}=0$。

要计算$M_{i,j}$，其中$j-i>3$。从r_j的视角考虑下面的情形。

情形1：在最优解中，r_j与任何其他碱基都不匹配。在此情况下，找到$r_ir_{i+1}\cdots r_j$与$M_{i,j} = M_{i,j-1}$的最优解（参见图7-15）。

情形2：在最优解中，r_j与r_i配对。在此情况下，找到$r_ir_{i+1}\cdots r_{j-1}$和$M_{i,j} = 1 + M_{i+1,j-1}$的最优解（参见图7-16）。

情形3：在最优解中，r_j与某个r_k配对，其中$i+1 \leqslant k \leqslant j-4$。在此情况下，找到$r_ir_{i+1}\cdots r_{k-1}$与$r_kr_{k+1}\cdots r_{j-1}$与$M_{i,j} = 1 + M_{i,k-1} + M_{k+1,j-1}$的最优解（参见图7-17）。

因为要找到$i+1$与$j+4$间的k使得$M_{i,j}$最大，我们得到

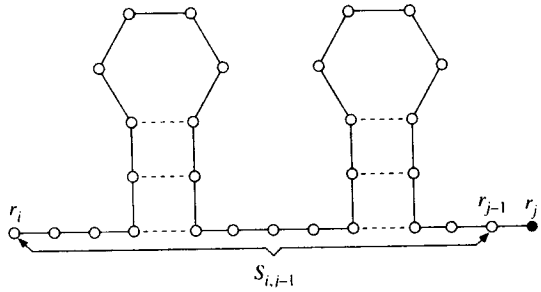


图7-15 情形1的图示

$$M_{i,j} = \max_{i+1 \le k \le j-4} \{1 + M_{i,k-1} + M_{k+1,j-1}\}$$

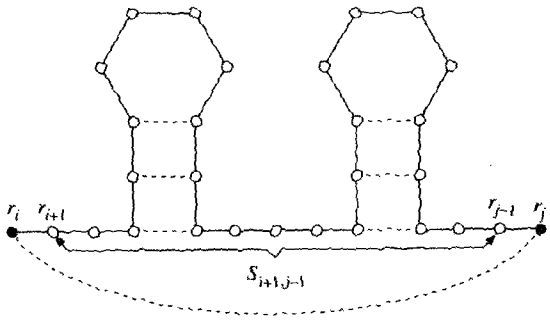


图7-16 情形2的图示

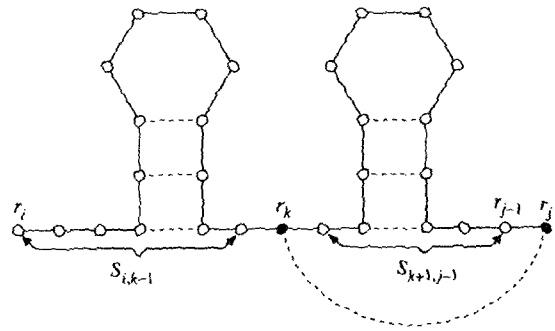


图7-17 情形3的图示

总之，我们有下面计算$M_{i,j}$的递归公式。

如果$j-i \leqslant 3$，那么$M_{i,j}=0$。

如果$j-i>3$，那么$M_{i,j} = \max\begin{cases} M_{i,j-1} \\ (1+M_{i+1,j-1}) \times \rho(r_i, r_j) \\ \max\limits_{i+1 \le k \le j-4} \{1 + M_{i,k-1} + M_{k+1,j-1}\} \times \rho(r_k, r_j) \end{cases}$

根据上面的公式，可以设计应用动态规划技术计算$M_{1,n}$的算法7-1。表7-5说明了对$M_{i,j}$的计算过程，其中$1 \leqslant i < j \leqslant 10$，对于某个RNA序列$R_{1,10}$ = A–G–G–C–C–U–U–C–C–U的计算。由于$M_{1,10}=3$，可以找到$S_{1,10}$中碱基对的最大数是3。

表7-5 RNA序列A-G-G-C-C-U-U-C-C-U的最大碱基对数的计算

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	1	2	2	2	3
2	—	0	0	0	0	1	1	2	2	2
3	—	—	0	0	0	0	1	1	1	1
4	—	—	—	0	0	0	0	0	0	0
5	—	—	—	—	0	0	0	0	0	0
6	—	—	—	—	—	0	0	0	0	0
7	—	—	—	—	—	—	0	0	0	0
8	—	—	—	—	—	—	—	0	0	0
9	—	—	—	—	—	—	—	—	0	0
10	—	—	—	—	—	—	—	—	—	0

算法7-1 RNA最大碱基对匹配算法

输入：RNA序列$R = r_1 r_2 \cdots r_n$。

输出：找出具有最大碱基对数的RNA次级结构。

步骤1. /*对于$1 \leqslant i < j \leqslant n$，计算函数$\rho(r_i, r_j)$的值*/

```
    WW = {(A, U), (U, A), (G, C), (C, G), (G, U), (U, G)};
    for i = 1 to n do
        for j = i to n do
            if (r_i, r_j)∈WW then ρ(r_i, r_j) = 1; else ρ(r_i, r_j) = 0;
        end for
    end for
```

（续）

```
步骤2. /*对于j-i≤3，初始化M_{i,j}*/
    for i = 1 to n do
        for j = i to i + 3 do
            if j≤n then M_{i,j} = 0;
        end for
    end for
步骤3. /*对于j-i >3，计算M_{i,j}*/
    for h = 4 to n-1 do
        for i = 1 to n-h do
            j = i + h;
            case1 = M_{i,j-1};
            case2 = (1 + M_{i+1,j-1}) × ρ(r_i, r_j);
            case3 = M_{i,j} max_{i+1≤k≤j-4} {(1 + M_{i,k-1} + M_{k+1,j-1}) × ρ(r_k, r_j)}
            M_{i,j} = max{case1, case2, case3};
        end for
    end for
```

现在给出整个过程的详细说明。

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
A	G	G	C	C	U	U	C	C	U

(1) $i=1$，$j=5$，$\rho(r_1, r_5)=\rho(A, C)=0$

$$M_{1,5}=\max\begin{cases}M_{1,4}\\(1+M_{2,4})\times\rho(r_1,r_5)\end{cases}$$
$$=\max\{0, 0\}=0。$$

(2) $i=2, j=6, \rho(r_2, r_6)=\rho(G, U)=1$

$$M_{2,6}=\max\begin{cases}M_{2,5}\\(1+M_{3,5})\times\rho(r_2,r_6)\end{cases}$$
$$=\max\{0, (1+0)\times 1\}=\max\{0, 1\}=1。$$

r_2与r_6相匹配。

(3) $i=3, j=7, \rho(r_3, r_7)=\rho(G, U)=1$

$$M_{3,7}=\max\begin{cases}M_{3,6}\\(1+M_{4,6})\times\rho(r_3,r_7)\end{cases}$$
$$=\max\{0, (1+0)\times 1\}=\max\{0, 1\}=1。$$

r_3与r_7相匹配。

(4) $i=4, j=8, \rho(r_4, r_8)=\rho(C, C)=0$

$$M_{4,8}=\max\begin{cases}M_{4,7}\\(1+M_{5,7})\times\rho(r_4,r_8)\end{cases}$$
$$=\max\{0, (1+0)\times 1\}=0。$$

(5) $i=5, j=9, \rho(r_5, r_9)=\rho(C, C)=0$

$$M_{5,9}=\max\begin{cases}M_{5,8}\\(1+M_{6,8})\times\rho(r_5,r_9)\end{cases}$$

$$= \max\{0, (1+0)\times 1\} = \max\{0, 0\} = 0。$$

(6) $i = 6, j = 10, \rho(r_6, r_{10}) = \rho(U, U) = 0$

$$M_{6,10} = \max\begin{cases} M_{6,9} \\ (1+M_{7,9})\times\rho(r_6, r_{10}) \end{cases}$$

$$= \max\{0, (1+0)\times 0\} = \max\{0, 0\} = 0。$$

(7) $i = 1, j = 6, \rho(r_1, r_6) = \rho(A, U) = 1$

$$M_{1,6} = \max\begin{cases} M_{1,5} \\ (1+M_{2,5})\times\rho(r_1, r_6) \\ (1+M_{1,1}+M_{3,5})\times\rho(r_2, r_6) \end{cases}$$

$$= \max\{0, (1+0)\times 1, (1+0+0)\times 1\} = \max\{0, 1, 1\} = 1。$$

r_1与r_6相匹配。

(8) $i = 2, j = 7, \rho(r_1, r_6) = \rho(G, U) = 1$

$$M_{2,7} = \max\begin{cases} M_{2,6} \\ (1+M_{3,6})\times\rho(r_2, r_7) \\ (1+M_{2,2}+M_{4,6})\times\rho(r_3, r_7) \end{cases}$$

$$= \max\{1, (1+0)\times 1, (1+0+0)\times 1\} = \max\{1, 1, 1\} = 1。$$

r_2与r_7相匹配。

(9) $i = 3, j = 8, \rho(r_3, r_8) = \rho(G, C) = 1$

$$M_{3,8} = \max\begin{cases} M_{3,7} \\ (1+M_{4,7})\times\rho(r_3, r_8) \\ (1+M_{3,3}+M_{5,7})\times\rho(r_4, r_8) \end{cases}$$

$$= \max\{1, (1+0)\times 1, (1+0+0)\times 0\} = \max\{1, 1, 0\} = 1。$$

r_3与r_8相匹配。

(10) $i = 4, j = 9, \rho(r_4, r_9) = \rho(C, C) = 0$

$$M_{4,9} = \max\begin{cases} M_{4,8} \\ (1+M_{5,8})\times\rho(r_4, r_9) \\ (1+M_{4,4}+M_{6,8})\times\rho(r_5, r_9) \end{cases}$$

$$= \max\{0, (1+0)\times 0, (1+0+0)\times 0\} = \max\{0, 0, 0\} = 0。$$

(11) $i = 5, j = 10, \rho(r_5, r_{10}) = \rho(C, C) = 0$

$$M_{5,10} = \max\begin{cases} M_{5,9} \\ (1+M_{6,9})\times\rho(r_5, r_{10}) \\ (1+M_{5,5}+M_{7,9})\times\rho(r_6, r_{10}) \end{cases}$$

$$= \max\{0, (1+0)\times 0, (1+0+0)\times 0\} = \max\{0, 0, 0\} = 0。$$

(12) $i = 1, j = 7, \rho(r_1, r_7) = \rho(A, U) = 1$

$$M_{1,7} = \max\begin{cases} M_{1,6} \\ (1+M_{2,6})\times\rho(r_1, r_7) \\ (1+M_{1,1}+M_{3,6})\times\rho(r_2, r_7) \\ (1+M_{1,2}+M_{4,6})\times\rho(r_3, r_7) \end{cases}$$

$$= \max\{1, (1+1)\times 1, (1+0+0)\times 1, (1+0+0)\times 1\} = \max\{1, 2, 1, 1\} = 2。$$

(13) $i = 1, j = 8, \rho(r_2, r_8) = \rho(G, C) = 1$

$$M_{2,8} = \max\begin{cases} M_{2,7} \\ (1+M_{3,7})\times\rho(r_2,r_8) \\ (1+M_{2,2}+M_{4,7})\times\rho(r_3,r_8) \\ (1+M_{2,3}+M_{5,7})\times\rho(r_4,r_8) \end{cases}$$

$$= \max\{1, (1+1)\times 1, (1+0+0)\times 1, (1+0+0)\times 0\} = \max\{1, 2, 1, 0\} = 2。$$

r_2与r_8相匹配；r_3与r_7相匹配。

(14) $i = 3, j = 9, \rho(r_3, r_9) = \rho(G, C) = 1$

$$M_{3,9} = \max\begin{cases} M_{3,8} \\ (1+M_{4,8})\times\rho(r_3,r_9) \\ (1+M_{3,3}+M_{5,8})\times\rho(r_4,r_9) \\ (1+M_{3,4}+M_{6,8})\times\rho(r_5,r_9) \end{cases}$$

$$= \max\{1, (1+0)\times 1, (1+0+0)\times 0, (1+0+0)\times 0\} = \max\{1, 1, 0, 0\} = 1。$$

r_3与r_9相匹配。

(15) $i = 4, j = 10, \rho(r_4, r_{10}) = \rho(C, U) = 0$

$$M_{4,10} = \max\begin{cases} M_{4,9} \\ (1+M_{5,9})\times\rho(r_4,r_{10}) \\ (1+M_{4,4}+M_{6,9})\times\rho(r_5,r_{10}) \\ (1+M_{4,5}+M_{7,9})\times\rho(r_6,r_{10}) \end{cases}$$

$$= \max\{0, (1+0)\times 0, (1+0+0)\times 0, (1+0+0)\times 0\}$$
$$= \max\{0, 0, 0, 0\} = 0。$$

(16) $i = 1, j = 8, \rho(r_1, r_8) = \rho(A, C) = 0$

$$M_{1,8} = \max\begin{cases} M_{1,7} \\ (1+M_{2,7})\times\rho(r_1,r_8) \\ (1+M_{1,1}+M_{3,7})\times\rho(r_2,r_8) \\ (1+M_{1,2}+M_{4,7})\times\rho(r_3,r_8) \\ (1+M_{1,3}+M_{5,7})\times\rho(r_4,r_8) \end{cases}$$

$$= \max\{2, (1+1)\times 0, (1+0+1)\times 1, (1+0+0)\times 1, (1+0+0)\times 0\}$$
$$= \max\{2, 0, 1, 1, 0\} = 2。$$

r_1与r_7相匹配；r_2与r_6相匹配。

(17) $i = 2, j = 9, \rho(r_2, r_9) = \rho(G, C) = 1$

$$M_{2,9} = \max\begin{cases} M_{2,8} \\ (1+M_{3,8})\times\rho(r_2,r_9) \\ (1+M_{2,2}+M_{4,8})\times\rho(r_3,r_9) \\ (1+M_{2,3}+M_{5,8})\times\rho(r_4,r_9) \\ (1+M_{2,4}+M_{6,8})\times\rho(r_5,r_9) \end{cases}$$

$$= \max\{2, (1+1)\times 1, (1+0+0)\times 1, (1+0+0)\times 0, (1+0+0)\times 0\}$$
$$= \max\{2, 2, 1, 0, 0\} = 2。$$

r_2与r_9相匹配；r_3与r_8相匹配。

(18) $i=3, j=10, \rho(r_3, r_{10})=\rho(G, U)=1$

$$M_{3,10}=\max\begin{cases}M_{3,9}\\(1+M_{4,9})\times\rho(r_3,r_{10})\\(1+M_{3,3}+M_{5,9})\times\rho(r_4,r_{10})\\(1+M_{3,4}+M_{6,9})\times\rho(r_5,r_{10})\\(1+M_{3,5}+M_{7,9})\times\rho(r_6,r_{10})\end{cases}$$

$$=\max\{1,(1+0)\times1,(1+0+0)\times0,(1+0+0)\times0,(1+0+0)\times0\}$$

$$=\max\{1,1,0,0,0\}=1。$$

r_3与r_{10}相匹配。

(19) $i=1, j=9, \rho(r_1, r_9)=\rho(A, C)=0$

$$M_{1,9}=\max\begin{cases}M_{1,8}\\(1+M_{2,8})\times\rho(r_1,r_9)\\(1+M_{1,1}+M_{3,8})\times\rho(r_2,r_9)\\(1+M_{1,2}+M_{4,8})\times\rho(r_3,r_9)\\(1+M_{1,3}+M_{5,8})\times\rho(r_4,r_9)\\(1+M_{1,4}+M_{6,8})\times\rho(r_5,r_9)\end{cases}$$

$$=\max\{2,(1+2)\times0,(1+0+1)\times1,(1+0+0)\times1,(1+0+0)\times0,(1+0+0)\times0\}$$

$$=\max\{2,0,1,1,0,0\}=2。$$

r_1与r_7相匹配；r_2与r_6相匹配。

(20) $i=2, j=10, \rho(r_2, r_{10})=\rho(G, U)=1$

$$M_{2,10}=\max\begin{cases}M_{2,9}\\(1+M_{3,9})\times\rho(r_2,r_{10})\\(1+M_{2,2}+M_{4,9})\times\rho(r_3,r_{10})\\(1+M_{2,3}+M_{5,9})\times\rho(r_4,r_{10})\\(1+M_{2,4}+M_{6,9})\times\rho(r_5,r_{10})\\(1+M_{2,4}+M_{7,9})\times\rho(r_6,r_{10})\end{cases}$$

$$=\max\{2,2,1,1,0,0\}=2。$$

r_2与r_{10}相匹配；r_3与r_9相匹配。

(21) $i=1, j=10, \rho(r_1, r_{10})=\rho(A, U)=1$

$$M_{1,10}=\max\begin{cases}M_{1,9}\\(1+M_{2,9})\times\rho(r_1,r_{10})\\(1+M_{1,1}+M_{3,9})\times\rho(r_2,r_{10})\\(1+M_{1,2}+M_{4,9})\times\rho(r_3,r_{10})\\(1+M_{1,3}+M_{5,9})\times\rho(r_4,r_{10})\\(1+M_{1,4}+M_{6,9})\times\rho(r_5,r_{10})\\(1+M_{1,5}+M_{7,9})\times\rho(r_6,r_{10})\end{cases}$$

$$=\max\{2,3,2,1,0,0,0\}=3。$$

r_1与r_{10}相匹配；r_2与r_9相匹配；r_3与r_8相匹配。

接下来，分析算法7-1的时间复杂度。步骤1的代价很明显是$O(n)$。对于步骤2，至多有$\sum_{i=1}^{n}\sum_{j=i+4}^{n}$次迭代，每次迭代都花费$O(j-i)$的时间。因此，步骤2的代价是

$$\sum_{i=1}^{n}\sum_{j=i+4}^{n}O(j-i)=O(n^3)$$

因此，算法7-1的总时间复杂度是$O(n^3)$。

7.5 0/1背包问题

0/1背包问题（0/1 knapsack problem）已经在第6章中讨论过。在本节中，说明此问题可以用动态规划方法轻松地求解。0/1背包问题定义如下：已知有n个物体和一个背包，物体i的重量为W_i，背包的容量为M。如果物体i放入背包中，那么可获得的利润为P_i。0/1背包问题是在所选择的所有物体总重量不超过M的条件下，获得最大的总利润。

需要采取一系列的行动。令X_i为变量，表示物体i是否被选择。就是说，如果物体i被选择，那么令$X_i=1$；否则令$X_i=0$。如果X_1赋值为1（物体1被选择），那么剩下的问题变成修正的0/1背包问题，其中M变成$M-W_1$。通常，在做出了X_1，X_2，…，X_i表示的决策序列后，问题简化为包含决策X_{i+1}，X_{i+2}，…，X_n和$M'=M-\sum_{j=1}^{i}X_jW_j$的问题。因此，无论决策$X_1$，$X_2$，…，$X_i$是什么，剩余的决策$X_{i+1}$，$X_{i+2}$，…，$X_n$必须关于新的背包问题是最优的。下面说明0/1背包问题可以用多阶段图问题来表示。假设有三个物体和一个容量为10的背包。这些物体的重量和利润如表7-6所示。

表7-6 三个物体的重量和利润

i	W_i	P_i
1	10	40
2	3	20
3	5	30

用动态规划方法解决这个0/1背包问题可以用图7-18来说明。

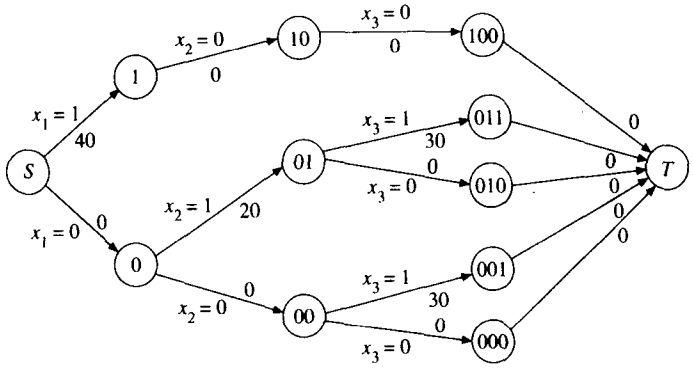


图7-18 用动态规划方法来解决0/1背包问题

在每个结点中，有标签来标记一直到该点已经做出的决策。例如，011意味着$X_1=0$，$X_2=1$和$X_3=1$。在这种情况下，我们对最长路径感兴趣，很明显，最长路径是$S\to0\to01\to011\to T$，对应于

$X_1=0$，

$X_2=1$，

和 $X_3=1$

总的代价等于$20+30=50$。

7.6 最优二叉树问题

二叉树（binary tree）可能是最常见的数据结构之一。已知n个标识符$a_1<a_2<\cdots<a_n$，可以有许多不同的二叉树。例如，假设有四个标识符3，7，9和12。图7-19列出了对于这个数据集的四棵不同二叉树。

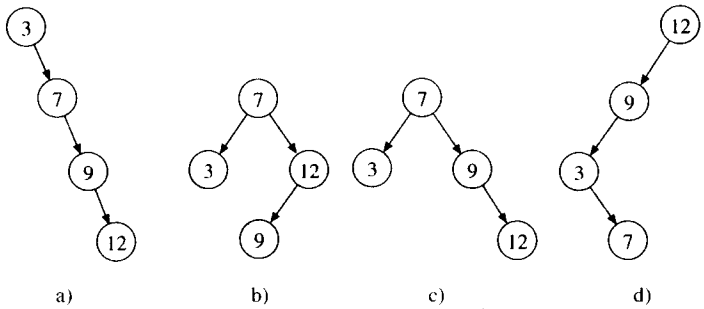


图7-19 对于同一个数据集的四棵不同二叉树

对于给定的二叉树，与经过多步检索才能找到存储在远离树根的数据相比，存储在离树根较近的标识符能更快得到检索。对于每个标识符a_i，添加一个概率P_i，它是这个标识符能够搜索到的概率。对于没有存储在树中的标识符，假设也有分配给它的概率。将不在树中的数据划分成$n+1$个等价类。定义Q_i为检索X的概率，其中$a_i<X<a_{i+1}$，并假设a_0是$-\infty$，a_{n+1}是$+\infty$。概率满足下面的等式：

$$\sum_{i=1}^{n} P_i + \sum_{i=0}^{n} Q_i = 1$$

很容易确定成功检索需要的步数。令$L(a_i)$表示a_i存储在二叉树的层次。如果令树根在第1层，那么检索a_i需要$L(a_i)$步。对于不成功的搜索，最好的理解方法是在二叉树上添加外部结点。现在考虑具有标识符4，5，8，10，11，12和14的情况。对于这个数据集，将划分整个区域如下：

	4，	5，	8，	10，	11，	12，	14	
↑	↑	↑	↑	↑	↑	↑	↑	
E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	

假设有如图7-20所示的二叉树。在这棵二叉树中，有些结点不含两个子孙结点。不成功的检索通常发生在这些结点上。可以在树上加上用方框表示的虚拟的结点，如图7-21所示。

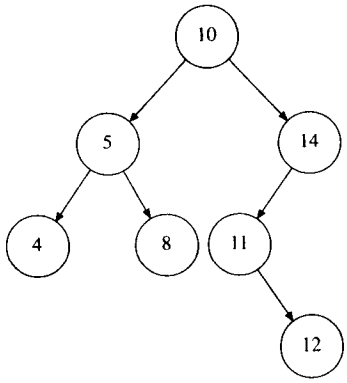


图7-20 二叉树

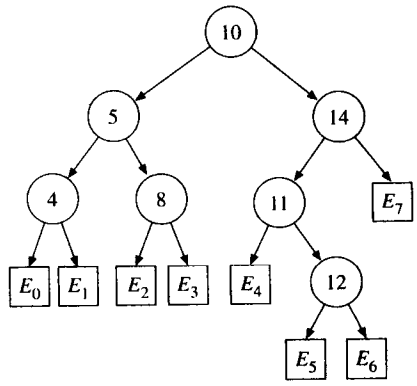


图7-21 添加了外部结点的二叉树

所有其他结点称为内部结点（internal node）。这些添加的结点称为外部结点（external node）。不成功的检索总是终止在这些外部结点上，而成功检索终止在内部结点上。

在添加了外部结点后，二叉树期望的代价值可以表示如下：

$$\sum_{i=1}^{n} P_i L(a_i) + \sum_{i=0}^{n} Q_i (L(E_i) - 1)$$

最优二叉树（optimal binary tree）是上式代价值最小的二叉树。应注意通过穷尽检测全部可能的二叉树来解决最优二叉树问题是相当耗费时间的。已知n个标识符，用这n个标识符构造的全部可能的相异二叉树的数目是$\frac{1}{n+1}\binom{2n}{n}$，近似为$O(4^n/n^{3/2})$。

为什么可以用动态规划方法来解决最优二叉树问题呢？找到最优二叉树关键的第一步是选择一个标识符作为树根，比如是a_k。在选择好a_k后，全部小于a_k的标识符构造为a_k的左后代，全部大于a_k的标识符构造在a_k的右后代，如图7-22所示。

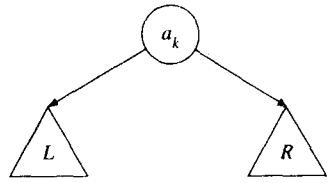


图7-22 在选择a_k作为根后的二叉树

我们不能简单地确定应该选择哪个a_k作为根，需要检查全部可能的a_k。然而，一旦a_k被选定，注意到子树L和R一定也是最优的。也就是，在a_k选定之后，我们的工作简化为找到比a_k小的标识符和比a_k大的标识符的最优二叉树。可以递归地解决该问题表明可以使用动态规划方法。

为什么动态规划方法适用于找到最优二叉树？首先，该方法给了我们一个系统化的思考方法。为了说明这点，假设有五个标识符：1，2，3，4和5。假设3选作为树根，如图7-23所示，左子树包含1和2，右子树包含4和5。

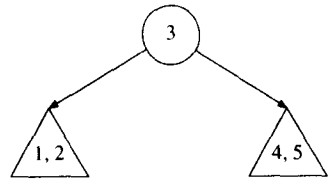


图7-23 以特定标识符选为根的二叉树

下一步要分别找到对1和2及4和5的最优二叉树。对于包含1和2的二叉树，选作根只有两种选择，1或者2。对于包含4和5的二叉树也同样。换句话说，以3作为根的最优二叉树由下面的四棵子树确定：

(a) 包含1和2且以1为根的子树。

(b) 包含1和2且以2为根的子树。

(c) 包含4和5且以4为根的子树。

(d) 包含4和5且以5为根的子树。

左子树为(a)或者为(b)，右子树为(c)或者为(d)。

现在考虑选择2为根的情况。此时，左子树只包含一个标识符1。右子树包含三个标识符：3，4和5。对于此右子树，必须考虑三种子情况：选择3为根，选择4为根和选择5为根。我们主要关心选择3为根的情况。如果选择3，那么其右子树包含4和5。就是说，对包含4和5的子树而论，还要检查应该选择4或5哪个为根。这样我们断定，为了找到以2为根的最优二叉树，在其他子树中要检查下面的两棵树：

(e) 包含4和5且以4为根的子树。

(f) 包含4和5且以5为根的子树。

很明显，(c) 与 (e) 等价，且 (d) 与 (f) 等价。这意味着，找到以3为根的最优二叉树的方法，对于找到以2为根的最优二叉树也是同样有效的。同理，我们看到要找到以4为根的最优二叉树，需要找到包含1和2的以1或2为根的最优二叉树。正如前面指出的那样，这项工作需要找到以3为根的最优二叉树。

总之，我们注意到，已知a_1，a_2，…，a_n，要找到以a_i为根的最优二叉树的工作，同样在找

到以a_j为根的最优二叉树也需要。这一原理可用到任何层次的所有子树上。因此，动态规划能够通过自底向上方法解决最优二叉树问题。也就是说，从构建较小的最优二叉树开始。应用这些小的最优二叉树来构建越来越大的最优二叉树。当找到包含全部标识符的最优二叉树时，就达到了我们的目标。

为了更明确些，假设要找到含有四个标识符：1，2，3和4的最优二叉树。用记号(a_k，a_i→a_j)表示包含从a_i到a_j的以a_k为根的最优二叉树。用(a_i →a_j)表示包含从a_i 到a_j的标识符的最优二叉树。找出包含1，2，3和4的最优二叉树的动态规划过程描述如下：

(1) 开始时找出

(1，1→2)

(2，1→2)

(2，2→3)

(3，2→3)

(3，3→4)

(4，3→4)

(2) 应用上面的结果，确定

(1→2)(由(1，1→2)和(2，1→2)确定)

(2→3)

(3→4)

(3) 然后寻找

(1，1→3)(由(2→3)确定)

(2，1→3)

(3，1→3)

(2，2→4)

(3，2→4)

(4，2→4)

(4) 应用上面的结果，确定

(1→3)(由(1，1→3)，(2，1→3)和(3，1→3)确定)

(2→4)

(5) 然后找出

(1，1→4)(由(2→4)确定)

(2，1→4)

(3，1→4)

(4，1→4)

(6) 最后，确定

(1→4)

因为它由下面几项决定：

(1，1→4)

(2，1→4)

(3，1→4)

(4，1→4)。

上面四棵具有最小代价的二叉树中任何一棵都是最优的。

读者能够看出动态规划是解决最优二叉树问题的有效方法。它不仅提供了系统的思考方法，

而且避免了冗余的计算。

在描述了用动态规划方法解决最优二叉树问题的基本原理后，现在给出精确的原理。因为对最优二叉树的检索由查找许多最优二叉树构成，这里只简单考虑检索任意子树的通常情形。想象已知一系列标识符a_1，a_2，…，a_n，且a_k是其中之一。如果选择a_k为二叉树的根，那么左（或右）子树将包含全部标识符a_1，…，a_{k-1}(a_{k+1}，…，a_n)。此外，检索a_k需要一步，所以其他的成功搜索需要1加上需要搜索左或右子树的步数，这对于不成功的检索同样适用。

令$C(i, j)$表示包含a_i到a_j的最优二叉树代价，那么以a_k为根的最优二叉树具有下面的代价

$$C(1,n)=\min_{1\le k\le n}\left\{P_k+\left[Q_0+\sum_{l=1}^{k-1}(P_l+Q_l)+C(1,k-1)\right]+\left[Q_k+\sum_{l=k+1}^{n}(P_l+Q_l)+C(k+1,n)\right]\right\}$$

其中P_k是检索根的代价，第二项和第三项分别是检索左子树和右子树的代价。上面的公式可以得到如下任意的$C(i, j)$：

$$\begin{aligned}C(i,j)&=\min_{i\le k\le j}\left\{P_k+\left[Q_{i-1}+\sum_{l=i}^{k-1}(P_l+Q_l)+C(i,k-1)\right]+\left[Q_k+\sum_{l=k+1}^{j}(P_l+Q_l)+C(k+1,j)\right]\right\}\\&=\min_{i\le k\le j}\left\{C(i,k-1)+C(k+1,j)+\sum_{l=i}^{j}(P_l+Q_l)+Q_{i-1}\right\}\end{aligned}$$

例如，如果有四个标识符a_1，a_2，a_3和a_4，目标是找到$C(1, 4)$。由于对于根有四种选择可能，需要计算四棵最优子树的代价，$C(2, 4)$（以a_1为根），$C(3, 4)$（以a_2为根），$C(1, 2)$（以a_3为根）和$C(1, 3)$（以a_4为根）。要计算$C(2, 4)$，需要计算$C(3, 4)$（以a_2为根）和$C(2, 3)$（以a_4为根）。要计算$C(1, 3)$，需要计算$C(2, 3)$（以a_1为根）和$C(1, 2)$（以a_3为根）。它们的关系在图7-24进行说明。

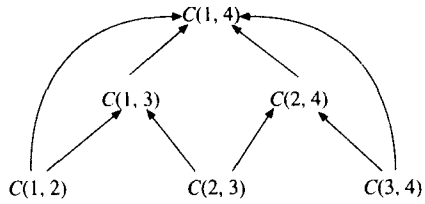


图7-24 子树间的计算关系

总之，给定n个标识符来计算$C(1, n)$，需要首先计算所有$j-i=1$的$C(i, j)$。接下来，计算所有$j-i=2$的$C(i, j)$，然后所有$j-i=3$的$C(i, j)$，等等。现在检验此过程的复杂度。计算过程需要计算对于$j-i=1, 2, \cdots, n-1$的$C(i, j)$。当$j-i=m$时，要计算$(n-m)$个$C(i, j)$。每次$C(i, j)$的计算都需要找到m个数量的最小值。因此，每个具有$j-i=m$的$C(i, j)$都能够在$O(m)$时间内计算得到。计算所有$j-i=m$个$C(i, j)$的总时间是$O(mn-m^2)$。因此，用动态规划方法解决最优二叉查找树问题的总时间复杂度是$O\left(\sum_{1\le m\le n}(mn-m^2)\right)=O(n^3)$。

7.7 树的带权完全支配问题

在带权的完全支配问题（weighted perfect domination problem）问题中，已知图$G=(V, E)$，其中每个顶点$v\in V$有代价$c(v)$，每条边$e\in E$有代价$c(e)$。图G的完全支配集（perfect dominating set）是顶点集V的子集D，使得每个不在D中的顶点恰好邻接于D中的一个顶点。如果v不属于D，u属于D，(u, v)为E中的边，那么完全支配集D的代价包含D中全部点的代价和$c(u, v)$的代价。带权的完全支配是要找出具有最小代价的完全支配集。

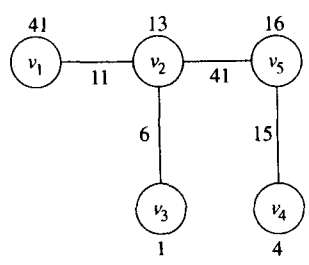


图7-25 一个带权的完全支配问题的图示

参见图7-25。有许多完全支配集。

例如，$D_1 = \{v_1, v_2, v_5\}$是完全支配集，D_1的代价是

$$\begin{aligned} & c(v_1) + c(v_2) + c(v_5) + c(v_3, v_2) + c(v_4, v_5) \\ &= 41 + 13 + 16 + 6 + 15 \\ &= 91 \end{aligned}$$

另一个例子是$D_2 = \{v_2, v_5\}$，D_2的代价是

$$\begin{aligned} & c(v_2) + c(v_5) + c(v_1, v_2) + c(v_3, v_2) + c(v_4, v_5) \\ &= 13 + 16 + 11 + 6 + 15 \\ &= 61 \end{aligned}$$

最后，令$D_3 = \{v_2, v_3, v_4, v_5\}$，$D_3$的代价是

$$\begin{aligned} & c(v_2) + c(v_3) + c(v_4) + c(v_5) + c(v_1, v_2) \\ &= 13 + 1 + 4 + 16 + 11 \\ &= 45 \end{aligned}$$

可以证明，D_3是最小代价完全支配集。

带权的完全支配问题对于二分图和弦图都是NP难的。但我们将说明应用动态规划方法能够解决树的带权完全支配问题。

应用动态规划策略的主要工作是合并两个可行解为一个新的可行解。现在通过一个例子来说明这种合并，参见图7-26。在图7-26中，有两个图G_1和G_2，它们将通过连接G_1中的v_1和G_2中的v_6合并成G。

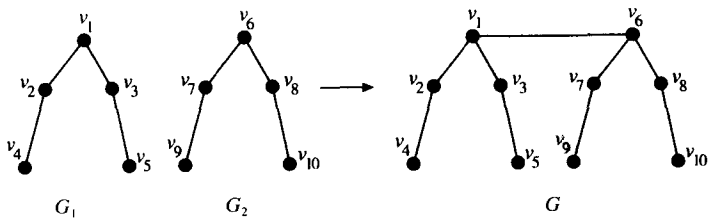


图7-26 一个说明解决带权完全支配问题合并方法的例子

通过连接G_1中的v_1和G_2中的v_6将G_1和G_2组合到一起，可以考虑将v_1作为G_1的根，将v_6作为G_2的根。现在考虑六个完全支配集：

$D_{11} = \{v_1, v_2, v_3\}$，$D_{21} = \{v_6, v_7, v_8\}$

$D_{12} = \{v_3, v_4\}$，　　$D_{22} = \{v_7, v_{10}\}$

$D_{13} = \{v_4, v_5\}$，　　$D_{23} = \{v_9, v_{10}\}$

在图$G_1(G_2)$的根$v_1(v_6)$包含其中的情况下，$D_{11}(D_{21})$是$G_1(G_2)$的完全支配集。

在图$G_1(G_2)$的根$v_1(v_6)$未包含其中的情况下，$D_{12}(D_{22})$是$G_1(G_2)$的完全支配集。

在$v_1(v_6)$中没有邻居包含其中的情况下，$D_{13}(D_{23})$是$G_1-\{v_1\}(G_2-\{v_6\})$的完全支配集。

现在通过合并上面G_1和G_2的完全支配集来产生G的完全支配集。

(1) $D_{11} \cup D_{21} = \{v_1, v_2, v_3, v_6, v_7, v_8\}$是$G$的完全支配集，其代价是$D_{11}$与$D_{21}$的代价和，这个完全支配集包含$v_1$和$v_6$。

(2) $D_{11} \cup D_{23} = \{v_1, v_2, v_3, v_9, v_{10}\}$是$G$的完全支配集，其代价是$D_{11}$与$D_{23}$的代价加上连接$v_1$和$v_6$的边的代价和，这个完全支配集包含$v_1$，但不包含$v_6$。

(3) $D_{12}\cup D_{22}=\{v_3, v_4, v_7, v_{10}\}$是$G$的完全支配集，其代价是$D_{12}$与$D_{22}$的代价和，这个完全支配集不包含$v_1$和$v_6$。

(4) $D_{13}\cup D_{21}=\{v_4, v_5, v_6, v_7, v_8\}$是$G$的完全支配集，其代价是$D_{13}$与$D_{21}$的代价加上连接$v_1$和$v_6$的边的代价和，这个完全支配集不包含$v_1$，但包含$v_6$。

上面的讨论描述了应用动态规划方法解决带权完全支配问题的策略基础。

下面首先假设图是以某点为根的，且当合并两个图时，连接两个根。

进行下面的定义：

$D_1(G, u)$：在完全支配集包含u的情况下，图G的最优完全支配集。$D_1(G, u)$的代价表示为$\delta_1(G, u)$。

$D_2(G, u)$：在完全支配集不包含u的情况下，图G的最优完全支配集。$D_2(G, u)$的代价表示为$\delta_2(G, u)$。

$D_3(G, u)$：在图$G-\{u\}$的完全支配集不包含u的任何邻接点的情况下，图$G-\{u\}$的最优完全支配集。$D_3(G, u)$的代价表示为$\delta_3(G, u)$。

已知图G_1和G_2的根分别为u和v，令图G表示将u和v相连得到的图，很明显，得到下面的规则：

规则1：$D_1(G_1, u)\cup D_1(G_2, v)$和$D_1(G_1, u)\cup D_3(G_2, v)$都是包含$u$的$G$的完全支配集。

规则1.1：$D_1(G_1, u)\cup D_1(G_2, v)$的代价是$\delta_1(G_1, u)+\delta_1(G_2, v)$。

规则1.2：$D_1(G_1, u)\cup D_3(G_2, v)$的代价是$\delta_1(G_1, u)+\delta_3(G_2, v)+c(u, v)$。

规则2：$D_2(G_1, u)\cup D_2(G_2, v)$和$D_3(G_1, u)\cup D_1(G_2, v)$都是不包含$u$的$G$的完全支配集。

规则2.1：$D_2(G_1, u)\cup D_2(G_2, v)$的代价是$\delta_2(G_1, u)+\delta_2(G_2, v)$。

规则2.2：$D_3(G_1, u)\cup D_1(G_2, v)$的代价是$\delta_3(G_1, u)+\delta_1(G_2, v)+c(u, v)$。

应用动态规划策略寻找图的最优完全支配集的基本原理总结如下：

如果图G可分解为两个子图G_1和G_2，通过将G_1的u和G_2的v相连进行G的重构，那么表示为$D(G)$的G的最优完全支配集可以按如下方法得到：

(1) 如果$\delta_1(G_1, u)+\delta_1(G_2, v)$小于$\delta_1(G_1, u)+\delta_3(G_2, v)+c(u, v)$，那么置$D_1(G, u)$为$D_1(G_1, u)\cup D_1(G_2, v)$，$\delta_1(G, u)$为$\delta_1(G_1, u)+\delta_1(G_2, v)$；否则置$D_1(G, u)$为$D_1(G_1, u)\cup D_3(G_2, v)$，$\delta_1(G, u)$为$\delta_1(G_1, u)+\delta_3(G_2, v)+c(u, v)$。

(2) 如果$\delta_2(G_1, u)+\delta_2(G_2, v)$小于$\delta_3(G_1, u)+\delta_1(G_2, v)+c(u, v)$，那么置$D_2(G, u)$为$D_2(G_1, u)\cup D_2(G_2, v)$，$\delta_2(G, u)$为$\delta_2(G_1, u)+\delta_2(G_2, v)$；否则置$D_2(G, u)$为$D_3(G_1, u)\cup D_1(G_2, v)$，$\delta_2(G, u)$为$\delta_3(G_1, u)+\delta_1(G_2, v)+c(u, v)$。

(3) $D_3(G, u)=D_3(G_1, u)+D_2(G_2, v)$。

$\delta_3(G, u)=\delta_3(G_1, u)+\delta_2(G_2, v)$。

(4) 如果$\delta_1(G, u)$小于$\delta_2(G, u)$，那么置$D(G)$为$D_1(G, u)$；否则置$D(G)$为$D_2(G, u)$。

上面的规则适用于一般的图。在下面将考虑树。对于树，使用特殊的从叶子结点出发向内部结点前进的算法。在介绍这种树的特殊算法之前，用一个完整的例子说明这种算法。再次参见图7-25，只从叶子结点v_1出发，仅有两种情况：完全支配集包含v_1，或不包含v_1，如图7-27所示。

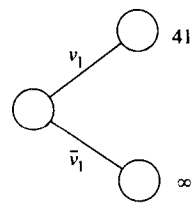


图7-27 包含v_1的完全支配集的计算

很容易得到

$D_1(\{v_1\}, v_1)=\{v_1\}$，

$D_2(\{v_1\}, v_1)$不存在，

$D_3(\{v_1\}, v_1)=\phi$，

$\delta_1(\{v_1\}, v_1)=c(v_1)=41$，

$\delta_2(\{v_1\}, v_1) = \infty$,

和 $\delta_3(\{v_1\}, v_1) = 0$。

现在考虑只包含v_2的子树。可以再次得到：

$D_1(\{v_2\}, v_2) = \{v_2\}$,

$D_2(\{v_2\}, v_2)$不存在,

$D_3(\{v_2\}, v_2) = \phi$,

$\delta_1(\{v_2\}, v_2) = c(v_2) = 13$,

$\delta_2(\{v_2\}, v_2) = \infty$,

和 $\delta_3(\{v_2\}, v_2) = 0$。

考虑包含v_1和v_2的子树，如图7-28所示。

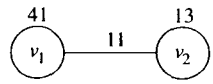


图7-28 包含v_1和v_2的子树

通过动态规划计算$\{v_1, v_2\}$的完全支配集过程如图7-29所示。

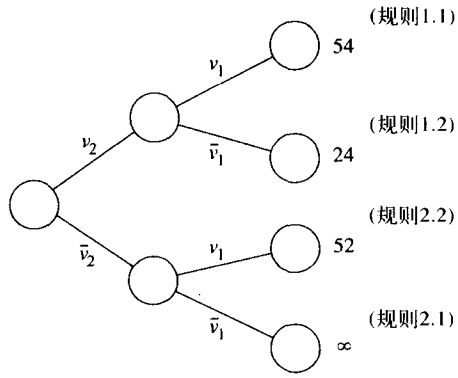


图7-29 对包含v_1和v_2的子树的完全支配集的计算

由于min(54, 24) = 24，得到

$D_1(\{v_1, v_2\}, v_2) = \{v_2\}$,

$\delta_1(\{v_1, v_2\}, v_2) = 24$。

由于min(∞, 52) = 52，得到

$D_2(\{v_2, v_1\}, v_2) = \{v_1\}$,

$\delta_2(\{v_2, v_1\}, v_2) = 52$。

此外，

$D_3(\{v_1, v_2\}, v_2)$不存在,

$\delta_3(\{v_1, v_2\}, v_2) = \infty$。

考虑只包含v_3的树。显然

$D_1(\{v_3\}, v_3) = \{v_3\}$,

$D_2(\{v_3\}, v_3)$不存在,

$D_3(\{v_3\}, v_3) = \phi$,

$\delta_1(\{v_3\}, v_3) = c(v_3) = 1$,

$\delta_2(\{v_3\}, v_3) = \infty$,

和 $\delta_3(\{v_3\}, v_3) = 0$。

将v_3添加到包含v_1和v_2的子树，如图7-30所示。

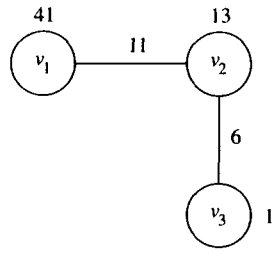


图7-30 包含v_1、v_2和v_3的子树

对于该子树的完全支配集的计算由图7-31进行说明。

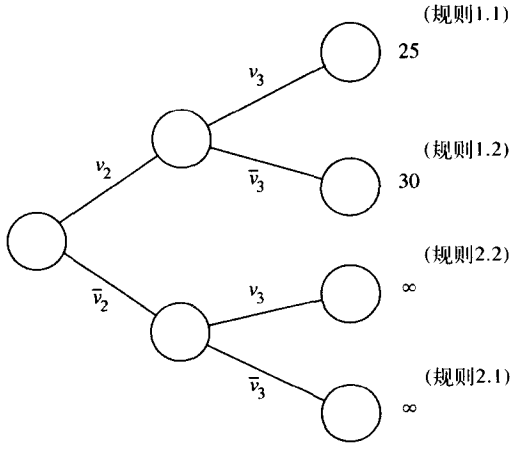


图7-31 对包含v_1、v_2和v_3的子树的完全支配集的计算

通过计算得出

$D_1(\{v_1, v_2, v_3\}, v_2) = \{v_2, v_3\}$,

$D_2(\{v_1, v_2, v_3\}, v_2)$不存在,

$D_3(\{v_1, v_2, v_3\}, v_2)$不存在,

$\delta_1(\{v_1, v_2, v_3\}, v_2) = c(v_2) + c(v_3) + c(v_1, v_2) = 13 + 1 + 11 = 25$,

$\delta_2(\{v_1, v_2, v_3\}, v_2) = \infty$,

和 $\delta_3(\{v_1, v_2, v_3\}, v_2) = \infty$。

考虑如图7-32所示的包含v_5和v_4的子树。

易知下面的结论是正确的。

$D_1(\{v_5, v_4\}, v_5) = \{v_5, v_4\}$,

$D_2(\{v_5, v_4\}, v_5) = \{v_4\}$,

$D_3(\{v_5, v_4\}, v_5)$不存在,

$\delta_1(\{v_5, v_4\}, v_5) = c(v_5) + c(v_4) = 16 + 4 = 20$,

$\delta_2(\{v_5, v_4\}, v_5) = c(v_4) + c(v_5, v_4) = 4 + 15 = 19$,

和 $\delta_3(\{v_5, v_4\}, v_5) = \infty$。

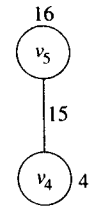


图7-32 包含v_5和v_4的子树

现在考虑包含v_1、v_2和v_3的子树及包含v_4和v_5的子树。就是说，考虑整棵树的情况，对其完全支配集的计算如图7-33所示。

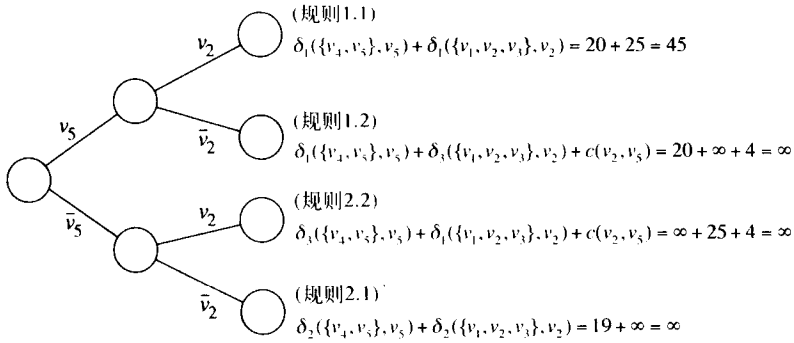


图7-33 对于如图7-25所示的整棵树的完全支配集的计算

可以得出具有最小代价的完全支配集是$\{v_2, v_3, v_4, v_5\}$，其代价为45。

在下面的算法中，令$TP(u)$表示以u为根的树的部分构建过程。

算法7-2 解决树的带权完全支配问题的算法

输入：树$T = (V, E)$，其中每个顶点$v \in V$有代价$c(v)$，每条边$e \in E$有代价$c(e)$。

输出：T的具有最小代价完全支配集$D(T)$。

步骤1. 对于每个顶点$v \in V$，do

$TP(v) = \{v\}$

$D_1(TP(v), v) = TP(v)$

$\delta_1(TP(v), v) = c(v)$

$D_2(TP(v), v)$不存在

$\delta_2(TP(v), v) = \infty$

$D_3(TP(v), v) = \phi$

$\delta_3(TP(v), v) = 0$

步骤2. $T' = T$。

步骤3. while T'有多于1个的顶点 do

选择T'的一片叶子v，它与T'中唯一的顶点u相邻接

置$TP'(u) = TP(u) \cup TP(v) \cup \{u, v\}$

If $(\delta_1(TP(u), u) + \delta_1(TP(v), v)) < (\delta_1(TP(u), u) + \delta_3(TP(v), v) + c(u, v))$

$D_1(TP'(u), u) = D_1(TP(u), u) \cup D_1(TP(v), v)$

$\delta_1(TP'(u), u) = \delta_1(TP(u), u) + \delta_1(TP(v), v)$

Otherwise,

$D_1(TP'(u), u) = D_1(TP(u), u) \cup D_3(TP(v), v)$

$\delta_1(TP'(u), u) = \delta_1(TP(u), u) + \delta_3(TP(v), v) + c(u, v)$

If $(\delta_2(TP(u), u) + \delta_2(TP(v), v)) < (\delta_3(TP(u), u) + \delta_1(TP(v), v) + c(u, v))$

（续）

$D_2(TP'(u),\ u) = D_3(TP(u),\ u) \cup D_2(TP(v),\ v)$

$\delta_2(TP'(u),\ u) = \delta_2(TP(u),\ u) + \delta_2(TP(v),\ v)$

Otherwise,

$D_2(TP'(u),\ u) = D_3(TP(u),\ u) \cup D_1(TP(v),\ v)$

$\delta_2(TP'(u),\ u) = \delta_3(TP(u),\ u) + \delta_1(TP(v),\ v) + c(u,\ v)$

$D_3(TP'(u),\ u) = D_3(TP(u),\ u) \cup D_2(TP(v),\ v)$

$\delta_3(TP'(u),\ u) = \delta_3(TP(u),\ u) + \delta_2(TP(v),\ v)$

$TP(u) = TP'(u)$

$T' = T' - v$;

end while

步骤4. If $\delta_1(TP(u),\ u) < \delta_2(TP(u),\ u)$

置 $D(T) = D_1(TP(u),\ u)$

Otherwise,

置 $D(T) = D_2(TP(u),\ u)$

返回$D(T)$作为$T = (V,\ E)$的最小完全支配集。

7.8 树的带权单步图边的搜索问题

在带权单步图边的搜索问题中，已知一个简单无向图$G(V,\ E)$，其中任意点$v \in V$有权$wt(v)$。假设G的每条边长度相同，一个以任意速度的逃亡者假设隐藏在图G的某条边中。对于G中的每条边，分配一个边的搜寻者来搜索该边。边的搜寻者总是从一个点开始。对于清理一条边$(u,\ v)$的代价，如果搜寻者搜寻起始于u，那么定义为$wt(u)$；如果从v搜寻，那么定义为$wt(v)$。假设每条边的搜寻者以相同的速度搜寻。带权单步图边的搜索问题（weighted single step gragh edge searching problem）是安排边搜寻者的搜索方向，以此种方式使逃亡者可在一步之内被捕获，并且使所用的搜寻者数量最少。

显然，如果有m条边，那么至少需要m位边搜寻者。因为逃亡者的速度与边的搜寻者一样快，并且边搜寻者是向前移动的，如果逃亡者不能偷偷地穿越一个搜索过的点而隐藏在搜寻者的后面，那么搜寻者队伍能够在单步内逮捕到逃亡者。

参见图7-34。在这个案例中，只需要一个边的搜寻者。不管搜寻者初始配置在哪儿，逃亡者都不可能逃脱。假设搜寻者配置在a并向b方向搜索。逃亡者顶多可到b，但是随后搜寻者就会达到b，抓住逃亡者。类似地，如果搜寻者初始配置在b，也可达到同样的目的。也就是，逃亡者在一步之内就会被逮捕。

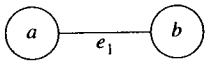


图7-34 一个不需要额外搜寻者的情况

现在，参见图7-35。通过把搜寻者放在a和c，也可以在一步之内逮捕逃亡者。

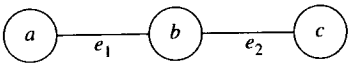


图7-35 另一个不需要额外搜寻者的情况

现在，参见图7-36。

在此情况下，如果只用3个搜寻者，那么不管如何安排搜索方向，逃亡者是可能避免被捕的。例如，考虑下面的搜索方案：

(1) e_1的边搜寻者从a向b搜索。

(2) e_2的边搜寻者从a向c搜索。

(3) e_3的边搜寻者从b向c搜索。

这样，如果逃亡者起初在e_1，他可以通过向e_3移动而不被发现。

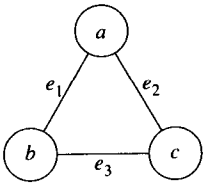


图7-36 一个需要额外搜寻者的情况

假设在点b放一个额外的搜寻者，应用上面的搜索

方法，这样不管逃亡者初始在哪里，都会被发觉。这表明在一些情况下，需要额外的搜寻者。

单步搜索计划是安排边的搜寻者的搜索方向，确定需要的额外搜寻者的最小数量。单步搜索方案的代价是所有搜寻者的代价之和。带权单步搜索问题是用最小的代价找到一种可行的单步搜索方案。对于上面的例子，这个搜索方案的代价是$wt(a) + 2wt(b) + wt(a)$。

带权单步图边的搜索问题对一般的图是NP难的，但是应该指出，应用动态规划策略可以在多项式时间内解决树的带权单步图边的搜索问题。

解决这个问题的动态规划方法是基于几个基本规则，这些规则解释如下。

首先定义一些记号。

令v_i为树中的一个点。

情况1：v_i是一个叶子结点，那么$T(v_i)$表示v_i和它的父结点，如图7-37a所示。

情况2：v_i是一个内部结点，但不是根结点，那么$T(v_i)$表示包含v_i、父结点v_j及所有v_i后代结点的树，如图7-37b所示。

情况3：v_i是树T的根，那么$T(v_i)$表示T，如图7-37c所示。

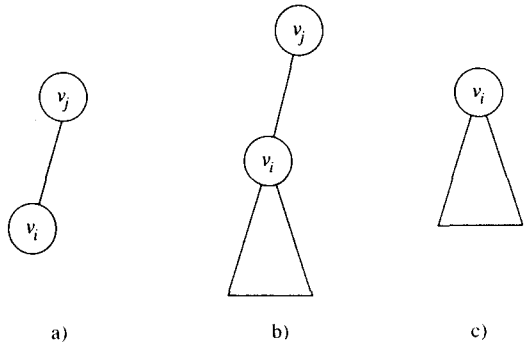


图7-37 $T(v_i)$的定义

进一步地，令v_j是v_i的父结点。$C(T(v_i), v_i, v_j)$（$C(T(v_i), v_j, v_i)$）表示一个最优的单步搜索方案代价，其中(v_i, v_j)的搜索方向是从v_i到v_j（从v_j到v_i）。

规则1：令r是树的根，$C(T(r), r)$($C(T(r), \bar{r})$)是带（不带）配置在r的额外搜寻者的树T的一个最优单步搜索方案的代价。这样，对于树T的一个最优单步搜索方案的代价，表示为$C(T(r))$，是$(C(T(r), r)$与$C(T(r), \bar{r}))$中的小者。

规则2：令r为树根，其中$v_1, v_2, \cdots, v_m$为r的后代结点（见图7-38）。如果没有额外守卫配置在r，那么$(r, v_1), (r, v_2), \cdots, (r, v_m)$的搜索方向或者是都从$r$到$v_1, v_2, \cdots, v_m$，或者是从$v_1, v_2, \cdots, v_m$全到$r$。如果有一个额外守卫配置在$r$，这样，每个$(r, v_i)$的搜索方向可以单独确定。

也就是，

$$C(T(r), \bar{r}) = \min\left\{\sum_{i=1}^{m} C(T(v_i), r, v_i), \sum_{i=1}^{m} C(T(v_i), v_i, r)\right\}$$

$$C(T(r), r) = wt(r) + \sum_{i=1}^{m} \min\{C(T(v_i), r, v_i), C(T(v_i), v_i, r)\}$$

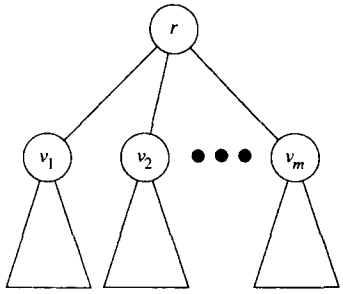


图7-38 规则2的解释

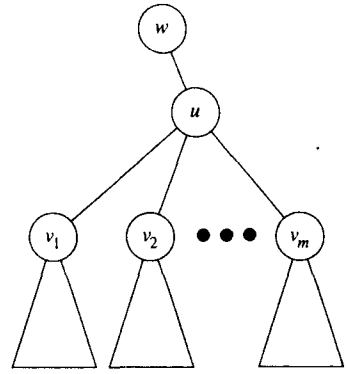


图7-39 规则3的解释

规则3：令u是一个内部结点，w是它的父结点并且v_1，v_2，…，v_m是它的后代结点（图7-39）。

如果没有额外守卫配置在u，那么(w, u)的搜索方向是从w到u（从u到w），所有(u, v_1)，(u, v_2)，…，(u, v_m)的搜索方向是从v_1，v_2，…，v_m到u（从u到v_1，v_2，…，v_m）。如果一个额外守卫配置在u，那么(u, v_j)的搜索方向可以单独确定。令$C(T(u), w, u, \bar{u})(C(T(u), w, u, u))$表示最优的单步搜索方案的代价，其中边$(u, w)$的搜索方向是从$w$到$u$，并且不需要额外的守卫配置在顶点$u$（需要额外的警卫配置在顶点$u$）。类似地，令$C(T(u), u, w, \bar{u})(C(T(u), u, w, u))$表示最优单步搜索方案的代价，其中边$(u, w)$的搜索方向是从$u$到$w$，并且不需要额外的守卫配置在顶点$u$（需要额外的警卫配置在顶点$u$）。这样，得到如下的公式：

$$C(T(u),w,u,\bar{u}) = wt(w) + \sum_{i=1}^{m} C(T(v_i),v_i,u)$$

$$C(T(u),w,u,u) = wt(w) + wt(u) + \sum_{i=1}^{m} \min\{C(T(v_i),v_i,u),C(T(v_i),u,v_i)\}$$

$$C(T(u),u,w,\bar{u}) = wt(u) + \sum_{i=1}^{m} C(T(v_i),u,v_i)$$

和

$$C(T(u),u,w,u) = 2wt(u) + \sum_{i=1}^{m} \min\{C(T(v_i),v_i,u),C(T(v_i),u,v_i)\}$$

这样，

$$C(T(u),w,u) = \min\{C(T(u),w,u,\bar{u}),C(T(u),w,u,u)\}$$
$$C(T(u),u,w) = \min\{C(T(u),u,w,\bar{u}),C(T(u),u,w,u)\}$$

最后，得到一个涉及边界条件的规则。

规则4：如果u是一个叶子结点，w是它的父结点，那么$C(T(u),w,u) = wt(w)$及$C(T(u),u,w) = wt(u)$。

通过从叶子结点逐渐向树根移动，动态规划策略解决带权单步图边的搜索问题。如果结点是叶子结点，那么使用规则4。如果结点是内部结点，但不是根结点，那么使用规则3。如果结点是根结点，那么先使用规则2，最后使用则1。

通过一个例子来阐释这个方案，参见图7-40所示的带权树。

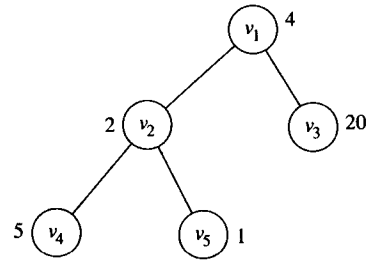


图7-40 一棵阐释动态规划策略的树

使用动态规划方法的过程如下：

步骤1. 选择叶子结点v_3，应用规则4，v_3的父结点是v_1。

$C(T(v_3), v_1, v_3) = wt(v_1) = 4$

$C(T(v_3), v_3, v_1) = wt(v_3) = 20$。

步骤2. 选择叶子结点v_4，应用规则4，v_4的父结点是v_2。

$C(T(v_4), v_2, v_4) = wt(v_2) = 2$

$C(T(v_4), v_4, v_2) = wt(v_4) = 5$。

步骤3. 选择叶子结点v_5，应用规则4，v_5的父结点是v_2。

$C(T(v_5), v_2, v_5) = wt(v_2) = 2$

$C(T(v_5), v_5, v_2) = wt(v_5) = 1$。

步骤4. 选择内部结点v_2，它不是根结点，应用规则3，v_2的父结点是v_1，v_2的后代结点是v_4和v_5。

$$C(T(v_2),v_1,v_2)=\min\{C(T(v_2),v_1,v_2,\overline{v_2}),\ C(T(v_2),v_1,v_2,v_2)\}$$

$$\begin{aligned}C(T(v_2),v_1,v_2,\overline{v_2})&=wt(v_1)+C(T(v_4),v_4,v_2)+C(T(v_5),v_5,v_2)\\&=4+5+1=10\end{aligned}$$

$$\begin{aligned}C(T(v_2),v_1,v_2,\overline{v_2})&=wt(v_1)+wt(v_2)+\min\{C(T(v_4),v_2,v_4),C(T(v_4),v_4,v_2)\}\\&\quad+\min\{C(T(v_5),v_2,v_5),C(T(v_5),v_5,v_2)\}\\&=4+2+\min\{2,5\}+\min\{2,1\}\\&=6+2+1=9\end{aligned}$$

$$C(T(v_2),v_1,v_2)=\min\{10,9\}=9$$

注意，上面的计算表明，如果$(v_1,\ v_2)$的搜索方向是从v_1到v_2，那么最优的搜索方案如图7-41所示，一个额外的守卫配置在v_2。

$$C(T(v_2),v_2,v_1)=\min\{C(T(v_2),v_2,v_1,\overline{v_2}),C(T(v_2),v_2,v_1,v_2)\}$$

$$\begin{aligned}C(T(v_2),v_2,v_1,\overline{v_2})&=wt(v_2)+C(T(v_4),v_2,v_4)+C(T(v_5),v_2,v_5)\\&=2+2+2=6\end{aligned}$$

$$\begin{aligned}C(T(v_2),v_2,v_1,v_2)&=wt(v_2)+wt(v_2)+\min\{C(T(v_4),v_2,v_4),C(T(v_4),v_4,v_2)\}\\&\quad+\min\{C(T(v_5),v_2,v_5),C(T(v_5),v_5,v_2)\}\\&=2+2+\min\{2,5\}+\min\{2,1\}\\&=4+2+1=7\end{aligned}$$

$$C(T(v_2),v_2,v_1)=\min\{6,7\}=6$$

上面的计算表明，如果$(v_1,\ v_2)$的搜索方向是从v_2到v_1，那么最优的搜索方案如图7-42所示。

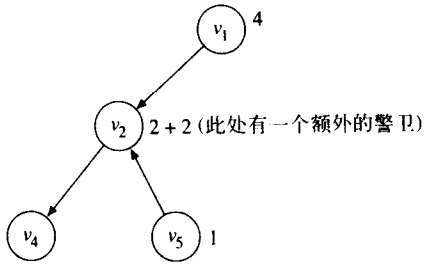


图7-41 一个包含v_2的单步搜索方案

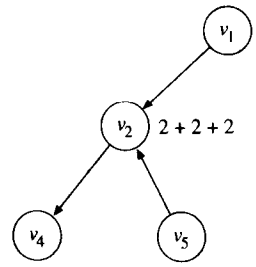


图7-42 另一个包含v_2的单步搜索方案

步骤5. 选择v_1，因为它是根结点，应用规则2。

$$\begin{aligned}C(T(v_1),\ \overline{v_1})&=\min\{C(T(v_2),v_1,v_2)+C(T(v_3),v_1,v_3),C(T(v_2),v_2,v_1)+C(T(v_3),v_3,v_1)\}\\&=\min\{9+4,6+20\}\\&=\min\{13,26\}\\&=13\end{aligned}$$

通过回溯了解到，如果没有额外的守卫配置在v_1，那么最优的搜索方案如图7-43所示。

$$\begin{aligned}C(T(v_1),v_1)&=wt(v_1)+\min\{C(T(v_2),v_1,v_2),C(T(v_2),v_2,v_1)\}\\&\quad+\min\{C(T(v_3),v_1,v_3),C(T(v_3),v_3,v_1)\}\\&=4+\min\{9,6\}+\min\{4,20\}\\&=4+6+4=14\end{aligned}$$

通过回溯了解到，如果一个额外的守卫配置在位置v_1，那么最优的搜索方案如图7-44所示。

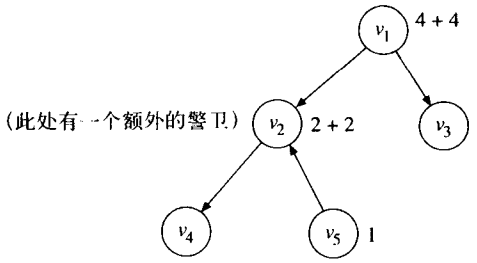


图7-43 一个包含v_1的单步搜索方案

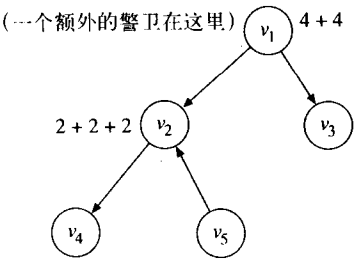


图7-44 另一个包含v_1的单步搜索方案

最后，应用规则1。

$$C(T(v_1)) = \min\{C(T(v_1), v_1), C(T(v_1), \overline{v_1})\}$$
$$= \min\{14, 13\} = 13$$

这表明没有额外守卫配置在v_1，那么最优的搜索方案如图7-43所示。

在每个点上有2个操作数：一个是计算最小搜索代价，另一个是确定边的搜索方向。因此，操作的总数是$O(n)$，其中n是树中的结点数。由于每个操作花费常数时间，所以这是一个线性算法。

7.9 用动态规划方法解决1螺旋多边形m守卫路由问题

m守卫路由问题（m-watchmen routes problem）定义如下：已知一个简单多边形及整数$m \geqslant 1$，要求为m名守卫找到路由，称为m守卫路由，使得多边形的每个点都至少被一名守卫从他路由的某个位置所见到。其目标是路由的长度之和最小。m守卫路由问题已证明是NP难的。

在本节中，将说明1螺旋多边形的m守卫路由问题可以用动态规划方法来解决。回忆第3章，将1螺旋多边形定义为分界线可以分解为凸链和反射链的简单多边形。通过遍历1螺旋多边形的边界，将反射链的起始点和终止点分别称为v_s和v_e。图7-45所示为1螺旋多边形的3守卫路由问题的解。

其基本思想是通过从反射链的某点内部平分线将1螺旋多边形分为两个部分，如图7-46所示。经过v_i的内部平分线与凸链相交，将1螺旋多边形分成两部分：两个都是1螺旋的，或者一个是1螺旋另一个是凸的。我们称包含v_s的子多边形为左子多边形，而另一个为相对于v_i的右子多边形。顶点v_i称为切割点（cut point）。为了方便后面的讨论，将反射链的起始点v_s（终止点v_e）的内部平分线定义为凸链的起始边（终止边）。

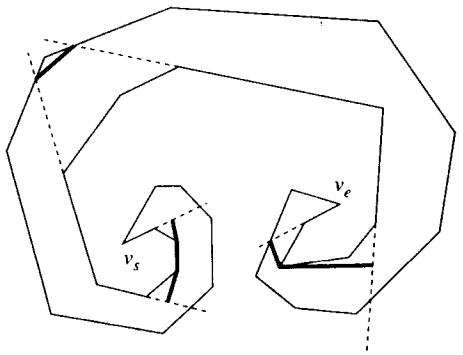


图7-45 1螺旋多边形的3守卫路由问题的解

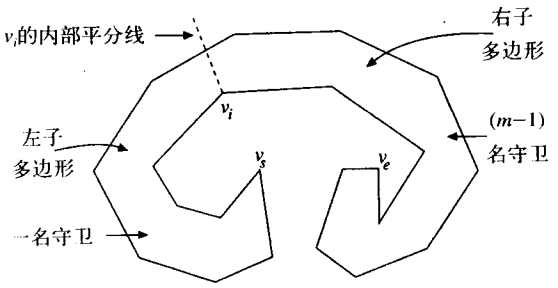


图7-46 求解m守卫路由问题的基本思想

将m名守卫分配到如下的两个子多边形中：一名守卫在左子多边形中，$(m-1)$名守卫在右子多边形中。假设知道怎样解决1守卫路由问题，那么$(m-1)$守卫路由问题可以递归地解决。就是说，可以再次将右子多边形划分成两部分，分配一名守卫到左边，$(m-2)$名守卫到右边。m守卫路由问题可以通过穷尽尝试全部可能的切割点和选择一个具有最短路由长度和的方法得到解决。下面说明为什么这种方法是动态规划方法。

参见图7-47，假设要解决一个3守卫路由问题。用动态规划方法解决这个3守卫路由问题的方法如下。令$SP(v_i, v_j)$表示边界在v_i和v_j内部平分线间的子多边形。

(1) 寻找下面的1守卫路由的解。

解1：对$SP(v_1, v_2)$的1守卫路由的解。

解2：对$SP(v_1, v_3)$的1守卫路由的解。

解3：对$SP(v_1, v_4)$的1守卫路由的解。

解4：对$SP(v_2, v_3)$的1守卫路由的解。

解5：对$SP(v_2, v_4)$的1守卫路由的解。

解6：对$SP(v_2, v_5)$的1守卫路由的解。

解7：对$SP(v_3, v_4)$的1守卫路由的解。

解8：对$SP(v_3, v_5)$的1守卫路由的解。

解9：对$SP(v_3, v_6)$的1守卫路由的解。

解10：对$SP(v_4, v_5)$的1守卫路由的解。

解11：对$SP(v_4, v_6)$的1守卫路由的解。

解12：对$SP(v_5, v_6)$的1守卫路由的解。

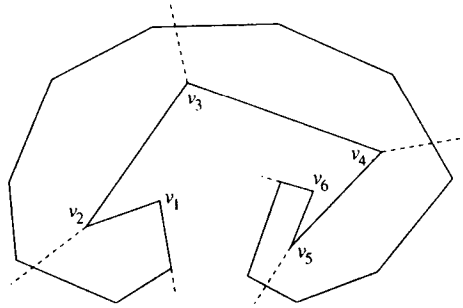


图7-47 在反射链中有六个点的1螺旋多边形

(2) 寻找下面的2守卫路由的解。

解13：对$SP(v_2, v_6)$的2守卫路由的解。

首先得到下面的解：

解13-1：组合解4和解9。

解13-2：组合解5和解11。

解13-3：组合解6和解12。

在解13-1、解13-2和解13-3中选择具有最短长度的解作为解13。

解14：$SP(v_3, v_6)$的2守卫路由的解。

首先得到下面的解：

解14-1：组合解7和解11。

解14-2：组合解8和解12。

在解14-1和解14-2中选择具有最短长度的解作为解14。

解15：$SP(v_4, v_6)$的2守卫路由的解。

通过组合解10和解12即可得到。

(3) 通过寻找下面的解求初始问题的3守卫路由的解。

解16-1：组合解1和解13。

解16-1：组合解2和解14。

解16-1：组合解3和解15。

在解16-1、解16-2或者解16-3中，选择任何一个都具有最短长度的。

讨论表明m守卫路由问题可以通过动态规划方法求解。注意到动态规划方法的本质是从解决基本问题开始，然后逐渐通过组合这些已求解的子问题解决更复杂的问题。首先解决所有相关的1守卫路由问题，然后是2守卫路由问题，等。

令$OWP_k(v_s, v_e)$表示子多边形$SP(v_i, v_j)$的最优k守卫路由的长度。最优m守卫路由$OWP_m(v_s, v_e)$可以通过下面的公式得到：

对于$2 \leqslant k \leqslant m-1$，$s+1 \leqslant i \leqslant e-1$，

$$OWR_m(v_s, v_e) = \min_{s+1 \leqslant i \leqslant e-1} \{OWR_1(v_s, v_i) + OWR_{m-1}(v_i, v_e)\}$$

$$OWR_k(v_i, v_e) = \min_{i+1 \leqslant j \leqslant e-1} \{OWR_1(v_i, v_j) + OWR_{k-1}(v_j, v_e)\}$$

1守卫路由问题在这里不再详述，因为我们的主要目的只是表明这个问题可以用动态规划方法求解。图7-48表明1守卫路由问题的一个典型的解，图7-49表明一种特殊的情况。

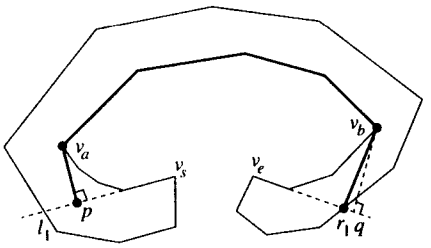
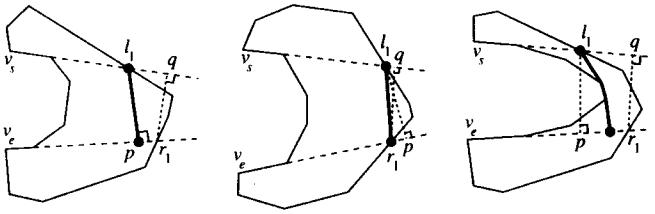


图7-48 在1螺旋多边形中一种典型的1守卫路由$p, v_a, C[v_a, v_b], v_b, r_1$



a) $\overline{l_1p}$或者$\overline{r_1q}$在多边形内部　b) p或q都在多边形的外部　c) $\overline{l_1p}$或者$\overline{r_1q}$之一与反射链相交

图7-49 在1螺旋多边形中1守卫路由问题的特殊情况

7.10 实验结果

为了说明动态规划的能力，我们在计算机上实现了用动态规划方法解决了最长公共子序列问题，也在同一台计算机上编程了直接的方法。表7-7描述了实验的结果，结果清楚地表明了动态规划的优势。

表7-7 实验结果

串 长	CPU毫秒时间	
	动态规划方法	穷举方法
4	<1	20
6	<2	172
8	<2	2 204
10	<10	32 952
12	<14	493 456

7.11 注释与参考

术语动态规划据说是由Bellman (1962)发明的。关于该主题许多作者都写了书籍：Nemhauser (1966)；Dreyfus and Law (1977)和Denardo (1982)。

在1962年，Bellman对于动态规划解决组合问题的应用做了回顾 (1962)。对于资源配置和调度问题的动态规划公式出现在文献Lawler and Moore (1969)；Sahni (1976)和Horowitz and Sahni (1978)中。旅行商问题的动态规划公式应归于文献Held and Karp (1962)和Bellman (1962)。对于最长公共子序列问题的动态规划应用由文献Hirschberg (1975)提出。解决0/1背包问题的动

态规划方法可在文献Nemhauser and Ullman (1969)和Horowitz and Sahni(1974)中找到。使用动态规划方法对最优二叉查找树的构造可在文献Gilbert and Moore (1959)，Knuth (1971)和Knuth (1973)中找到。对于树的带权完全支配集问题的动态规划的应用可在文献Yen and Lee (1990)中找到，解决单步检索问题可在文献Hsiao，Tang and Chang (1993)中找到。2序列比对算法可在文献Neddleman and Wunsch (1970)中找到，而对于RNA次级结构预测可在文献Waterman and Smith (1978)中找到。

在本书中没有提及的动态规划的其他重要应用，包括系列矩阵相乘：Godbole (1973)；Hu and Shing (1982)；Hu and Shing (1984)；所有点对的最短路径：Floyd (1962)；上下文无关语言的系统化分析：Younger (1967)，与/或串并联图：Simon and Lee (1971)，以及Viterbi解码：Viterbi (1967)和Omura (1969)。

动态规划可以与分支限界法一起使用，可参阅文献Morin and Marsten (1976)。它也用于解决一类特殊的划分问题，可参阅文献Garey and Johnson (1979)的4.2节，此观点也被Hsu and Nemhauser (1979)所使用。

7.12 进一步的阅读资料

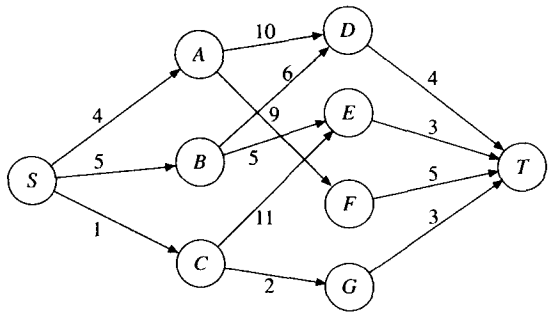
由于动态规划是如此优美，所以在计算机科学领域中，已经有很长时间在理论和实践中一直是研究主题。为了进一步地研究，推荐下面的论文：Akiyoshi and Takeaki (1997)；Auletta，Parente and Persiano (1996)；Baker (1994)；Bodlaender (1993)；Chen，Kuo and Sheu (1988)；Chung，Makedon，Sudborough and Turner (1985)；Even，Itai and Shamir (1976)；Farber and Keil (1985)；Fonlupt and Nachef (1993)；Gotlieb (1981)；Gotlieb and Wood (1981)；Hirschberg and Larmore(1987)；Horowitz and Sahni (1974)；Huo and Chang (1994)；Johnson and Burrus (1983)；Kantabutra (1994)；Kao and Queyranne (1982)；Kilpelainen and Mannila (1995)；Kryazhimskiy and Savinov (1995)；Liang (1994)；Meijer and Rappaport (1992)；Morin and Marsten (1976)；Ozden(1988)；Park (1991)；Peng，Stephens and Yesha (1993)；Perl (1984)；Pevzner (1992)；Rosenthal (1982)；Sekhon (1982)；Tidball and Atman (1996)；Tsai and Hsu (1993)；Tsai and Lee (1997)；Yannakakis (1985)；Yen and Lee (1990)；Yen and Lee (1994) and Yen and Tang (1995)。

以下是有趣的新成果的列表：Aho，Ganapathi and Tjang (1989)；Akutsu (1996)；Alpert and Kahng (1995)；Amini，Weymouth and Jain (1990)；Baker and Giancarlo (2002)；Bandelloni，Tucci and Rinaldi(1994)；Barbu (1991)；Brown and Whitney (1994)；Charalambous (1997)；Chen，Chern and Jang (1990)；Cormen (1999)；Culberson and Rudnicki (1989)；Delcoigne and Hansen (1975)；Eppstein，Galil，Giancarlo and Italiano (1990)；Eppstein，Galil，Giancarlo and Italiano (1992(a))；Eppstein，Galil，Giancarlo and Italiano (1992(b))；Erdmann (1993)；Farach and Thorup (1997)；Fischel-Ghodsian，Mathiowitz and Smith (1990)；Geiger，Gupta，Costa and Vlontzos (1995)；Gelfand and Roytberg (1993)；Galil and Park (1992)；Hanson (1991)；Haussmann and Suo (1995)；Hein (1989)；Hell，Shamir and Sharan (2001)；Hirosawa，Hoshida，Ishikawa and Toya (1993)；Holmes and Durbin (1998)；Huang，Liu and Viswanathan (1994)；Huang and Waterman (1992)；Ibaraki and Nakamura (1994)；Karoui and Quenez (1995)；Klein (1995)；Kostreva and Wiecek (1993)；Lewandowski，Condon and Bach (1996)；Liao and Shoemaker (1991)；Lin，Fan and Lee (1993)；Lin，Chen，Jiang and Wen (2002)；Littman，Cassandra and Kaelbling (1996)；Martin and Talley (1995)；Merlet and Zerubia (1996)；Miller and Teng (1999)；Mohamed and Gader (1996)；Moor (1994)；Motta and Rampazzo (1996)；

Myoupo (1992)；Ney (1984)；Ney (1991)；Nuyts，Suetens，Oosterlinck，Roo and Mortelmans (1991)；Ohta and Kanade (1985)；Ouyang and Shahidehpour (1992)；Pearson and Miller (1992)；Rivas and Eddy (1999)；Sakoe and Chiba (1978)；Schmidt (1998)；Snyder and Stormo (1993)；Sutton (1990)；Tataru (1992)；Tatman and Shachter (1990)；Tatsuya (2000)；Vintsyuk (1968)；Von Haeselerm，Blum，Simpson，Strum and Waterman (1992)；Waterman and Smith (1986)；Wu (1996)；Xu (1990) and Zuker (1989)。

习题

7.1 见下图。用动态规划方法找出从S到T的最短路由。



7.2 对于显示在图7-1中的图，使用分支限界方法解决同样的问题。对于此问题，哪个方法（动态规划与分支限界方法对比）更好？为什么？

7.3 对于显示在图7-13中的图，用分支限界方法解决旅行商问题，并将它与动态规划方法进行对比。

7.4 对于下面的表，有三个项目和四个资源，找出一个最优的资源分配从而得到最大总利润。

项目 \ 资源	1	2	3	4
1	3	7	10	12
2	1	2	6	9
3	2	4	8	9

7.5 使用动态规划解决下面的线性规划问题。

最大化 $x_0 = 8x_1 + 7x_2$

应满足 $2x_1 + x_2 \leqslant 8$

$5x_1 + 2x_2 \leqslant 15$

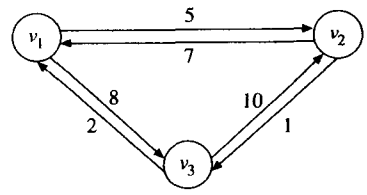
其中x_1和x_2是非负整数。

7.6 找出$S_1 = a\ a\ b\ c\ d\ a\ e\ f$和$S_2 = b\ e\ a\ d\ f$的最长公共子序列。

7.7 一般，划分问题是NP完全的。然而，在一些限制下，因为多项式问题可以通过动态规划解决，所以这是一类特殊的划分问题。参阅文献Garey and Johnson(1979)的4.2节。

7.8 找出a_1，a_2，…，a_6的一个最优二叉树，如果标识符的概率按顺序分别是0.2，0.1，0.15，0.2，0.3，0.05，其他标识符的概率为0。

7.9 参见右面的图：求解图中所有点对的最短路径问题。所有点对的最短路径问题是确定每对顶点间的最短路径问题。参考文献Horowitz and Sahni(1978)的5.3节，或者Brassard and Bratley (1988)的5.4节。



7.10 令f是x的实函数，并且$y = (y_1, y_2, \cdots, y_k)$。如果$f$是可分

的，而且该函数对于第二个参数是单调非递减的，我们说f可分解为f_1和f_2($f(x, y) = f_1(x), f_2(y)$)。证明，如果f是$f(x, y) = f_1(x), f_2(y)$可分解的，那么

$$\underset{(x,y)}{Opt}\{f(x, y)\} = Opt\{f_1(x, Opt\{f_2(y)\})\} \quad (Opt = \min\text{或}\max)$$

（参考[Minoux 1986]的9.2节。）

7.11 佛洛伊德（Floyd）算法可以在许多教科书中找到，该算法可以在一个带权图上找出所有点对的最短路径。举出一个例子解释该算法。

7.12 编写一个动态规划算法，求解最长的渐增子序列问题。

7.13 已知在字母表集合Σ上的两个序列S_1和S_2，以及一个得分函数f：$\Sigma \times \Sigma \rightarrow \Re$，局部比对问题是找出$S_1$的子序列$S_1'$和$S_2$的子序列$S_2'$，使得在$S_1$和$S_2$的所有可能的子序列中，通过$S_1'$与$S_2'$的比对得分最高。使用动态规划策略设计一个$O(mn)$时间复杂度或更好的算法解决该问题，其中$n$和$m$分别表示$S_1$和$S_2$的长度。

第 8 章　NP完全性理论

NP完全性理论（theory of NP-completeness ）也许是计算机科学领域最有趣的主题之一，该领域主要的研究者——多伦多大学的S.A.库克教授因其在该领域内的杰出贡献而获得了图灵奖。毫无疑问，在计算机科学领域众多有趣的研究成果中，NP完全性理论是最激动人心、同时也是令人迷惑的理论之一。现在称为库克定理（Cook's theorem）的最重要定理，也许是被最广泛引用的定理。

本书不仅介绍库克定理的应用，而且尽力解释库克定理真正的含义。

8.1　关于NP完全性理论的非形式化讨论

NP完全性理论是重要的，因为它确认了一大类难问题。这里定义的难问题是指其下界似乎存在指数函数级的问题。换句话说，NP完全性理论明确了一大类问题，看上去找不到能解决它们的任何多项式时间算法。

概略地说，我们认为NP完全性理论首先指出许多问题可称为NP问题（non-deterministic polynomial problem，非确定型多项式）。（NP的形式定义将在8.4节给出。）并非所有的NP问题都是难的，许多是容易的。例如，搜索问题是NP问题，它可以用多项式时间复杂度的算法解决。另一个例子是最小生成树问题，该问题也可以由多项式时间算法解决。我们称这些问题为P问题（polynomial problem，多项式问题）。

由于在NP问题集中包含许多P问题，可以画一幅图描述它们之间的关系，如图8-1所示。

此外，该图表明还有另外一大类问题，它们是NP完全问题（NP-complete problem），包含在NP问题集内，如图8-2所示。

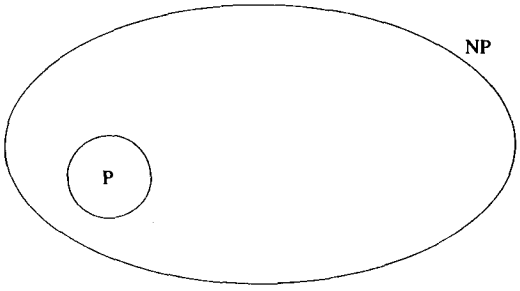


图8-1　NP问题集

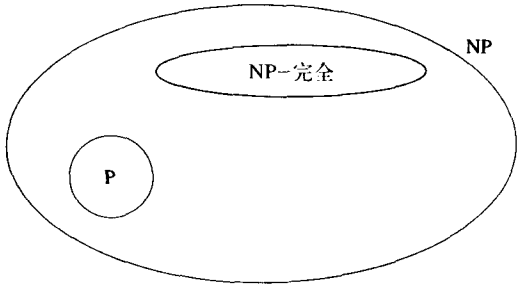


图8-2　NP问题包含P问题和NP完全问题

目前，已知的NP完全问题集是非常大的，并且一直在增加。这包括许多有名的问题，比如可满足性问题（satisfiability problem）、旅行商问题（traveling salesperson problem），以及装箱问题（bin packing problem）。所有这些问题都有一个共同的特性：到目前为止，没有一个NP完全问题在最坏情况下可被任一个多项式算法解决。换句话说，直到现在，在最坏情况下，解决任何NP完全问题的最好算法是有指数时间复杂度。这里特别强调NP完全性理论总是讨论最坏情况。

NP完全问题已经能够通过具有多项式平均时间复杂度的算法来解决。在本章余下的内容

中，除非特别说明，在任何时间讨论时间复杂度都是指最坏时间复杂度。

我们并没有足够的兴趣去明确一个问题集，其到目前为止不能通过任何多项式算法来解决。根据对NP完全性的定义，下面的结论是正确的：

如果任何NP完全问题可在多项式时间内解决，那么所有的NP问题可在多项式时间内解决。或者，如果任何NP完全问题能在多项式时间内解决，那么NP＝P。

因此，NP完全性理论表明每个NP完全问题就像一个重要的柱子。如果它倒下，那么整个建筑就会倒塌。或者用另外一种比喻方式，NP完全问题好似一位重要的将军，如果他向敌人投降，那么整个军队也会投降。

因为所有的NP问题都能由多项式算法解决是非常不可能的，所以任何NP完全问题能由任何多项式算法解决更是不可能的。

这里需要强调，NP完全性理论并未断言NP完全问题永不能被多项式算法解决，而只是说任何NP完全问题在多项式步数内解决是相当不可能的。这对我们尽力找出解决NP完全问题的多项式算法多少有些阻碍。

8.2 判定问题

我们将本书所考虑过的大多数问题划分为两类：优化问题（optimization problems）和判定问题（decision problems）。

考虑旅行商问题。该问题定义如下：已知一个点集合，找出从任何点v_0开始的最短回路。旅行商问题显然是一个优化问题。

*判定问题是其简单回答为“yes”或“no”的问题。*对于旅行商问题，有一个相对应的判定问题，定义如下：已知一个点集合，是否存在从任何点v_0开始的回路，其总长度小于已知的常数c？

需要注意旅行商问题比旅行商判定问题（traveling salesperson person problem）困难得多。如果能够解决旅行商问题，那么我们会知道最短回路等于某个值，比如是a。如果$a<c$，那么对旅行商判定问题的回答就是“yes”，否则就是“no”。因此可以说，*如果能够解决旅行商问题，就能够解决旅行商判定问题，但反之不行。所以，我们得出结论：旅行商问题比旅行商判定问题困难得多。*

再看另一个例子，在第5章中已经介绍过的0/1背包问题，定义如下：

已知M，W_i和P_i，$W_i>0$，$P_i>0$，$1\leqslant i\leqslant n$，$M>0$，在$\sum_{i=1}^{n}W_ix_i\leqslant M$条件下，找出$x_i$，$x_i=1$或0，使得$\sum_{i=1}^{n}P_ix_i$最大。

0/1背包问题显然也是一个优化问题，它也有一个相应的判定问题，定义如下：

已知M，R，W_i和P_i，$M>0$，$R>0$，$W_i>0$，$P_i>0$，$1\leqslant i\leqslant n$，确定是否存在x_i，$x_i=1$或0，使得$\sum_{i=1}^{n}P_ix_i\geqslant R$和$\sum_{i=1}^{n}W_ix_i\leqslant M$。

这很容易证明0/1背包问题比0/1背包判定问题（0/1 knapsack decision problem）困难得多。

*一般地，优化问题比它们相应的判定问题难以解决。*所以，如果只对问题是否可被多项式算法解决的讨论关心，那么可以只考虑判定问题。如果旅行商判定问题不能被多项式算法解决，那么可以推断旅行商问题也不能被多项式算法解决。*在讨论NP问题时，将只讨论判定问题。*

在下一节中，将讨论可满足性问题，它是最著名的判定问题之一。

8.3 可满足性问题

因为可满足性问题（satisfiability problem）是找出的第一个NP完全问题，所以它是很重要的。考虑下面的逻辑公式：

$$
\begin{aligned}
& x_1 \vee x_2 \vee x_3 \\
\& \quad & -x_1 \\
\& \quad & -x_2
\end{aligned}
$$

下面的赋值会使公式为真。

$x_1 \leftarrow F$

$x_2 \leftarrow F$

$x_3 \leftarrow T$

接下来将使用记号$(-x_1, -x_2, x_3)$表示$\{x_1 \leftarrow F, x_2 \leftarrow F, x_3 \leftarrow T\}$。如果一个赋值使得公式为真，那么就说这个赋值满足该公式；否则，它不满足该公式。

如果至少有一个赋值能满足公式，那么说此公式是可满足的；否则，该公式是不可满足的。

典型的不可满足公式是

$$
\begin{aligned}
& x_1 \\
\& \quad & -x_1
\end{aligned}
$$

另一个不可满足公式是

$$
\begin{aligned}
& x_1 \vee x_2 \\
\& \quad & x_1 \vee -x_2 \\
\& \quad & -x_1 \vee x_2 \\
\& \quad & -x_1 \vee -x_2
\end{aligned}
$$

可满足性问题定义如下：已知一个布尔公式（Boolean formula），确定该公式是否可满足。

在本节余下的内容中，将讨论一些解决可满足性问题的方法，首先需要定义一些术语。

定义 文字（literal）是x_i或$-x_i$，其中x_i是一个布尔变量。

定义 子句（clause）是文字的析取（disjunction）。应该明白没有子句同时包含文字及其文字的非。

定义 公式（formula）是$c_1 \& c_2 \& \cdots \& c_m$形式的合取范式（conjunctive normal form），其中每个c_i（$1 \leqslant i \leqslant m$）是子句。

众所周知每个布尔公式能转换成合取范式形式。所以，假定所有的公式都已经是合取范式形式了。

定义 公式G是公式F的逻辑结论，当且仅当F为真时，G也为真。换句话说，每个赋值满足F，也满足G。

例如，

$$
\begin{aligned}
& -x_1 \vee x_2 && (1) \\
\& \quad & x_1 && (2) \\
\& \quad & x_3 && (3)
\end{aligned}
$$

是合取范式形式的公式。满足上面公式仅有的赋值是(x_1, x_2, x_3)。读者可以容易地推断x_2是上面公式的逻辑结论。已知两个子句

c_1：$L_1 \vee L_2 \vee \cdots \vee L_j$

和 c_2：$-L_1 \vee L_2' \vee \cdots \vee L_k'$，
可以推出一个子句

$$L_2 \vee \cdots \vee L_j \vee L_2' \vee \cdots \vee L_k'$$

如果子句

$$L_2 \vee \cdots \vee L_j \vee L_2' \vee \cdots \vee L_k'$$

没有包含任何互补的一对文字，那么可作为子句c_1&c_2的逻辑结论。

例如，考虑下面的子句：

$$c_1: -x_1 \vee x_2$$
$$c_2: x_1 \vee x_3$$

那么

$$c_3: x_2 \vee x_3$$

是子句c_1&c_2的逻辑结论。

上面的推理规则称为**消解原理**（resolution principle），对c_1和c_2应用消解原理产生的子句c_3称为c_1和c_2的**消解式**（resolvent）。

再看一个例子：

c_1：$-x_1 \vee -x_2 \vee x_3$

c_2：$x_1 \vee x_4$。

那么

c_3：$-x_2 \vee x_3 \vee x_4$

是c_1和c_2的消解式。当然，它也是c_1&c_2的逻辑结论。

考虑下面的两个子句：

c_1：x_1

c_2：$-x_1$

那么消解式是个特殊的子句，因为它不包含文字，表示为

$c_3 = \square$

它是个空子句。

如果能从子句集中推导出空子句，那么这个子句集一定是不可满足的。考虑下面的子句集：

$x_1 \vee x_2$ (1)

$x_1 \vee -x_2$ (2)

$-x_1 \vee x_2$ (3)

$-x_1 \vee -x_2$ (4)

可以按如下方式推导出空子句：

(1) &(2) x_1 (5)

(3) &(4) $-x_1$ (6)

(5) &(6) $\square$ (7)

由于(7)是一个空子句，所以可以推断(1)&(2)&(3)&(4)是不可满足的。

已知一个子句集，我们可以重复地应用消解原理推导新子句。新子句可以添加到原来的子句集中，消解原理可以再次应用其中。这个过程直到产生空子句或没有新子句产生为止。如果

推导出空子句，那么这个子句集是不可满足的。如果当消解过程终止而没有新子句生成，那么这个子句集是可满足的。

看一个可满足的子句集

$$-x_1 \vee -x_2 \vee x_3 \quad (1)$$
$$x_1 \quad (2)$$
$$x_2 \quad (3)$$
$$(1)\&(2)\quad -x_2 \vee x_3 \quad (4)$$
$$(4)\&(3)\quad x_3 \quad (5)$$
$$(1)\&(3)\quad -x_1 \vee x_3 \quad (6)$$

注意到从子句(1)到(6)不再有新子句生成，可以推断这个子句集是可满足的。

如果通过增加一个$-x_3$来修改上面的子句集，那么得到一个不可满足的子句集：

$$-x_1 \vee -x_2 \vee x_3 \quad (1)$$
$$x_1 \quad (2)$$
$$x_2 \quad (3)$$
$$-x_3 \quad (4)$$
$$(1)\&(2)\quad -x_2 \vee x_3 \quad (5)$$
$$(5)\&(4)\quad -x_2 \quad (6)$$
$$(6)\&(3)\quad \square \quad (7)$$

因为可以推导出空子句，所以现在可以建立是不可满足的属性。

对于消解原理更多理论性的讨论可参考有关机器定理证明（mechanical theorem proving）的书籍。

在上面的讨论中，说明可满足性问题可以看作是演绎问题。换句话说，就是坚持不懈地寻找矛盾。如果推导出矛盾，就可以断定子句集是不可满足的，否则是可满足的。我们采用的方法似乎与找出满足公式的赋值无关。事实上，立刻可以证明演绎（或推导）的方法等价于找出赋值的方法。也就是，对空子句的演绎的确等价于找出满足任何子句赋值的失败。相反地，推导不出空子句等价于找出至少一个满足所有子句的赋值。

从一个最简单的例子开始：

$$x_1 \quad (1)$$
$$-x_1 \quad (2)$$

由于上面的子句集只包含一个变量x_1，那么开始构造一棵语义树，如图8-3所示。

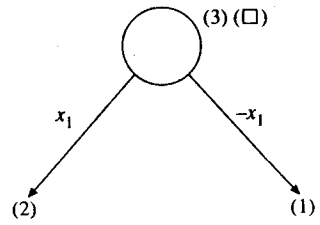


图8-3 一棵语义树

左分支意味着赋值包含x_1（意指$x_1 \leftarrow T$），而右分支意味着赋值包含$-x_1$（意指$x_1 \leftarrow F$）。注意到左分支赋值x_1使得子句(2)为假。所以用子句(2)终止左分支。类似地，用子句(1)终止右分支。图8-3表明子句(1)必须包含x_1，而子句(2)必须包含$-x_1$。对上面子句应用消解原理推导出一个新子句，就是空子句，表示如下：

$$(1)\&(2)\quad \square \quad (3)$$

可将子句(3)放在父结点旁边，如图8-3所示。

考虑下面的子句集：

$$-x_1 \vee -x_2 \vee x_3 \quad (1)$$

$x_1 \vee -x_2$ (2)

x_2 (3)

$-x_3$ (4)

可以构造如图8-4所示的语义树。在图8-4中，从树根到终端结点的每条路径代表了一类赋值。例如，每个赋值必须包含x_2或$-x_2$。第一个右分支标记为$-x_2$，表示所有包含$-x_2$的赋值（有四个这样的赋值）。由于子句(3)只含有x_2，这使每个包含$-x_2$的赋值为假。因此，第一个右分支被子句(3)终止。

考虑包含x_2、$-x_3$和x_1的路径，这个赋值使子句(1)为假。类似地，包含x_2、$-x_3$和$-x_1$的路径表示使子句(2)为假的赋值。

考虑标记为(1)和(2)的终端结点。由于通向它们的分支分别标记为x_1和$-x_1$，子句(1)必定包含$-x_1$，而子句(2)必定包含x_1。对子句(1)和(2)应用消解原理，可以推导出子句$-x_2 \vee x_3$。这个子句可以标记为子句(5)，并且邻接其父结点上，如图8-4所示。使用同样的推理，可以对子句(5)和(4)应用消解原理，从而得到子句(6)。而子句(6)和子句(3)是相互矛盾的，由此推导出空子句。所有这些都表示在图8-4中，整个推导过程详细地描述如下：

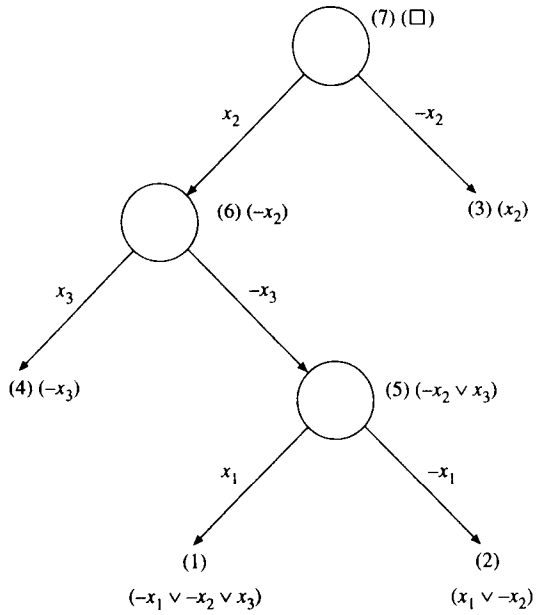


图8-4 一棵语义树

(1)&(2) $-x_2 \vee x_3$ (5)

(4)&(5) $-x_2$ (6)

(6)&(3) □ (7)

总之，已知表示布尔公式的子句集，可以根据下面的规则构造语义树：

(1) 从语义树的每个内部结点有两个分支离开它。一个用x_i标记，另一个用$-x_i$标记，这里x_i是出现在子句集中的一个变量。

(2) 一个结点被终止，只要对应于出现在从树根到该结点路径中的文字赋值使子句集中的子句(j)为假，就标记此结点为终端结点，并将子句(j)连接到结点上。

(3) 在语义树中没有路径包含互补对，使得每个赋值都相容。

显然，每棵语义树都是有限的。如果每个端结点连接一个子句，那么不存在满足所有子句的赋值，这意味着该子句集是不可满足的；否则，至少存在一个满足所有子句的赋值，且该子句集是可满足的。

考虑下面的子句集：

$-x_1 \vee -x_2 \vee x_3$ (1)

$x_1 \vee x_4$ (2)

$x_2 \vee -x_1$ (3)

构造语义树如图8-5所示。

从上面的语义树推断该子句集是可满足的，下面的赋值都满足公式：

(x_1，x_2，x_3，x_4)

$(x_1, x_2, x_3, -x_4)$

$(-x_1, -x_2, x_3, x_4)$

$(-x_1, -x_2, -x_3, x_4)$

$(-x_1, x_2, x_3, x_4)$

和 $(-x_1, x_2, -x_3, x_4)$

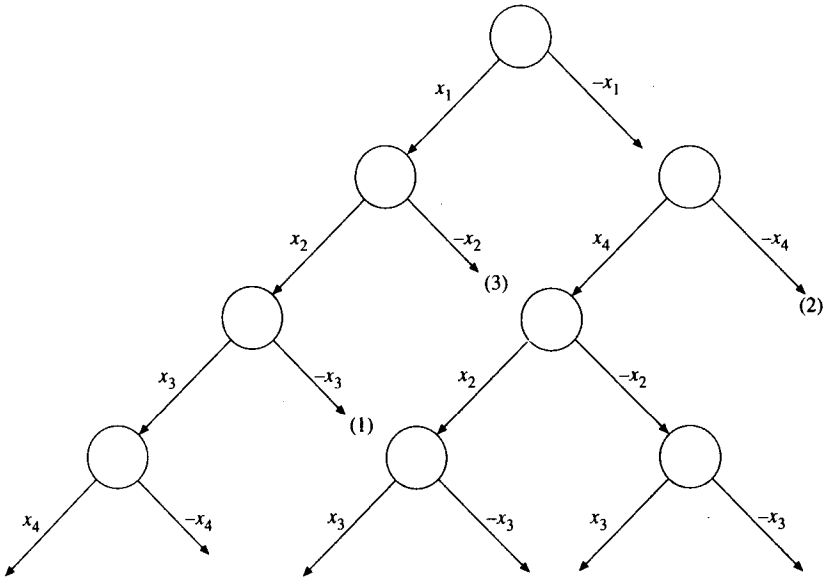


图8-5 一棵语义树

如果子句集是不可满足的，那么使用消解原理的每棵语义树都相对应于空子句的推导。从语义树中提取的推导如下：

(1) 考虑其后代是终端结点的内部结点，假定连接到两个终端结点的子句分别是c_i和c_j。对两个子句应用消解原理，将消解式连接到它们的父结点上。同时删除后代结点，那么原来的内部结点成为终端结点。

(2) 重复上面的过程，直到语义树成为空树，那么空子句就推导出来。

通过另一个例子解释上面的思想。考虑下面的子句集：

$-x_1 \vee -x_2 \vee x_3$ (1)

$x_1 \vee x_3$ (2)

x_2 (3)

$-x_3$ (4)

构造的语义树如图8-6所示。

图8-6的语义树可逐步地消减成图8-7。

推导过程如下：

(1)&(2) $-x_2 \vee x_3$ (5)

(5)&(4) $-x_2$ (6)

(6)&(3) $\square$ (7)

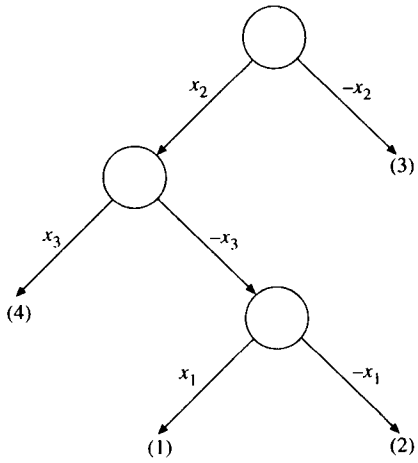


图8-6 一棵语义树

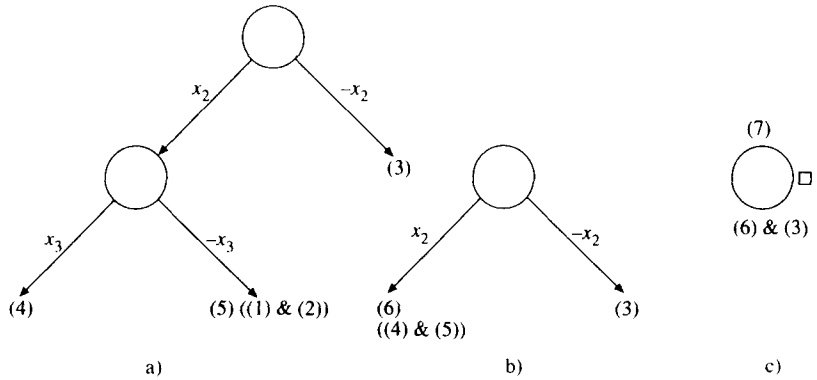


图8-7 图8-6中语义树的消减

我们展示了所使用的推导方法的确找到了满足所有子句的赋值。如果有n个变量，那么会有2^n个可能的赋值。到目前为止，在最坏情况下，对于可用的最好算法，在得出任何结论前必须检查指数量级的可能赋值。

可满足性问题在多项式时间内解决有可能吗？NP完全性理论并没有排除这种可能性，但做了下面的断言：*如果可满足性问题可在多项式步数内解决，那么所有的NP问题也可在多项式步数内解决。*

到目前仍没有定义NP问题，这将在下一节中讨论。

8.4 NP问题

符号NP表示非确定性多项式。我们首先定义非确定性算法（nondeterministic algorithm）如下：*非确定性算法是由猜测和验证构成的算法。更进一步，假定非确定性算法总是做出正确的猜测。*

例如，给定与具有特定布尔公式对应的可满足性问题，非确定性算法首先猜测一个赋值，接着验证这个赋值是否满足该公式。需要注意的概念是正确解总是通过猜测得到的。换句话说，如果公式是可满足的，那么非确定性算法总是正确地猜测，并获得满足该公式的赋值。

考虑旅行商判定问题，非确定性算法总是猜测一个回路，并验证回路是否比常数c短。

读者可能会被非确定性算法的概念所激怒，因为实际上不可能有这样的算法。那么，怎么总能做出正确地猜测呢？

实际上，非确定性算法的确并不存在，也决不会存在。只是因为它将有助于我们定义一类问题，称为NP问题，非确定性算法才有用。

如果非确定性算法的验证阶段是多项式时间复杂度的，那么这个非确定性算法称为非确定性多项式算法（nondeterministic polynomial algorithm）。*如果判定问题能被非确定性多项式算法解决，那么该问题称为非确定性多项式*（NP）*问题*（nondeterministic polynomial problem）。

从上面的定义，立即可以推断出*每个能在多项式时间（当然根据确定性算法）解决的问题一定是非确定性多项式问题。*典型的例子是搜索、合并、排序以及最小生成树等问题。这里提醒读者我们在讨论判定问题。搜索是判定问题，而排序显然不是，但总能够产生来自排序问题的判定问题。最初的排序问题是将a_1，a_2，…，a_n排序成递增或递减序列。可构造如下的判定问题：已知a_1，a_2，…，a_n和C，确定是否存在一个a_i的排列$(a_1', a_2', \cdots, a_n')$，使得$|a_2'-a_1'| + |a_3'-a_2'| + \cdots + |a_n'-a_{n-1}'| < C$。所有能在多项式时间解决的问题都称为$P$问题。

可满足性问题和旅行商判定问题都是NP问题，因为这两个问题的验证阶段都是多项式时

间复杂度的。实际上，人们能想到的许多可解问题都是NP问题。

一个有名的不是NP问题的判定问题是停机问题（halting problem）。停机问题定义如下：已知带有任意输入数据的任意程序，此程序能终止吗？另一个问题是一阶谓词演算可满足性问题（first-order predicate calculus problem）。这两个问题都是所谓的不可判定问题（undecidable problems）。

不可判定问题不能够靠猜测和验证解决。尽管它们是判定问题，但也不能穷尽验证整个解空间来解决。读者应该注意在布尔逻辑（Boolean logic）（或者叫做命题逻辑（propositional logic））中，一个指派是用n元组刻画的。但是对于一阶谓词演算，指派不是有界的，可能是无限长的。这就是说为什么一阶谓词演算不是NP问题。这足以提醒读者不可判定性问题甚至比NP问题还难。

可以更清晰地说，对于可满足性问题和旅行商判定问题，解的数量是有限的。对于可满足性问题可能有2^n个指派，对于旅行商判定问题可能有$(n-1)!$条回路。所以，尽管这些问题很难，但它们至少有某些上限。例如，对于可满足性问题，至少可以用$O(2^n)$时间复杂度的算法解决。

可是，不可判定问题没有上限，可以证明这样的上界不存在。直观地说，让程序运行一百万年，仍然不能得到任何结论，因为程序在下一步停机是完全可能的。相似地，一阶谓词演算判定问题也面临同样的情况。假设程序在运行了很长时间之后，仍未得到空子句。但是，要产生的下一子句是空子句是完全可能的。

8.5 库克定理

在本节介绍库克定理。将只给一个非形式化的证明，因为形式化的证明非常复杂。库克定理陈述如下。

库克定理 当且仅当可满足性问题是P问题，那么NP = P。

上面定理的证明由两部分组成。第一部分是“如果NP = P，那么可满足性问题是P问题。”此部分是显然的，因为可满足性问题是NP问题。第二部分是“如果可满足性问题是P问题，那么NP = P。”这是库克定理的关键部分，本节余下的内容将详细阐述这部分。

现在先对库克定理的本质进行解释。假如有一个很难解决的NP问题A，我们不直接解决该问题，而是生成另一个新问题A'，通过求解A'得到A的解。需要注意每个问题都是判定问题，采用的方法如下：

(1) 由于问题A是NP问题，那么一定存在解决该问题的NP算法B。NP算法是非确定性多项式算法，它实质上是不可能存在的，所以并不能使用它，但是，正如将要看到的步骤中我们仍然可以从概念上使用算法B。

(2) 构造对应于B的布尔公式C，使得C是可满足的当且仅当非确定性算法B成功终止，并返回“yes”。如果C是不可满足的，那么算法B将不成功终止，并返回“no”。

需要提醒一点的是，当提及一个问题时，是指一个问题的实例（instance）。也就是问题有特定的输入，否则不能说算法终止。

我们将推迟如何构造公式C的讨论，这部分是库克定理的关键部分。

(3) 在构造公式C后，暂时忘记原来的问题A和非确定性问题B。尽力推断公式C是否可满足。如果是可满足的，那么对问题A的回答是“yes”；否则，回答是“no”。这么做是由于在步骤(2)中所陈述的公式C的属性，也就是，C是可满足的当且仅当B成功终止。

一切都是美好的。上面的方法似乎表明只需关注可满足性问题。例如，我们不必知道如何解决旅行商判定问题，只要知道如何确定相对应旅行商问题的布尔公式是否可满足。这是一个严肃的大问题。如果可满足性问题很难解决，那么原来的旅行商问题也很难解决。这是库克定

理的核心，它表明，如果可满足性问题能在多项式步数内解决，那么每个NP问题也可在多项式步数内解决，本质上是因为上面的方法。

读者能够理解上面的方法是有效的，当且仅当总能构造来自与非确定性算法B的布尔公式C，使得C是可满足的，当且仅当B成功终止。现在所能做的就是举例说明。

例8-1 搜索问题的布尔公式（情况1）

考虑搜索问题。已知n个数的集合$S = \{x(1), x(2), \cdots, x(n)\}$，要确定在集合$S$中是否存在一个数，比如说7。为了使讨论简单，假定$n = 2$，$x(1) = 7$，$x(2) \neq 7$。

非确定性算法表示如下：

$i = choice(1, 2)$

if $x(i) = 7$ then *SUCCESS*

else *FAILURE*.

相对应于上面非确定性算法的布尔公式如下：

$$
\begin{array}{llll}
 & i=1 & \vee & i=2 \\
\& & i=1 & \rightarrow & i\neq 2 \\
\& & i=2 & \rightarrow & i\neq 1 \\
\& & x(1)=7 \ \& \ i=1 & \rightarrow & SUCCESS \\
\& & x(2)=7 \ \& \ i=2 & \rightarrow & SUCCESS \\
\& & x(1)\neq 7 \ \& \ i=1 & \rightarrow & FAILURE \\
\& & x(2)\neq 7 \ \& \ i=2 & \rightarrow & FAILURE \\
\& & FAILURE & \rightarrow & -SUCCESS \\
\& & SUCCESS & & \text{（保证成功终止）} \\
\& & x(1)=7 & & \text{（输入数据）} \\
\& & x(2)\neq 7 & & \text{（输入数据）}
\end{array}
$$

为便于讨论，将上面的公式转换成合取范式：

$$
\begin{array}{lllll}
i=1 & \vee & i=2 & & & (1) \\
i\neq 1 & \vee & i\neq 2 & & & (2) \\
x(1)\neq 7 & \vee & i\neq 1 & \vee & SUCCESS & (3) \\
x(2)\neq 7 & \vee & i\neq 2 & \vee & SUCCESS & (4) \\
x(1)=7 & \vee & i\neq 1 & \vee & FAILURE & (5) \\
x(2)=7 & \vee & i\neq 2 & \vee & FAILURE & (6) \\
-FAILURE & \vee & -SUCCESS & & & (7) \\
SUCCESS & & & & & (8) \\
x(1)=7 & & & & & (9) \\
x(2)\neq 7 & & & & & (10)
\end{array}
$$

上面的子句集是通过“&”连接的，暂且忽略。由于下面的赋值满足所有的子句，所以它们是满足的：

$i = 1$	满足	(1)
$i \neq 2$	满足	(2)，(4)和(6)
SUCCESS	满足	(3)，(4)和(8)
$-FAILURE$	满足	(7)
$x(1) = 7$	满足	(5)和(9)
$x(2) \neq 7$	满足	(4)和(10)

正如所看到的，现在所有的子句都可满足。上面满足所有子句的赋值也可通过构造语义树来找到，如图8-8所示。

从语义树上可以看到。仅有的满足所有子句的赋值正是前面已给出的。

我们已证明该公式是可满足的。为什么能推断非确定性搜索算法将成功终止呢？这是由于下面的事实：这个公式描述了非确定算法的执行，有一个特定的子句*SUCCESS*，坚持算法成功终止。

证明上面子句集的可满足性不仅告诉我们算法将成功终止并返回“yes”，而且回答为什么是“yes”的原因，这在赋值中可以找到。在赋值中，有一个文字

$$i = 1$$

它构造解。也就是，我们不仅知道搜索会成功回答“yes”，也明白搜索因为$x(1) = 7$而成功。

这里需要强调成功地将搜索问题转换为一个可满足性问题。不过，人们对于这种转换并不激动，因为直到今天可满足性问题还是很难解决。

例8-2 搜索问题的布尔公式（情况2）

在例8-1中，说明了一个非确定性算法可以成功终止。在本例中，将说明非确定性算法将失败终止。在这种情况下，相对应的布尔公式是不可满足的。

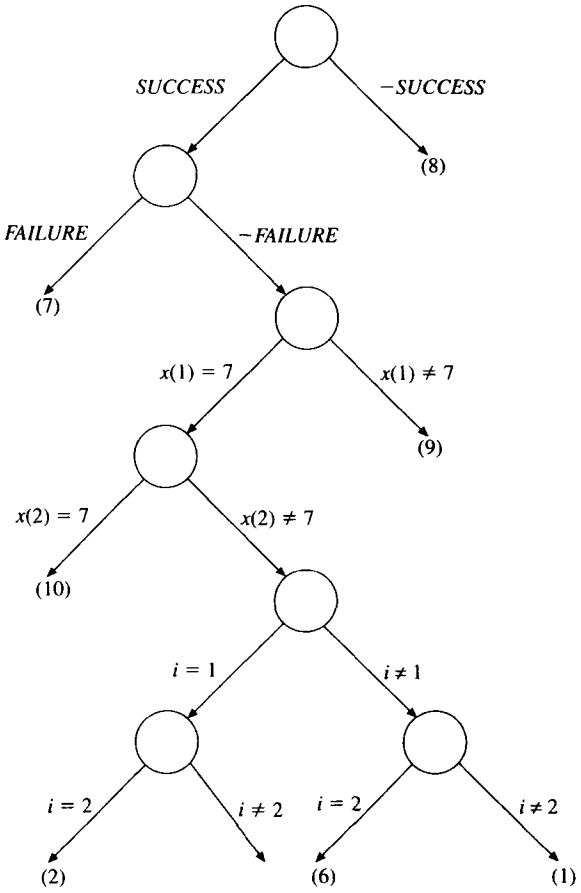


图8-8 一颗语义树

仍然使用搜索问题作为例子。为了简化讨论，假定$n = 2$，并且两个数都不等于7，那么布尔公式将包含下面的子句集：

$$i = 1 \quad \vee \quad i = 2 \tag{1}$$

$$i \neq 1 \quad \vee \quad i \neq 2 \tag{2}$$

$$x(1) \neq 7 \quad \vee \quad i \neq 1 \quad \vee \quad SUCCESS \tag{3}$$

$$x(2) \neq 7 \quad \vee \quad i \neq 2 \quad \vee \quad SUCCESS \tag{4}$$

$$x(1) = 7 \quad \vee \quad i \neq 1 \quad \vee \quad FAILURE \tag{5}$$

$$x(2) = 7 \quad \vee \quad i \neq 2 \quad \vee \quad FAILURE \tag{6}$$

$$SUCCESS \tag{7}$$

$$-SUCCESS \quad \vee \quad -FAILURE \tag{8}$$

$$x(1) \neq 7 \tag{9}$$

$$x(2) \neq 7 \tag{10}$$

上面的子句集是不可满足的，通过使用消解原理很容易证明。

$$(9)\&(5) \qquad i \neq 1 \quad \vee \quad FAILURE \tag{11}$$

$$(10)\&(6) \qquad i \neq 2 \quad \vee \quad FAILURE \tag{12}$$

$$(7)\&(8) \qquad -FAILURE \tag{13}$$

(13)&(11) $i \neq 1$ (14)

(13)&(12) $i \neq 2$ (15)

(14)&(1) $i = 2$ (16)

(15)&(16) □ (17)

上面对空子句的推导可转换成类英语式的证明，或许会很有意思：：

(1) 由于$x(1) \neq 7$和$i = 1$蕴含着*FAILURE*，$x(1)$不必等于7，所以可得

$$i = 1\text{蕴含着}FAILURE \quad (11)$$

(2) 类似地，可得

$$i = 2\text{蕴含着}FAILURE \quad (12)$$

(3) 由于我们要主张*SUCCESS*，即要得到

$$-FAILURE \quad (13)$$

(4) 所以，i既不能是1也不能是2. (14)&(15)

(5) 可是，i是1或者是2。如果i不是1，那么i必是2. (16)

(6) 导出矛盾。

例8-3 搜索问题的布尔公式（情况3）

将再次修改例8-1，使两个数都等于7。对于这种情况，将有下面的子句集：

$i = 1 \quad \vee \quad i = 2$ (1)

$i \neq 1 \quad \vee \quad i \neq 2$ (2)

$x(1) \neq 7 \quad \vee \quad i \neq 1 \quad \vee \quad SUCCESS$ (3)

$x(2) \neq 7 \quad \vee \quad i \neq 2 \quad \vee \quad SUCCESS$ (4)

$x(1) = 7 \quad \vee \quad i \neq 1 \quad \vee \quad FAILURE$ (5)

$x(2) = 7 \quad \vee \quad i \neq 2 \quad \vee \quad FAILURE$ (6)

$SUCCESS$ (7)

$-SUCCESS \quad \vee \quad -FAILURE$ (8)

$x(1) = 7$ (9)

$x(2) = 7$ (10)

构造语义树如图8-9所示。

在上面的语义树中，能够看到有两个赋值满足上面的所有子句集。一个赋值是$i = 1$，另一个是$i = 2$。

例8-4 可满足性问题的布尔公式（情况1）

本例说明将上面的思想对应于可满足性问题，也就是对可满足性问题，可以构造一个布尔公式，使原来的可满足性问题可以回答“yes”求解，当且仅当构造的布尔公式是可满足的。

考虑下面的子句集：

x_1 (1)

$-x_2$ (2)

我们将尽力确定上面的子句集是否可满足，解决这个问题的非确定性算法如下：

Do $i = 1, 2$

$x_i = choice(T, F)$

如果x_1和x_2满足子句(1)和(2)，那么*SUCCESS*，否则*FAILURE*。

我们将说明该算法如何转换成布尔公式。首先，我们知道非确定性算法依次以*SUCCESS*终止，必须使子句(1)和(2)为真。所以，

$-SUCCESS$	$\vee$	$c_1 = T$	(1)	$(SUCCESS \rightarrow c_1 = T \ \& \ c_2 = T)$
$-SUCCESS$	$\vee$	$c_2 = T$	(2)	
$-c_1 = T$	$\vee$	$x_1 = T$	(3)	$(c_1 = T \rightarrow x_1 = T)$
$-c_2 = T$	$\vee$	$x_2 = F$	(4)	$(c_2 = T \rightarrow x_2 = F)$
$x_1 = T$	$\vee$	$x_1 = F$	(5)	
$x_2 = T$	$\vee$	$x_2 = F$	(6)	
$x_1 \neq T$	$\vee$	$x_1 \neq F$	(7	
$x_2 \neq T$	$\vee$	$x_2 \neq F$	(8)	
$SUCCESS$。			(9)	

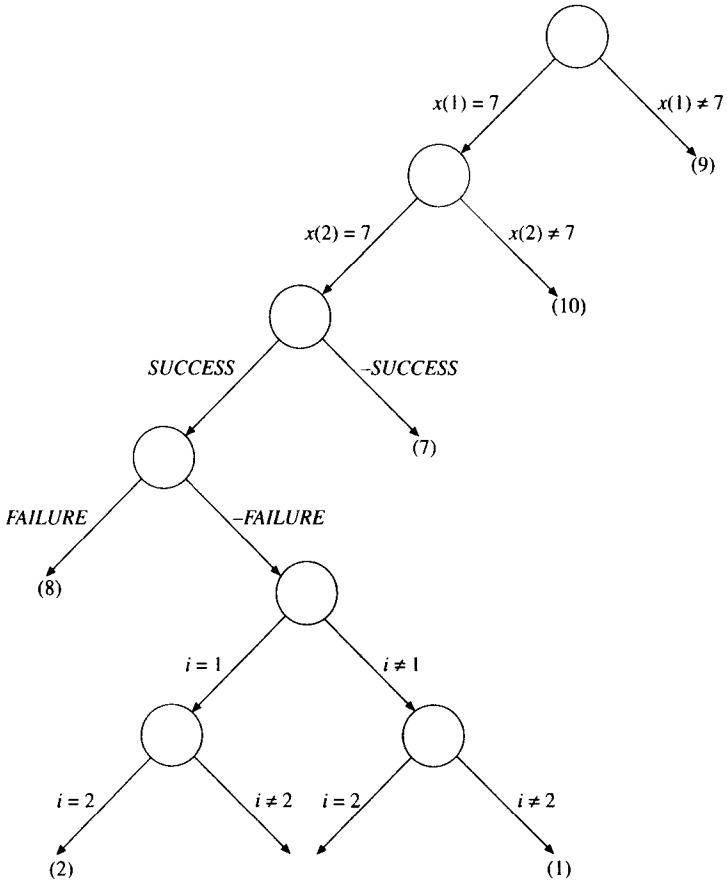


图8-9 一棵语义树

很容易理解下面的指派满足所有的子句。

$c_1 = T$	满足	(1)
$c_2 = T$	满足	(2)
$x_1 = T$	满足	(3)和(5)
$x_2 = F$	满足	(4)和(6)
$x_1 \neq F$	满足	(7)
$x_2 \neq T$	满足	(8)
$SUCCESS$	满足	(9)

所以，子句集是可满足的。

例8-5 可满足性问题的布尔公式（情况2）

在例8-4中，说明构造的公式是可满足的，因为原来的公式是可满足的。如果以不可满足的子句集开始，那么对应的公式一定也是不可满足的，这个事实可通过下面的例子说明。

考虑下面的子句集：

$$x_1 \tag{1}$$

$$-x_1 \tag{2}$$

对于上面的子句集，可构造下面的布尔公式：

$$-SUCCESS \quad \vee \quad c_1 = T \tag{1}$$

$$-SUCCESS \quad \vee \quad c_2 = T \tag{2}$$

$$-c_1 = T \quad \vee \quad x_1 = T \tag{3}$$

$$-c_2 = T \quad \vee \quad x_1 = F \tag{4}$$

$$x_1 = T \quad \vee \quad x_1 = F \tag{5}$$

$$x_1 \neq T \quad \vee \quad x_1 \neq F \tag{6}$$

$$SUCCESS \tag{7}$$

上面的子句集是不可满足的可通过使用消解原理得到证明：

$$(1)\ \&\ (7) \qquad c_1 = T \tag{8}$$

$$(2)\ \&\ (7) \qquad c_2 = T \tag{9}$$

$$(8)\ \&\ (3) \qquad x_1 = T \tag{10}$$

$$(9)\ \&\ (4) \qquad x_1 = F \tag{11}$$

$$(10)\ \&\ (6) \qquad x_1 \neq F \tag{12}$$

$$(11)\ \&(12) \qquad \square \tag{13}$$

当转换非确定性算法为布尔公式时，要小心对应的布尔公式一定不包含指数量级的子句；否则，这个转换将是无意义的。例如，假定转换含有n个变量的可满足性问题为含有2^n个子句的集合，那么，这个转换本身是一个指数级的过程。

为了强调这一点，接着考虑另一个例子（参看例8-6）。

例8-6 顶点覆盖判定问题

已知图$G = (V, E)$，在V中的顶点集S称为图G的一个顶点覆盖（node cover），如果每一条边关联S中的某个顶点。

考虑图8-10。对于此图，$S = \{v_2\}$是一个顶点覆盖，由于每条边都关联到结点v_2。

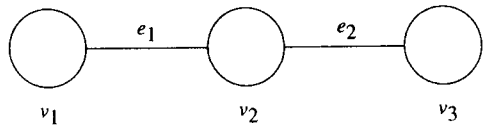


图8-10 一个图

顶点覆盖判定问题（node cover decision problem）是：已知图$G = (V, E)$和一个正整数k，确定是否存在图G的一个顶点覆盖S使得$|S| \leqslant k$。

上面的问题可以由非确定性算法解决。然而，该非确定性算法不能试探V的所有子集，由于这样的子集个数是指数级的。但是，可以使用下面的非确定性算法：令$|V| = n$，$|E| = m$。

Begin

$i_1 = choice(\{1, 2, \cdots, n\})$

$i_2 = choice(\{1, 2, \cdots, n\} - \{i_1\})$

$\vdots$

$i_k = choice(\{1, 2, \cdots, n\} - \{i_1, i_1, \cdots, i_{k-1}\})$

```
    For j := 1 to m do
        Begin
            If e_j 没有关联V_{i_t}中的某个顶点(1≤t≤k)
            then FAILURE; 终止
        End
    SUCCESS
End
```

考虑图8-10，并且假定$k=1$。在这种情况下，有下面的子句集，其中$v_i \in e_j$表明e_j关联v_i。

$$i_1 = 1 \quad \vee \quad i_1 = 2 \quad \vee \quad i_1 = 3 \tag{1}$$
$$i_1 \neq 1 \quad \vee \quad v_1 \in e_1 \quad \vee \quad FAILURE \tag{2}$$
$$i_1 \neq 1 \quad \vee \quad v_1 \in e_2 \quad \vee \quad FAILURE \tag{3}$$
$$i_1 \neq 2 \quad \vee \quad v_2 \in e_1 \quad \vee \quad FAILURE \tag{4}$$
$$i_1 \neq 2 \quad \vee \quad v_2 \in e_2 \quad \vee \quad FAILURE \tag{5}$$
$$i_1 \neq 3 \quad \vee \quad v_3 \in e_1 \quad \vee \quad FAILURE \tag{6}$$
$$i_1 \neq 3 \quad \vee \quad v_3 \in e_2 \quad \vee \quad FAILURE \tag{7}$$
$$v_1 \in e_1 \tag{8}$$
$$v_2 \in e_1 \tag{9}$$
$$v_2 \in e_2 \tag{10}$$
$$v_3 \in e_2 \tag{11}$$
$$SUCCESS \tag{12}$$
$$-SUCCESS \quad \vee \quad -FAILURE \tag{13}$$

下面的指派满足上面的子句集：

$i_1 = 2$	满足	(1)
$v_1 \in e_1$	满足	(2)和(8)
$v_2 \in e_1$	满足	(4)和(9)
$v_2 \in e_2$	满足	(5)和(10)
$v_3 \in e_2$	满足	(7)和(11)
$SUCCESS$	满足	(12)
$-FAILURE$	满足	(13)
$i_1 \neq 1$	满足	(3)
$i_1 \neq 3$	满足	(6)

由于子句集是可满足的，则顶点覆盖判定问题的回答是YES，其解是选择顶点v_2。

从上面的非正式描述，可以明白库克定理的意义。对每个NP问题A，可以转换对应于该NP问题的NP算法B为布尔公式C，使得公式C是可满足的当且仅当B成功终止并返回答案“yes”。进一步说，花费多项式步数完成这个转换。所以，如果能够在多项式步数内确定布尔公式C的可满足性，那么可明确对问题A的回答是“yes”还是“no”。或者，可以等价地说，如果可满足性问题在多项式步数内可以解决，那么每个NP问题可在多项式步数内解决。另一种说法是：如果可满足性问题在P中，那么NP = P。

注意，库克定理是在一个约束下有效：花费多项式步数转换NP问题为对应的布尔公式。如果花费指数步数来构造对应的布尔公式，那么库克定理是不成立的。

另一个要点是：尽管能够构造描述原问题的布尔公式，仍不能容易地解决原问题，因为布尔公式的可满足性并不容易确定。注意，在证明一个公式可满足时，我们一直在找满足该公式

的指派。这项工作等价于找原问题的解。非确定性算法不负责任地忽略了需要找到解的时间，只是说肯定总做出正确的猜测。而解决可满足性问题的确定性算法不能忽略找到指派的时间。库克定理表明，如果在多项式时间内找到满足布尔公式的指派，那么，可以在多项式时间内真正猜测到正确解。遗憾的是，直到现在，我们并不能在多项式时间内找到指派。因此，不能在多项式时间内正确猜测。

库克定理提醒我们在所有NP问题中可满足性问题是非常难的问题，如果它能在多项式时间内解决，那么所有的NP问题都能在多项式时间内解决。但是，在NP问题中，可满足性问题是具有此属性的唯一问题吗？我们将看到在这样的意义下，有一类问题等价于另一类问题，即如果所有问题能在多项式时间内解决，那么所有NP问题也能在多项式时间内解决。这类问题称为NP完全问题（NP-complete problem），在下一节将讨论这些问题。

8.6 NP完全问题

定义 假设A_1和A_2是两个问题，当且仅当通过使用解决A_2的多项式时间算法，A_1能在多项式时间内解决，那么称A_1规约到A_2（写作$A_1 \propto A_2$）。

从上面的定义可知，如果$A_1 \propto A_2$，并且有解决A_2的多项式时间算法，那么有解决A_1的多项式时间算法。

使用库克定理，每个NP问题都可规约到可满足性问题，因为总是首先通过解决对应于布尔公式的可满足性问题来解决NP问题。

例8-7 n元优化问题

考虑下面的问题：已知正整数$C(C>1)$和正整数n，确定是否存在正整数c_1，c_2，…，c_n，使得$\prod_{i=1}^{n} c_i = C$，且$\sum_{i=1}^{n} c_i$最小。称这样的问题为n元优化问题（n-tuple optimization problem）。

现在考虑有名的质数问题，它确定一个正整数C是否质数。显而易见，下面的关系成立：质数问题$\propto n$元优化问题。原因是显然的，在解决了n元优化问题之后，可验证解c_1，c_2，…，c_n；当且仅当存在唯一的c_i且不等于1，而其他所有的c_i都等于1，C是质数。这个验证过程只花费n步，所以是多项式过程。总之，如果n元优化问题能在多项式时间内解决，那么质数问题也能在多项式时间内解决。

直到目前，n元优化问题仍不能用任何多项式时间算法解决。

例8-8 装箱问题和桶分配问题

考虑装箱判定问题（bin packing problem）和桶分配判定问题（bucket assignment decision problem）。

装箱判定问题定义如下：已知n个物品的集合，将它们放进B个装箱中。每个装箱的容量为C，每个物品需要c_i个单位的空间。装箱判定问题是确定是否能划分这n个物品为k（$1 \leqslant k \leqslant B$）组，使得每组物品能够放入一个装箱中。

例如，令$(c_1, c_2, c_3, c_4) = (1, 4, 7, 4)$，$C = 8$及$B = 2$，我们可以划分这些物品为两组：物品1和3在一组，物品2和4在一组。

如果$(c_1, c_2, c_3, c_4) = (1, 4, 8, 4)$，$C = 8$，$B = 2$，那么没有方法能划分这些物品为两组或一组，使得每组物品能放入装箱中而不超过装箱的容积。

桶分配判定问题定义如下：已知用一个关键字做标记的n个记录，关键字假定是h个不同的值：v_1，v_2，…，v_h，并有n_i个记录对应于v_i，也就是$n_1 + n_2 + \cdots + n_h = n$。桶分配判定问题是确定是否能将这$n$个记录放入$k$个桶中，以致具有相同$v_i$的记录在一个桶中，并且没有桶容纳超过$C$个记录。

例如，令关键字值为a，b，c和d，$(n_a, n_b, n_c, n_d) = (1, 4, 2, 3)$，$k = 2$及$C = 5$，那么将这些记录放入如下两个桶中：

桶1	桶2
a	c
b	c
b	d
b	d
b	d

如果$(n_a, n_b, n_c, n_d) = (2, 4, 2, 2)$，那么在不超过每个桶容量的情况下没有方法分配这些记录到桶中，使同一个桶有同样的关键字值。

桶分配判定问题是个有趣的问题。如果记录存在磁盘上，那么桶分配判定问题相关于最小化磁盘存取次数问题。当然，如果要最小化磁盘存取的总次数，应该把同样关键字的记录尽可能放入同一个桶中。

很容易证明装箱问题能规约到桶分配判定问题。每个能在多项式时间步数内产生的装箱问题都对应一个桶分配判定问题。所以，如果有解决桶分配判定问题的多项式算法，那么能在多项式时间内解决装箱问题。

从“规约到”的定义可容易地得到：如果$A_1 \propto A_2$，$A_2 \propto A_3$，那么$A_1 \propto A_3$。

有了“规约到”的定义，现在能够定义NP完全问题。

定义 如果$A \in$ NP，且每个NP问题规约到A，那么问题A是NP完全的。

从上面的定义，如果A是NP完全问题，并且能在多项式时间内解决，那么每个NP问题能够在多项式时间内解决。显然，由于库克定理，可满足性问题是NP完全问题。

根据定义，如果任何NP完全问题能在多项式时间内解决，那么NP = P。

可满足性问题是发现的第一个NP完全问题。随后，R. Karp证明了21个NP完全问题，这些NP完全问题包括顶点覆盖，反馈弧集问题（feedback arc set），哈密顿回路等。在1985年，Karp获得了图灵奖。

为了证明一个问题A是NP完全的，不必证明所有NP问题都规约到A。这是库克在证明可满足性问题的NP完全性时所做的。现在，只使用“规约到”的传递属性。如果A_1是一个NP完全问题，A_2是一个NP问题，并且能证明$A_1 \propto A_2$，那么A_2也是一个NP完全问题。这个推理是相当直接的。如果A是一个NP完全问题，那么所有NP问题可规约到A。如果$A \propto B$，那么由于“规约到”的传递属性，所有NP问题可规约到B，所以B一定是NP完全的。

在上面的讨论中，我们做下面的陈述：

(1) 可满足性问题在所有NP问题中是最难的问题。

(2) 在证明一个问题A的NP完全性时，我们经常尽力证明可满足性问题可规约到A。因此，显然A比可满足性问题难。

为了明白这里的陈述没有矛盾，要注意每个NP问题可规约到可满足性问题。所以，如果我们对一个NP问题A感兴趣，那么，当然A可规约到可满足性问题。然而，这里必须强调：说一个问题A规约到可满足性问题根本不重要，因为它只意味着可满足性问题比A难得多，这已是很好接受的事实。如果成功地证明可满足性问题可规约到A，那么A甚至比可满足性问题更困难，这是更重要的陈述。注意$A \propto$ 可满足性问题，并且可满足性问题$\propto A$。所以，只要关注困难程度，A等价于可满足性问题。

可以扩展上面所有NP完全问题的主题。如果A是一个NP完全问题，那么根据定义每个NP问题，比如B，可规约到A。如果进一步通过$A \propto B$证明B是NP完全的，那么A和B是相互等价的。

总之，所有NP完全问题形成一个等价类。

注意，我们限定NP问题是判定问题。现在通过定义“NP困难性”（NP-hardness）扩展NP完全性概念到优化问题。一个问题A是NP难的，如果每个NP问题可规约到A。（注意A不必是NP问题。实际上，A可以是一个优化问题。）所以，如果A是NP难的，并且A是一个NP，那么问题是NP完全的。通过这条途径，优化问题是NP难的，如果它对应的判定问题是NP完全的。例如，旅行商问题是NP难的。

8.7 证明NP完全性的例子

在本节中将证明许多问题都是NP完全的。我们提醒读者要证明问题A是NP完全的，通常做下面两步：

(1) 首先证明A是NP问题；

(2) 然后证明某个NP完全问题可规约到A。

许多读者错误地证明A可规约到某个NP完全问题。这是毫无意义的，因为根据定义，每个NP问题可规约到每个NP完全问题。

注意，直到目前，我们只接受可满足性问题是NP完全的。为了产生更多NP完全的问题，必须从可满足性问题开始。也就是说，应该尽力证明可满足性问题可规约到我们感兴趣的问题上。

例8-9 3可满足性问题

3可满足性问题（3-satisfiability problem）类似于可满足性问题，但有更多的限制：每个子句恰好包含三个文字。

显然，3可满足性问题是NP问题。为了证明它是NP完全问题，需要证明可满足性问题可规约到3可满足性问题。我们将证明，对每个任意布尔公式F_1，可以生成另一个布尔公式F_2，在公式F_2中的每个子句恰好包含三个文字，使得F_1是可满足的当且仅当F_2是可满足的。

首先从一个例子开始。考虑下面的子句集：

$$x_1 \quad \vee \quad x_2 \tag{1}$$

$$-x_1 \tag{2}$$

可以扩展上面的子句集，使现在每个子集包含三个文字：

$$x_1 \quad \vee \quad x_2 \quad \vee \quad y_1 \tag{1'}$$

$$-x_1 \quad \vee \quad y_2 \quad \vee \quad y_3 \tag{2'}$$

可以看到(1)&(2)是可满足的，(1)'&(2)'也是可满足的。但是，上面增加新文字的方法可能产生问题，比如看到下面的情况：

$$x_1 \tag{1}$$

$$-x_1 \tag{2}$$

(1)&(2)是不可满足的。如果增加任意一些新文字到这两个子句中：

$$x_1 \quad \vee \quad y_1 \quad \vee \quad y_2 \tag{1'}$$

$$-x_1 \quad \vee \quad y_3 \quad \vee \quad y_4 \tag{2'}$$

(1)'&(2)'成为可满足的。

上面的讨论表明不能任意增加新文字到公式中，而不影响其可满足性。我们能够做的是添加新的子句，而它们本身对于原来的子句集是不可满足的。如果原子句集是可满足的，那么新的子句集也将是可满足的。如果原子句集是不可满足的，那么新的子句集也将是不可满足的。

如果原子句只包含一个文字，我们可以增加下面的子句集：

$$
\begin{array}{ccc}
y_1 & \vee & y_2 \\
-y_1 & \vee & y_2 \\
y_1 & \vee & -y_2 \\
-y_1 & \vee & -y_2
\end{array}
$$

例如，假如原子句是x_1，那么新生成的子句将是：

$$
\begin{array}{ccccc}
x_1 & \vee & y_1 & \vee & y_2 \\
x_1 & \vee & -y_1 & \vee & y_2 \\
x_1 & \vee & y_1 & \vee & -y_2 \\
x_1 & \vee & -y_1 & \vee & -y_2
\end{array}
$$

如果原子句包含两个文字，可以增加

$$
\begin{array}{c}
y_1 \\
-y_1
\end{array}
$$

例如，假定原子句是

$$
x_1 \quad \vee \quad x_2
$$

那么新生成的子句将是

$$
\begin{array}{ccccc}
x_1 & \vee & x_2 & \vee & y_1 \\
x_1 & \vee & x_2 & \vee & -y_1
\end{array}
$$

考虑下面的不可满足子句集：

$$
\begin{array}{c}
x_1 \qquad (1) \\
-x_1 \qquad (2)
\end{array}
$$

现在我们有

$$
\begin{array}{ccccc}
x_1 & \vee & y_1 & \vee & y_2 \\
x_1 & \vee & -y_1 & \vee & y_2 \\
x_1 & \vee & y_1 & \vee & -y_2 \\
x_1 & \vee & -y_1 & \vee & -y_2 \\
-x_1 & \vee & y_3 & \vee & y_4 \\
-x_1 & \vee & -y_3 & \vee & y_4 \\
-x_1 & \vee & y_3 & \vee & -y_4 \\
-x_1 & \vee & -y_3 & \vee & -y_4
\end{array}
$$

这新的子句集仍是不可满足的。

如果子句含有多于三个的文字，可通过增加下面的不可满足子句集将该子句分成一个新子句集：

$$
\begin{array}{ccc}
y_1 & & \\
-y_1 & \vee & y_2 \\
-y_2 & \vee & y_3 \\
 & \vdots & \\
-y_{i-1} & \vee & y_i \\
-y_i & &
\end{array}
$$

考虑下面的子句：

$$x_1 \quad \vee \quad -x_2 \quad \vee \quad x_3 \quad \vee \quad x_4 \quad \vee \quad -x_5$$

可以增加新的变量以生成只包含三个文字的新子句：

$$\begin{array}{ccccc} x_1 & \vee & -x_2 & \vee & y_1 \\ x_3 & \vee & -y_1 & \vee & y_2 \\ x_4 & \vee & -x_5 & \vee & -y_2 \end{array}$$

我们可以总结转换子句到恰包含三个文字子句集的规则。（下面所有的y_i表示新的变量。）

(1) 如果子句只包含一个文字L_1，那么生成下面的四个子句：

$$\begin{array}{ccccc} L_1 & \vee & y_1 & \vee & y_2 \\ L_1 & \vee & -y_1 & \vee & y_2 \\ L_1 & \vee & y_1 & \vee & -y_2 \\ L_1 & \vee & -y_1 & \vee & -y_2 \end{array}$$

(2) 如果子句包含两个文字L_1和L_2，那么生成下面的两个子句：

$$\begin{array}{ccccc} L_1 & \vee & L_2 & \vee & y_1 \\ L_1 & \vee & L_2 & \vee & -y_1 \end{array}$$

(3) 如果子句包含三个文字，那么什么也不作。

(4) 如果子句包含多于三个文字，那么如下生成新子句：假定文字是L_1，L_2，…，L_k，那么新子句是

$$\begin{array}{ccccc} L_1 & \vee & L_2 & \vee & y_1 \\ L_3 & \vee & -y_1 & \vee & y_2 \\ \vdots & & & & \\ L_{k-1} & \vee & L_k & \vee & -y_{k-3} \end{array}$$

考虑下面的子句集：

$$\begin{array}{ccccccccc} x_1 & \vee & x_2 & & & & & & \\ -x_3 & & & & & & & & \\ x_1 & \vee & -x_2 & \vee & x_3 & \vee & -x_4 & \vee & x_5 \end{array}$$

现在生成下面的子句集：

$$\begin{array}{ccccc} x_1 & \vee & x_2 & \vee & y_1 \\ x_1 & \vee & x_2 & \vee & -y_1 \\ -x_3 & \vee & y_2 & \vee & y_3 \\ -x_3 & \vee & -y_2 & \vee & y_3 \\ -x_3 & \vee & y_2 & \vee & -y_3 \\ -x_3 & \vee & -y_2 & \vee & -y_3 \\ x_1 & \vee & -x_2 & \vee & y_4 \\ x_3 & \vee & -y_4 & \vee & y_5 \\ -x_4 & \vee & x_5 & \vee & -y_5 \end{array}$$

正如前面所指出的，上面的转换必须保持原子句集的可满足属性。也就是，令S表示原子句集，S'表示转换后的子句集，S'中每个子句含有三个文字，那么S'是可满足的当且仅当S是可满足的。这将在下面证明。

(1) 第一部分：如果S是可满足的，那么S'是可满足的。其中I表示满足S的一个指派，那么，

显然I满足包含不超过三个文字的子句生成的所有子句。令C是一个在S中含有超过三个文字的子句，且$T(C)$是在S'中与C相关的子句集。例如，对于

$$C = x_1 \vee x_2 \vee -x_3 \vee x_4 \vee -x_5$$

$$T(C) = \begin{cases} x_1 \vee x_2 \vee y_1 \\ -x_3 \vee -y_1 \vee y_2 \\ x_4 \vee -x_5 \vee -y_2 \end{cases}$$

如同假设的I满足C。如果I也满足$T(C)$，我们已做。否则，I必须至少满足$T(C)$的一个子集。现在解释I怎么能扩展到I'，使I'满足$T(C)$中的所有子句。这可解释如下：令C_i是$T(C)$中一个I'满足的子句，赋值子句C_i的最后文字为假，这样的赋值将满足$T(C)$中另一个子句C_j。如果$T(C)$中每个子句都满足，我们已做；否则，赋值子句C_j的最后文字为假，此过程可重复直到每个句子都满足。

考虑$C = x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$。

$T(C)$是：

$$x_1 \quad \vee \quad x_2 \quad \vee \quad y_1 \tag{1}$$

$$x_3 \quad \vee \quad -y_1 \quad \vee \quad y_2 \tag{2}$$

$$x_4 \quad \vee \quad x_5 \quad \vee \quad -y_2 \tag{3}$$

假如$I = \{x_1\}$，则I满足(1)。我们赋值(1)中的y_1为假。因此，$I' = \{x_1, -y_1\}$和(2)都可满足。进一步赋值(2)中的y_2为假，这最终使(3)满足，所以，$I' = \{x_1, -y_1, -y_2\}$满足所有的子句。

(2) 第二部分：如果S'是可满足的，那么S是可满足的。对于S'，新添加的子句本身是不可满足的。所以，如果S'是可满足的，那么满足S'的赋值只不包含y_i，它必须对某原始变量x_i赋值真。考虑S'中子句集S_j，它是对应于通过S中的句子C_j生成的。由于满足S_j的任何赋值一定满足至少C_j中的一个文字，这样的赋值满足C_j。所以，如果S'是可满足的，那么S也是可满足的。

由于要花费多项式时间转换任意子句集S为子句集S'，每个S'中的子句都含有三个文字，并且是可满足的当且仅当S是可满足的，我们推断，如果能在多项式时间内解决3可满足性问题，那么也可在多项式时间内解决可满足性问题。因此，可满足性问题规约到3可满足性问题，而且3可满足性问题是NP完全的。

在上面的例子中，证明了3可满足性问题是NP完全的。可能许多读者在这一点上心存疑惑，即根据下面错误的推断：3可满足性问题只是可满足性问题的一种特殊情况。由于可满足性问题是一个NP完全问题，则3可满足性问题自然也是NP完全问题。

一般问题的特殊情况的难度是不能通过验证一般问题来推导的。考虑可满足性问题，它是NP完全的，但可满足问题的特殊情况不是NP完全的，是完全可能的。考虑每个子句只含有正文字的情况，也就是，没有负号出现。一个典型的例子如下：

$x_1 \quad \vee \quad x_2 \quad \vee \quad x_3$

$x_1 \quad \vee \quad x_4$

$x_4 \quad \vee \quad x_5$

x_6

在本例中，满足性赋值可通过赋值每个变量为T很容易得到。因此，这个可满足性问题的特例不是NP完全的。

一般来说，如果问题是NP完全的，它的特例可能是也可能不是NP完全的。另一方面，如果一个问题的特例是NP完全的，那么该问题是NP完全的。

例8-10　色数判定问题

本例将证明色数判定问题（chromatic number decision problem）是NP完全的。色数判定问题定义如下：已知图$G = (V, E)$，对每个顶点着色，如果两个顶点有一条边连接，那么这两个顶点一定要着不同的颜色。色数判定问题是确定能否使用k种颜色对所有顶点着色。

考虑图8-11。对于该图，使用三种颜色对该图着色如下：

$a \leftarrow 1$，$b \leftarrow 2$，$c \leftarrow 1$，$d \leftarrow 2$，$e \leftarrow 3$

考虑图8-12，很容易明白需要四种颜色着色。

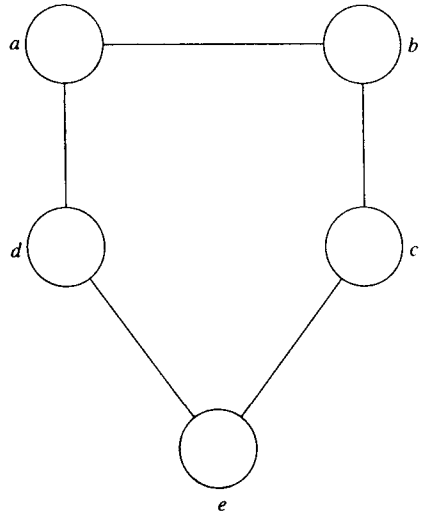


图8-11 一个3可着色图

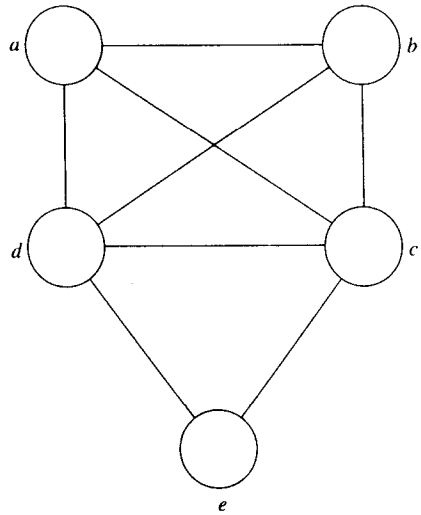


图8-12 一个4可着色图

为了证明色数判定问题是NP完全的，需要一个NP完全问题证明该NP完全问题可规约到色数判定问题。在本例中，采用例8-9中讨论过的一个类似3可满足性的问题。该问题是可满足性问题，它的每个子句至多含有三个文字。由于例8-9的讨论，很容易明白每个子句具有最多三个文字的可满足性问题也是NP完全的。对任意的一个可满足性问题，总是可以转换为一个可满足性问题，该问题的每个子句具有最多三个文字，而不影响原问题的可满足性。当然要将多于三个文字的子句分成只含有三个文字的子句。

现在证明每个至多含有三个文字的子句可满足性问题可规约到色数判定问题。本质上要证明每个子句至多含有三个文字的可满足性问题可构造一个对应的图，使原布尔公式是可满足的，当且仅当所构造的图能用$n + 1$种颜色着色，其中n是出现在布尔公式中的变量数。

令x_1，x_2，…，x_n标记布尔公式F中的变量，其中$n \geqslant 4$。如果$n<4$，那么n是常数，可满足性问题可很容易地确定。令C_1，C_2，…，C_r是子句，每个子句至多含有三个文字。

对应于布尔公式的图G依据如下的规则构造：

(1) 图G的顶点是x_1，x_2，…，x_n，$-x_1$，$-x_2$，…，$-x_n$，y_1，y_2，…，y_n，C_1，C_2，…，C_r。

(2) 图G的边通过如下的规则形成：

(a) 对于$1 \leqslant i \leqslant n$，在每一对$x_i$和$-x_i$之间有一条边。

(b) 如果$i \neq j$，$1 \leqslant i$，$j \leqslant n$，那么在每一对y_i和y_j之间有一条边。

(c) 如果$i \neq j$，$1 \leqslant i$，$j \leqslant n$，那么在每一对y_i和x_j之间有一条边。

(d) 如果$i \neq j$，$1 \leqslant i$，$j \leqslant n$，那么在每一对y_i和$-x_j$之间有一条边。

(e) 如果$x_i \notin C_j$，$1 \leqslant i \leqslant n$，$1 \leqslant j \leqslant r$，那么在每一对$x_i$和$C_j$之间有一条边。

(f) 如果$-x_i \notin C_j$，$1 \leqslant i \leqslant n$，$1 \leqslant j \leqslant r$，那么在每一对$-x_i$和$C_j$之间有一条边。

现在证明F是可满足的当且仅当G是$n+1$可着色的，该证明由两部分组成：

(1) 如果F是可满足的，那么G是$n+1$可着色的。

(2) 如果G是$n+1$可着色的，那么F是可满足的。

首先证明第一部分。假定F是可满足的。在这种情况下，可以选择任何满足F的指派A，按如下方式对顶点着色：

(1) 对所有的y_i，y_i用颜色i着色。

(2) 对所有的x_i和$-x_i$，按如下方法进行着色：如果x_i在A中赋值为T，那么x_i用颜色i着色，$-x_i$用颜色$n+1$着色；否则，x_i用颜色$n+1$着色，$-x_i$用颜色i着色。

(3) 对于每个C_j，在C_j中找出一个文字L_i，其在A中为真。由于A满足每个子句，所有对于所有的子句L_i存在，分配C_j与此文字同样的颜色。也就是，如果L_i是x_i，那么分配C_j与x_i同样的颜色；否则，分配C_j与$-x_i$同样的颜色。

考虑下面的子句集：

$$x_1 \quad \vee \quad x_2 \quad \vee \quad x_3 \tag{1}$$

$$-x_3 \quad \vee \quad -x_4 \quad \vee \quad x_2 \tag{2}$$

着色图如图8-13所示。

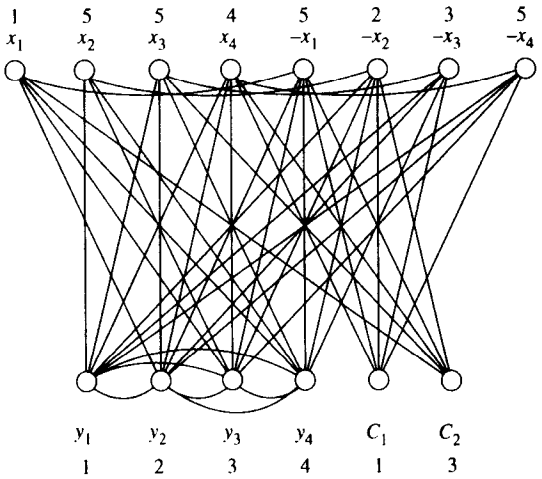


图8-13 一个色数判定问题构造的图

令$A=(x_1, -x_2, -x_3, x_4)$，那么x_i赋值如下：

$x_1 \leftarrow 1$，$-x_1 \leftarrow 5$

$x_2 \leftarrow 5$，$-x_2 \leftarrow 2$

$x_3 \leftarrow 5$，$-x_3 \leftarrow 3$

$x_4 \leftarrow 4$，$-x_4 \leftarrow 5$

C_j着色如下：

$C_1 \leftarrow 1$　　(x_1在A中满足C_1。)

$C_2 \leftarrow 3$　　($-x_3$在A中满足C_2。)

为了证明上面的着色是合法的，可以进行下面的推理：

(1) 如果$i \neq j$，那么每个y_i都连接到每个y_j。所以，没有两个y_i着相同的颜色，这如同将y_i着颜色i一样。

(2) 每个x_i连接到$-x_i$，所以x_i和$-x_i$不能着相同的颜色。这样做是因为没有x_i和$-x_i$分配同样的颜色。此外，根据规则没有y_j与x_i或$-x_i$着相同的颜色。

(3) 考虑C_j。假如L_i在C_j中出现，L_i在A中为真，那么C_j着与L_i相同的颜色。只有特殊的$-x_i$或x_i等于L_i，那么所有的x_i和$-x_i$将着与C_j相同的颜色。但是，L_i不能连接到C_j，因为L_i出现在C_j中。由于连接到C_j，所以没有C_j与x_i或$-x_i$有同样的颜色。

从上面的讨论可以推断，如果F是可满足的，那么G是$n+1$可着色的。

现在证明另一部分，如果G是$n+1$可着色的，那么F是可满足的。推导如下：

(1) 不失一般性，假定y_i着以颜色i。

(2) 由于x_i与$-x_i$相连接，所以，x_i与$-x_i$不能分配相同的颜色。如果$i \neq j$，那么x_i和$-x_i$与y_j相连接，所以或者x_i用颜色i着色，$-x_i$用颜色$n+1$着色；或者$-x_i$用颜色i着色，x_i用颜色$n+1$着色。

(3) 由于每个C_j含有至多三个文字以及$n \geqslant 4$，所以至少有一个i使x_i和$-x_i$都不出现在C_j中。因此，每个C_j连接到至少一个用$n+1$着色的顶点上。因此，没有C_j用$n+1$着色。

(4) 对于每个C_j，$1 \leqslant j \leqslant r$。如果$C_j$用颜色$i$着色，$x_i$用颜色$i$着色，那么赋值$A$对$x_i$赋值为真。

如果C_j用颜色i着色，$-x_i$用颜色i着色，那么指派A对x_i指派为假。

(5) 需要注意C_j用颜色i着色，x_i和$-x_i$无论哪个用颜色i着色。它一定不能连接到x_i或$-x_i$，这意味着用颜色i着色的文字一定在C_j中出现。在A中，这个特殊的文字指派为真，所以满足C_j。因此，A满足所有的子句。由于存在至少一个指派满足所有的子句，所有F一定是可满足的。

在上面的讨论中，我们证明了对每个含有至多三个文字的子句集，可以构造一个图使n个变量的原子句集是可满足的，当且仅当对应的图是$n+1$可着色的。此外，很容易证明图的构造花费了多项式步数。所以，每个子句含有至多三个文字的可满足性问题可规约到色数判定问题，色数判定问题是NP完全的。

在本节的剩余部分中，将证明VLSI离散布图问题（discrete layout problem）是NP完全的。为了证明，首先必须证明其他一些问题是NP完全的。

例8-11 严格覆盖问题

假定集合族$F=\{S_1, S_2, \cdots, S_k\}$和具有元素$\{u_1, u_2, \cdots, u_n\}$的集合$S$，集合$S$是$\bigcup_{S_i \in F} S_i$。严格覆盖问题（exact cover problem）是确定是否有一个$T \subseteq F$的互不相交的子集使得

$$\bigcup_{S_i \in T} S_i = \{u_1, u_2, \cdots, u_n\} = \bigcup_{S_i \in F} S_i$$

例如，假定$F=\{(a_3, a_1), (a_2, a_4), (a_2, a_3)\}$，那么$T=\{(a_3, a_1), (a_2, a_4)\}$是$F$的一个严格覆盖。注意$T$中的每对集合必须是不相交的。如果$F=\{(a_3, a_1), (a_4, a_3), (a_2, a_3)\}$，那么没有严格覆盖。

现在设法证明这个严格覆盖问题是NP完全的，通过将着色问题规约到严格覆盖问题得到。例8-10中对着色问题已作介绍。

令已知在着色问题中的图顶点集是$V=\{v_1, v_2, \cdots, v_n\}$，边集是$E=\{e_1, e_2, \cdots, e_m\}$，以及一个整数$k$，现在转换这个着色问题实例为严格覆盖问题实例$S=\{v_1, v_2, \cdots, v_n, E_{11}, E_{12}, \cdots, E_{1k}, E_{21}, E_{22}, \cdots, E_{2k}, \cdots, E_{m1}, E_{m2}, \cdots, E_{mk}\}$，其中$E_{i1}, E_{i2}, \cdots, E_{ik}$对应于$e_i$，$1 \leqslant i \leqslant m$，并且子集族$F=\{C_{11}, C_{12}, \cdots, C_{1k}, \cdots, C_{n1}, C_{n2}, \cdots, C_{nk}, D_{11}, D_{12}, \cdots, D_{1k}, \cdots, D_{m1}, D_{m2}, \cdots, D_{mk}\}$。每个$C_{ij}$和$D_{ij}$都是根据下面规则确定：

(1) 如果边e_i以顶点v_a和v_b作为终端，那么对$d=1, 2, \cdots, k$，C_{ad}和C_{bd}都包含E_{id}。

(2) 对所有的i和j，$D_{ij}=\{E_{ij}\}$。

(3) 对$j=1, 2, \cdots, k$，C_{ij}包含v_i。

举一个例子，考虑图8-14。

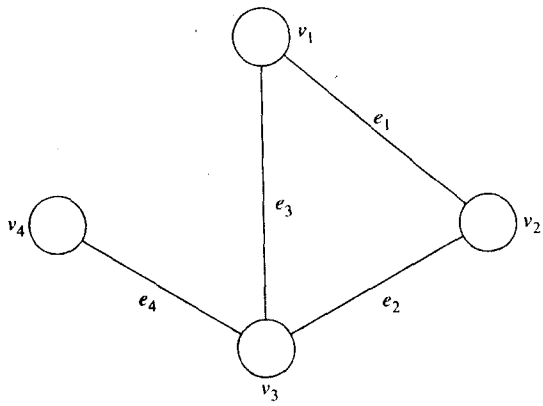


图8-14 说明着色问题到严格覆盖问题的转换图

在本例中，$n=4$，$m=4$。假如$k=3$，因此有$S=\{v_1, v_2, v_3, v_4, E_{11}, E_{12}, E_{13}, E_{21}, E_{22}, E_{23}, E_{31}, E_{32}, E_{33}, E_{41}, E_{42}, E_{43}\}$，$F=\{C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, C_{33}, C_{41}, C_{42}, C_{43}, D_{11}, D_{12}, D_{13}, D_{21}, D_{22}, D_{23}, D_{31}, D_{32}, D_{33}, D_{41}, D_{42}, D_{43}\}$，每个$D_{ij}$只包含一个$E_{ij}$。对于$C_{ij}$，我们通过例子来说明其内容。考虑$e_1$，它连接$v_1$和$v_2$。这意味着$C_{11}$和$C_{12}$都含有$E_{11}$。类似地，$C_{11}$和$C_{12}$都含有$E_{12}$。$C_{11}$和$C_{13}$都含有$E_{13}$。

整个集合族F构造如下：

$C_{11}=\{E_{11}, E_{31}, v_1\}$，

$C_{12}=\{E_{12}, E_{32}, v_1\}$，

$C_{13}=\{E_{13}, E_{33}, v_1\}$,
$C_{21}=\{E_{11}, E_{21}, v_2\}$,
$C_{22}=\{E_{12}, E_{22}, v_2\}$,
$C_{23}=\{E_{13}, E_{23}, v_2\}$,
$C_{31}=\{E_{21}, E_{31}, E_{41}, v_3\}$,
$C_{32}=\{E_{22}, E_{32}, E_{42}, v_3\}$,
$C_{33}=\{E_{23}, E_{33}, E_{43}, v_3\}$,
$C_{41}=\{E_{41}, v_4\}$,
$C_{42}=\{E_{42}, v_4\}$,
$C_{43}=\{E_{43}, v_4\}$,
$D_{11}=\{E_{11}\}$, $D_{12}=\{E_{12}\}$, $D_{13}=\{E_{13}\}$,
$D_{21}=\{E_{21}\}$, $D_{22}=\{E_{22}\}$, $D_{23}=\{E_{23}\}$,
$D_{31}=\{E_{31}\}$, $D_{32}=\{E_{32}\}$, $D_{33}=\{E_{33}\}$,
$D_{41}=\{E_{41}\}$, $D_{42}=\{E_{42}\}$, $D_{43}=\{E_{43}\}$。

我们忽略着色问题是k可着色的当且仅当构造的严格覆盖问题有解。同时，通过实例说明这种转换的有效性。

图8-14是3可着色的。使v_1，v_2，v_3和v_4分别着色1，2，3和1。在本例中，选择C_{11}，C_{22}，C_{33}，C_{41}，D_{13}，D_{21}，D_{32}和D_{42}作为覆盖。首先很容易明白它们是互不相交的，从而，集合S严格被这些集合覆盖。例如，v_1，v_2，v_3和v_4分别被C_{11}，C_{22}，C_{33}和C_{41}所覆盖，E_{11}被C_{11}所覆盖，E_{12}被C_{22}所覆盖，以及E_{13}被D_{13}所覆盖。

这个可规约性的形式化证明留做练习。

例8-12 子集和问题

子集和问题（sum of subsets problem）定义如下：假定数的集合$A=\{a_1, a_2, \cdots, a_n\}$和常数$C$，确定是否存在$A$的子集$A'$，使得$A'$中元素相加为$C$。

例如，令$A=\{7, 5, 19, 1, 12, 8, 14\}$和常数$C$为21，那么这个子集和问题有解即$A'=\{7, 14\}$。如果$C$为11，那么该子集和问题无解。

可通过规约严格覆盖问题到子集和问题来证明该问题的NP完全性。已知严格覆盖问题实例为$F=\{S_1, S_2, \cdots, S_k\}$，以及集合$S=\bigcup_{S_i\in F} S_i=\{u_1, u_2, \cdots, u_m\}$。根据下面的规则构造对应的子集和问题的实例：子集和问题实例包含集合$A=\{a_1, a_2, \cdots, a_n\}$，其中

$a_j=\sum_{1\leqslant i\leqslant m} e_{ji}(n+1)^{i-1}$，如果$u_i\in S_j$，那么$e_{ji}=1$；否则$e_{ji}=0$。

$$C=\sum_{0\leqslant i\leqslant m}(n+1)^i=((n+1)^m-1)/n$$

同样，将上面转换的有效性留做练习。

例8-13 分割判定问题

分割问题（partition problem）定义如下：已知$A=\{a_1, a_2, \cdots, a_n\}$，每个$a_i$都是正整数。分割问题是确定是否有一个分割$A=\{A_1, A_2\}$，使得

$$\sum_{a_i\in A_1} a_i = \sum_{a_i\in A_2} a_i$$

例如，令$A=\{1, 3, 8, 4, 10\}$，可分割A成两个子集$\{1, 8, 4\}$和$\{3, 10\}$。很容易验证第一个子集中元素和等于第二个集合中元素和。

该问题的NP完全性可通过规约子集和问题为该问题来证明，如何证明也留做练习。

例8-14 装箱判定问题

之前介绍的装箱判定问题描述如下：已知n个物品的集合，每个大小c_i是正整数，还已知正整数B和C，分别是装箱个数和装箱容积。确定能否分配物品到k个装箱中（$1 \leqslant k \leqslant B$），使所有分配到每个装箱中的物品的$c_i$之和不超过$C$。

可以通过规约分割问题到该问题证明其NP完全性。假如分割问题实例有$A = \{a_1, a_2, \cdots, a_n\}$，可通过设定$B = 2$，$c_i = a_i$，$1 \leqslant i \leqslant n$和$C = \sum_{1 \leqslant i \leqslant n} a_i / 2$来定义相应的装箱判定问题。显然，装箱判定问题当$k=2$有解，当且仅当存在对$A$的分割。

例8-15 VLSI离散布图问题

在本题中，已知n个矩形的集合S，在某些限制下，确定是否可能将这些矩形放到一个有特定面积的大矩形中。形式化地说，已知n个矩形的集合和一个整数A。对于$1 \leqslant i \leqslant n$，每个矩形$r_i$有长、宽尺寸$h_i$和$w_i$（$h_i$和$w_i$为正整数）。VLSI离散布图问题是确定能否在该平面上放置这些矩形，使得：

(1) 每条边界都平行于某一系统坐标轴。

(2) 矩形的角放在平面的整数点上。

(3) 没有两个矩形重叠。

(4) 两个矩形的边界有至少1个单位长度间隔。

(5) 在平面中有一个矩形确定了放置其他矩形的范围，它的边界平行于坐标轴且有至多为A的面积。确定范围的矩形边界允许容纳放置矩形的边界。

图8-15所示为一个成功放置。VLSI离散布图问题的NP完全性可通过证明将装箱判定问题规约到VLSI离散布图问题来证明。在装箱判定问题中，已知n个物品及每件物品的大小是c_i，装箱数是B，以及每个装箱的容量是C。对于装箱判定问题，构造VLSI离散布图问题如下：

(1) 对应于每个c_i，有高度为$h_i = 1$和宽度$w_i = (2B + 1)\, c_i - 1$的矩形r_i。

(2) 另外，有另一个宽度为$w = (2B + 1)C - 1$和高度为$h = 2Bw + 1$的矩形。

(3) 边界的矩形面积是$A = w(h + 2B)$。

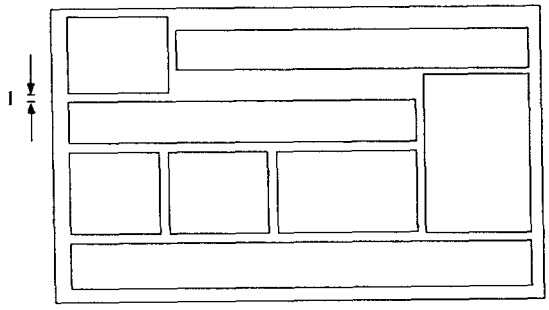


图8-15 一个成功的放置

首先证明如果装箱判定问题有解，那么构造的VLSI离散布图问题也有解。假定对装箱i，存放其中物品的大小是d_{i1}，d_{i2}，$\cdots$，d_{ij_i}。放置按行安排对应的矩形，使行的高度保持为1，如图8-16所示。如果使用k个装箱，那么将有k行高度为1的矩形。然后在这些行下放置具有宽度为w、高度为h的矩形，如图8-17所示。

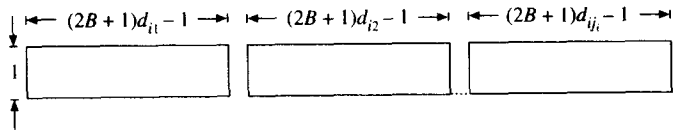


图8-16 特定的放置行

每个行为i的j_i个矩形，宽度可按如下方法找到：

$$\sum_{a=1}^{j_i}((2B+1)d_{i_a}-1)+(j_i-1)$$
$$=(2B+1)\left(\sum_{a=1}^{j_i}d_{i_a}\right)-j_i+j_i-1$$
$$\leqslant(2B+1)C-1$$
$$=w$$

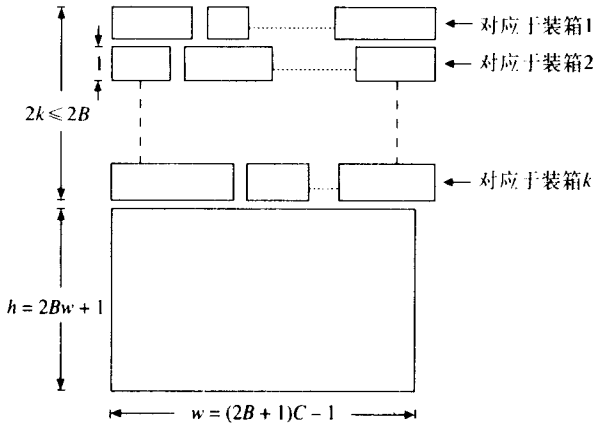


图8-17 $n+1$个矩形的放置

所以，这$n+1$个矩形的宽度小于或等于w，而对应的高度小于$(2k+h)\leqslant(2B+h)$。放置这$n+1$个矩形的总面积因此小于

$$(2B+h)w=A$$

这意味着如果装箱判定问题有解，那么对应的$n+1$个矩形可放置在大小为A的矩形中。

现在证明另一个方面。假定成功在一个大小为$A=w(h+2B)$的矩形内放置了$n+1$个矩形。证明原装箱判定问题有解。

主要的推理是成功地放置不能是任意的，必须在一些限制下，这些限制如下：

(1) 放置的宽度必须小于$w+1$。如若不然，由于有一个高度为h的矩形，总的面积将大于

$$h(w+1)$$
$$=hw+h$$
$$=hw+2Bw+1$$
$$=w(h+2B)+1$$
$$=A+1>A,$$

这是不可能的。

(2) 每个矩形r_i一定以高度为1的方法放置。换句话说，矩形r_i必须按图8-18a方式放置，而不能按图8-18b方式放置。

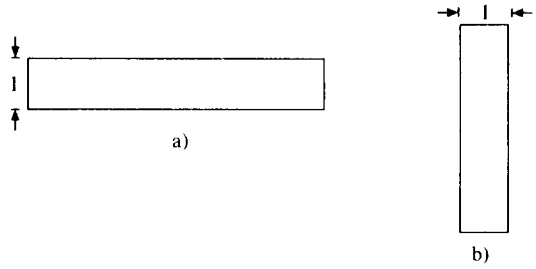


图8-18 放置r_i的可能方式

如若不然，那么总的高度一定大于

$$h+(2B+1)c_i-1+1$$
$$=h+(2B+1)c_i$$

对于某个i，在这种情况下，总的面积将大于

$$
\begin{aligned}
&(h+(2B+1)c_i)w \\
&=hw+2Bc_i w+wc_i \\
&=(2Bc_i+h)w+wc_i \\
&\geqslant A+wc_i \\
&>A,
\end{aligned}
$$

这是不可能的。

(3) n个矩形所占据的行总数不大于B，如果大于B，那么面积将大于

$$w(h+2B)=A$$

这是不可能的。

基于上面的讨论，我们很容易证明

$$\sum_{k=1}^{j_i} d_{i_k} \leqslant C$$

换句话说，我们能将对应于行为i矩形的物品放入装箱i中而不超过装箱的容积。因此，这样完成了证明。

例8-16 简单多边形的艺术陈列馆问题

在第3章提到的艺术陈列馆问题定义如下：已知一个艺术陈列馆，需配置最小数量的警卫，使艺术陈列馆的每点至少有一名警卫监视。假设艺术陈列馆以简单多边形表示。因此，艺术陈列馆问题也可定义如下：*已知一个简单多边形，在其中放置最小数量的警卫，使得简单多边形的每点至少有一名警卫监视*。例如，考虑图8-19中的简单多边形，至少需要三名警卫。

如果规定每名警卫仅放置在简单多边形的顶点上，那么称这个特殊的艺术陈列馆问题为最小顶点警卫问题(minimum vertex guard problem)。现在证明最小顶点警卫问题是NP难的。艺术陈列馆问题的NP难度可以类似地证明。首先定义顶点警卫判定问题如下：*已知有n个顶点的简单多边形P，以及一个正整数$K<n$，确定是否存在子集$T\subseteq V$，并且$|T|\leqslant K$，以致在T中每个顶点放一名警卫使P中每个顶点被至少T中一名警卫监视到*。

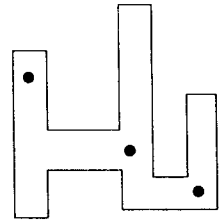


图8-19 简单多边形和最小数目的警卫i_i

很容易明白顶点警卫判定问题是NP的，因为非确定性算法只需猜测有K个顶点的子集$V'\subseteq V$，并在多项式时间内验证是否在V'中的每个顶点放一名警卫，使得在多边形的每点能被那名警卫监视。为了证明该判定问题的NP完全性，将使用NP完全的3可满足性（3SAT）问题。

我们将证明3SAT问题是多项式可规约到顶点警卫判定问题的。首先定义可区分点如下：*在多边形中两个不同点是可区分的，如果它们不能从多边形中的任何点可见*。这个定义在后面的讨论中需要。为了从给定的3SAT问题实例构造一个简单多边形，首先分别定义对应于文字、子句和变量的基本多边形。这些基本多边形可组合成最后转换的简单多边形。

文字多边形

对于每个文字，构造多边形如图8-20所示。在图8-20中，点记号“.”表示一个可区分的点，后面将会解释，这些文字多边形以这种方式安排，将在最后的简单多边形中没有点能被两者看到。容易看到只有a_1和a_3能

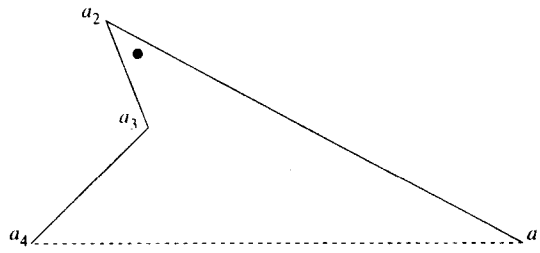


图8-20 文字模式子多边形

看到此完整的文字多边形。在一个3SAT问题实例中，每个子句有三个文字。所以，对应于子句的每个多边形将有三个这样的文字多边形。

子句多边形

对于每个子句$C_i = A \vee B \vee D$，构造一个子句多边形如图8-21所示。令$(a_1, a_2, \cdots, a_n)$表示$a_1, a_2, \cdots, a_n$在同一条直线上。那么，在图8-21中，我们有(g_{h8}, g_{h4}, g_{h5})，$(g_{h3}, g_{h4}, g_{h7}, g_{h1})$，$(g_{h2}, g_{h8}, a_{h4}, a_{h1}, b_{h4}, b_{h1}, d_{h4}, d_{h1}, g_{h9}, g_{h1})$和$(g_{h9}, g_{h7}, g_{h6})$。此外，$|(g_{h2}, g_{h8})| = |(g_{h8}, g_{h4})|$和$|(g_{h1}, g_{h9})| = |(g_{h9}, g_{h1})|$，其中$|(u, v)|$表示线段$(u, v)$的长度。

在子句多边形中有一些重要的性质。首先，容易看到没有g_{hi}，$i = 1, 2, \cdots, 7$能看到任何整个文字多边形。所以，必须在文字多边形内配置警卫。注意在任何文字多边形中没有配置警卫能看到整个完整的另外两个文字多边形。所以，我们需要三名警卫，在每个文字多边形中配置一名警卫。但是在这些文字多边形中有某些顶点没有配置警卫。在后续的内容中，将证明从$\{a_{h1}, a_{h3}, b_{h1}, b_{h3}, d_{h1}, d_{h3}\}$中配置警卫，只有七种候选组合。推理如下：

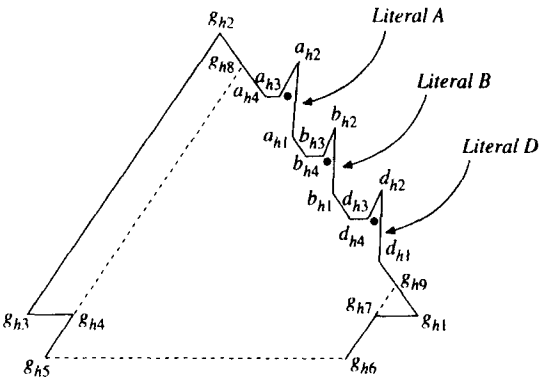


图8-21 子句结$C_i = A \vee B \vee D$

(1) a_{h4}、b_{h4}和d_{h4}中没有一个能看到三个完整的文字多边形，所以它们不能是候选。

(2) 考虑文字A的文字多边形。如果选择a_{h2}，那么无论如何对文字B和D的文字多边形配置，$\Delta a_{h1}a_{h3}a_{h4}$不能被任何警卫看到。因此，$a_{h2}$应被取消。也就是，只应从$\{a_{h1}, a_{h3}, b_{h1}, b_{h3}, d_{h1}, d_{h3}\}$中选择警卫。

(3) 没有两个顶点从同一文字多边形中选择。因此，从$\{a_{h1}, a_{h3}, b_{h1}, b_{h3}, d_{h1}, d_{h3}\}$中选择的顶点只有八种可能的组合。因为$\{a_{h3}, b_{h3}, d_{h3}\}$不能看到整个$\Delta g_{h1}g_{h2}g_{h3}$，所以应该取消。这意味着只有七种可能的组合，现总结如下.

性质1：只有从$\{a_{h1}, a_{h3}, b_{h1}, b_{h3}, d_{h1}, d_{h3}\}$中选择顶点的七种组合能看到整个子句多边形。这些顶点的七种组合是$\{a_{h1}, b_{h1}, d_{h1}\}$，$\{a_{h1}, b_{h1}, d_{h3}\}$，$\{a_{h1}, b_{h3}, d_{h1}\}$，$\{a_{h1}, b_{h3}, d_{h3}\}$，$\{a_{h3}, b_{h1}, d_{h1}\}$，$\{a_{h3}, b_{h1}, d_{h3}\}$和$\{a_{h3}, b_{h3}, d_{h1}\}$。

对子句多边形的标记机制1

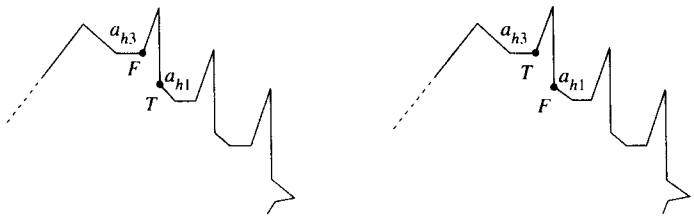
需要注意，每个文字多边形对应一个文字。在子句中文字以肯定或否定出现。然而，从上面对文字多边形构造的讨论，我们似乎没有考虑文字的符号。这有些迷惑。实际上，后面将看到，文字的符号将确定该文字一些多边形的顶点标记，这样通过考虑顶点的符号，对变量真值的指派将确定警卫在哪里配置。

考虑文字A，其他情况是类似的。对于文字A，必须在a_{h1}或a_{h2}配置一名警卫。如果A是肯定（否定），那么标记顶点$a_{h1}(a_{h3})$为T，顶点$a_{h1}(a_{h1})$为F，参见图8-22。如果变量指派为真（假），那么在标记为$T(F)$的顶点配置警卫。这意味着顶点$a_{h1}(a_{h3})$表示对文字A的指派为真（假）。

现在用一个例子解释标记机制、真值指派和最终警卫配置之间的关系。考虑情况$C_h = u_1 \vee -u_2 \vee u_3$，图8-23显示对此情况的标记。

考虑满足指派，比如

$$u_1 \leftarrow T,\ u_2 \leftarrow F,\ u_3 \leftarrow T$$



a) 文字$A = u_i$　　b) 文字$A = -u_i$

图8-22 对子句交点的标记机制1

在此情况下，警卫将配置在a_{h1}、b_{h1}和d_{h1}。以此种方式配置警卫将不能看到整个子句多边形。考虑另一种满足指派，比如

$$u_1 \leftarrow T,\ u_2 \leftarrow T,\ u_3 \leftarrow F$$

在此情况下，警卫将配置在a_{h1}、b_{h3}和d_{h3}。以此种方式配置警卫将能看到整个子句多边形。最后，考虑下面的不可满足性指派：

$$u_1 \leftarrow F,\ u_2 \leftarrow T,\ u_3 \leftarrow F$$

在此情况下，警卫将配置在a_{h3}、b_{h3}和d_{h3}。以此种方式配置警卫将不能看到整个子句多边形。通过仔细地验证标记机制1，可以看到从顶点$\{a_{h1}, a_{h3}, b_{h1}, b_{h3}, d_{h1}, d_{h3}\}$的八种组合中只有$\{a_{h3}, b_{h3}, d_{h3}\}$符合不可满足的真值指派在子句中出现。所以，有下面的性质：

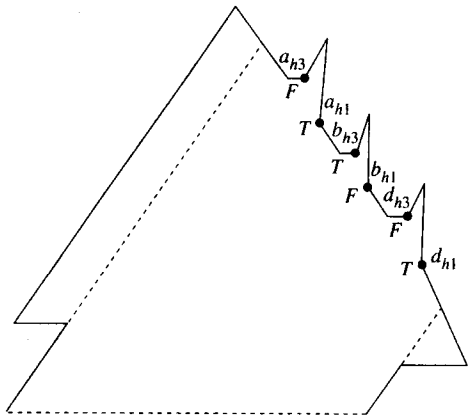


图8-23 对子句交点$C_h = u_1 \vee -u_2 \vee u_3$的标记机制1的例子

性质2：子句C_h的子句多边形从文字多边形的三个顶点可见，当且仅当对应顶点标记的真值构成对子句C_h的可满足性指派。

变量多边形

对每个变量构造一个变量多边形，如图8-24所示。我们有$(t_{i3}, t_{i5}, t_{i6}, t_{i8})$。在$\Delta t_{i1}t_{i2}t_{i3}$中存在可区分的点，注意$\Delta t_{i1}t_{i2}t_{i3}$只能从$t_{i1}$，$t_{i2}$，$t_{i3}$，$t_{i5}$，$t_{i6}$和$t_{i8}$可见。

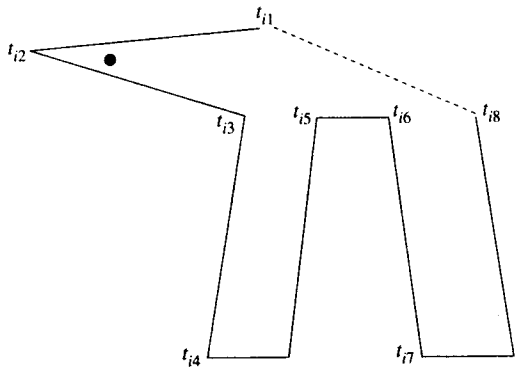


图8-24 u_i的变量多边形

我们已描述了子句多边形和变量多边形。现在描述如何合并它们成为问题实例多边形。

问题实例多边形的构造

步骤1. 将变量多边形和子句多边形放在一起，如图8-25所示。有下面的事实：

(1) 顶点w除了$\Delta t_{i1}t_{i2}t_{i3}$外，可见$n$个变量多边形，其中$i = 1, 2, \cdots, n$。

(2) $(w, g_{15}, g_{16}, g_{25}, g_{26}, \cdots, g_{m5}, g_{m6})$和$(t_{11}, g_{h5}, g_{h4}, g_{h8})$是可满足的，其中$h = 1, 2, \cdots, m$。

步骤2. 用尖峰增加变量多边形。假设变量u_i出现在子句C_h中，如果u_i在子句C_h中肯定出现，那么如图8-26a所示的两个尖峰$\overline{pq}$和$\overline{rs}$使得(a_{h1}, t_{i5}, p, q)和(a_{h3}, t_{i8}, r, s)可满足。u_i在子句C_h是否定出现的情况显示在图8-26b中。

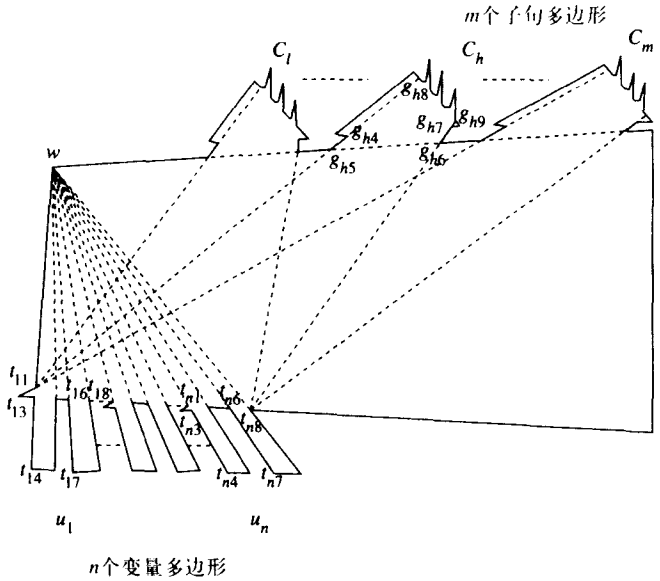


图8-25 合并变量多边形和子句多边形

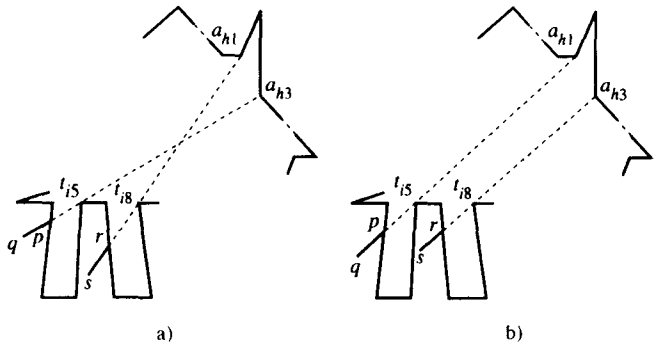


图8-26 增加尖峰

步骤3. 用多边形代替尖峰。由于增加的尖峰 $\overline{pq}$ 和 $\overline{rs}$ 是线段，在步骤2完成后没有得到简单的多边形，所以按如下方法用多边形代替尖峰。我们将在图8-26 a中解释这种情况，图8-26b是类似的。在图8-26a中，用阴影区域代替 $\overline{pq}$ 和 $\overline{rs}$ 如图8-27a所示，得到(a_{h1}, t_{i5}, p, q)，(a_{h1}, e, f)，(a_{h3}, t_{i8}, r, s)和(a_{h3}, u, v)。也就是，我们得到$\Delta a_{h1}qf$ 和$\Delta a_{h3}sv$。在图8-27中所产生的多边形称为一致性检验多边形（consistency-check polygon）。一致性检验的含义将在后面解释。

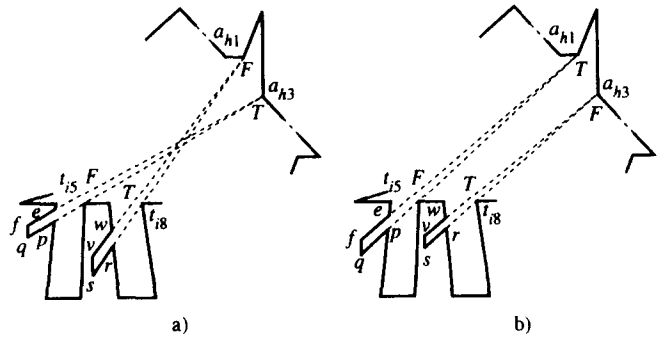


图8-27 通过称为一致性检验模式的小区域代替每个尖峰

变量多边形标记机制2

对于变量多边形u_i，顶点t_{i5}和t_{i8}总是分别标记F和T，如图8-27所示。如果u_i指派真（假），那么在标记为$T(F)$的顶点配置一名警卫。

再次考虑图8-27a。在此情况下，$u_i \in C_h$，有两种配置方法，一名警卫配置在文字多边形，一名警卫配置在变量多边形。

$$F = (u_1 \vee u_2 \vee -u_3) \wedge (-u_1 \vee -u_2 \vee u_3)$$

它们是$\{a_{h1}, t_{i8}\}$和$\{a_{h3}, t_{i5}\}$。第一种情况对应于给u_i指派真，第二种情况对应于给u_i指派假。

图8-28中是从3SAT公式$F = (u_1 \vee u_2 \vee -u_3) \wedge (-u_1 \vee -u_2 \vee u_3)$中构造完全简单多边形。圆点表示警卫。在此情况下，最小警卫数为10。根据标记机制1和2，警卫的配置模式对应于

$$u_1 \leftarrow T,\ u_2 \leftarrow F,\ u_3 \leftarrow T$$

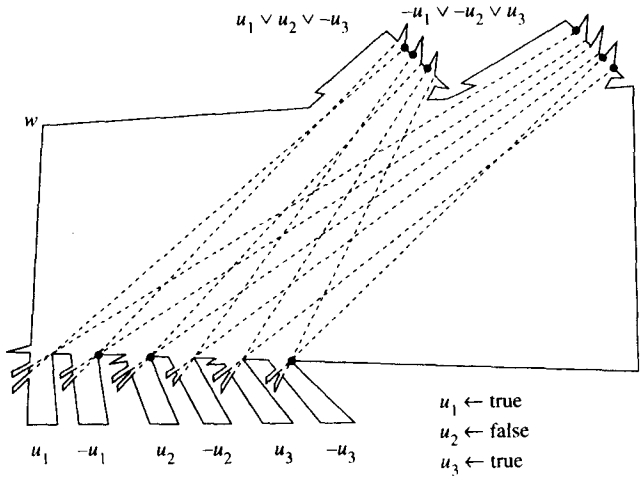


图8-28 由3SAT公式构造的简单多边形的例子

现在我们证明下面主要的结论。

断言1：已知子句集S，如果S是可满足的，那么需要监视整个问题实例S的多边形最小的顶点数是$K = 3m + n + 1$。

可进行如下推理：如果S是可满足的，那么在S中出现的变量有真值指派，使得在这种指派下每个子句C_h为真。如果u_i指派为真，那么在相对应u_i的变量多边形t_{i8}配置一名警卫，在相对应文字多边形的a_{h1}或a_{h3}配置一名警卫，这要分别依赖于u_i还是$-u_i$在C_h中。如果u_i指派为假，那么在相对应u_i的变量多边形t_{i5}配置一名警卫，在相对应文字多边形的a_{h1}或a_{h3}配置一名警卫，这要分别依赖于$-u_i$还是u_i在C_h中。因此，对于通过一致性检验多边形和文字多边形定义的多边形，根据标记机制1和2，需要配置$3m + n$名警卫来监视。对于通过变量多边形定义的剩余矩形，我们只需要在顶点w放置一位警卫来监视。所以，我们需要$K = 3m + n + 1$。

断言2：已知子句集S，如果最小警卫数为$K = 3m + n + 1$，那么S是可满足的。

这部分的证明比较复杂。首先断言w是必须选的；否则，由于在n个变量多边形中，$2n$个矩形中的每一个必须配置警卫，所以至少需要$3m + n$个顶点。因此，下面只考虑剩余的$K-1 = 3m + n$个顶点。

在构造简单多边形时，$3m$个文字多边形中的每个都有可区分的点，n个变量多边形也有。因此，在简单多边形中有$3m + n$个可区分的点。我们知道没有监视可区分点的顶点能监视任何其他可区分点。那么，至少需要$3m + n$个顶点来监视$3m + n$个可区分点。

我们不能任意选择$3m+n$个点，否则，它们不能监视剩余的简单多边形。下述实际上将证明它们必须选择，以致对应于满足S的变量的真值指派。将这样的警卫放置模式称为是一致的，它的定义如下：

假如u_i是变量，u或肯定或否定出现在C_1，C_2，…，C_a中，x_1，x_2，…，x_a是从u在C_1，C_2，…，C_a中的出现对应于文字多边形的顶点，X_i是$\{x_1, x_2, \cdots, x_a\}$。那么，如果下面的条件可满足，$u_i$的变量多边形相对于$X_i$是一致的：

(1) 连接到它的两个矩形之一的所有一致检验多边形能从在X_i中的顶点可见。

(2) 在所有来自同一集合X_i，连接到另一个矩形的那些都是不可见的。

否则，u_i的变量多边形相对于X_i是不一致的。

例如，考虑图8-29。假定变量u_i分别以肯定(u_i)、否定($-u_i$)和肯定(u_i)出现在子句C_h、C_p和C_q中。如果选择顶点x_{h1}，y_{p3}和z_{q1}，由于剩余的矩形能从这三个变量监视，并且右侧的矩形根本监视不到，所以变量多边形相对于这个顶点集是一致的。类似地，对应u_i的顶点集指派真，它相对于x_{h3}，y_{p2}和z_{p3}是一致的。对应u_i顶点集指派假，它相对于x_{h3}，y_{p1}和z_{q1}是不一致的。对于变量u_i的变量多边形相对于X_i一致，那么X_i对应于对u_i的某个真值指派，如果它相对于X_i不一致，那么X_i不对应u_i的任何真值指派。

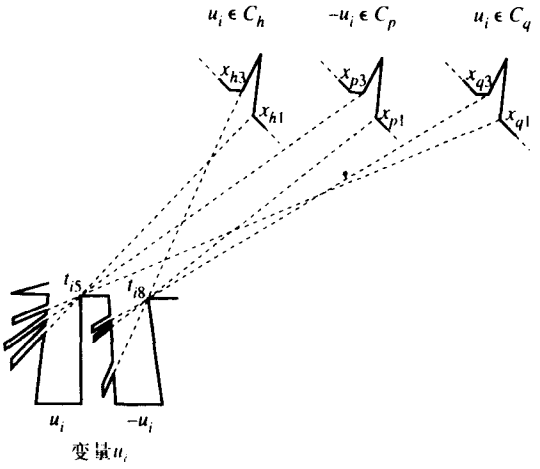


图8-29 一致性概念的举例

如果只有$K=3m+n$个顶点，那么所有的变量多边形一定与文字多边形中的顶点集是一致的。在U_i的任何变量多边形L_i，一致检验多边形的数量依赖于u_i出现在子句中的数量。假定这个数是q，由于连接到L_i左边矩形的一致性验证多边形的数量和右边多边形的数量是一样的，所以在L_i中一致性验证多边形的总数量是$2q$。其中q个连接到一个矩形，或者左面或者右面。如果是左（右），那么我们将警卫配置到$t_{i8}(t_{i5})$。因此，仅需要$3m+n+1$名警卫。如果L_i不可满足，那么连接到左面矩形的一些一致性验证多边形与连接到右面矩形的一些一致性验证多边形就不能从任何顶点监视。所以，至少需要$3m+n+2$个顶点。因此，$K=3m+n+1$要求所有的变量多边形对于文字多边形中选择的顶点是一致的。

由于每个u_i与X_i是一致的，那么连接到它的矩形之一的一致性验证多边形可从X_i中的顶点看到。对于另一个矩形，一致性验证多边形或者在t_{i8}或者在t_{i5}可看到。此外，t_{i8}或者t_{i5}也可视为变量多边形的可区分点。这意味着只需要一个顶点监视每个变量多边形，那么需要n个顶点监视n个变量多边形。为此，得到所有m个完整的子句多边形，一定可以从文字多边形选择的$3m+n+1-(n+1)=3m$个顶点看到。根据性质2，每个子句的三个顶点将对应于出现在该子句变量的真值指派。因此，C是可满足的。

8.8 2可满足性问题

在本章中，我们证明了可满足性问题是NP完全的，也声明了当一个问题是NP完全的，它的变种不必也是NP完全的。为了强调这一点，我们将证明2可满足性问题（2-satisfiablility problem）在P类中，这相当令人惊讶。

2可满足性问题（2SAT）是可满足性问题的特例，在每个子句中只有两个文字。

已知2SAT问题实例，可以构造一个有向图，称为问题实例的指派图（assignment graph）。指派图构造如下：如果在问题实例中x_1，x_2，…，x_n是变量，那么指派图的结点是x_1，x_2，…，x_n和$-x_1$，$-x_2$，…，$-x_n$。如果有子句是$L_1 \vee L_2$形式，其中L_1和L_2是文字，那么在指派图中，从$-L_1$到L_2有一条边，从$-L_2$到L_1也有一条边。由于每条边相关一个且仅有一个子句，我们可用对应的子句标记每条边。

考虑下面的2SAT问题：

$-x_1 \quad \vee \quad x_2$ (1)

$x_2 \quad \vee \quad x_3$ (2)

上面2SAT问题的指派图如图8-30所示。

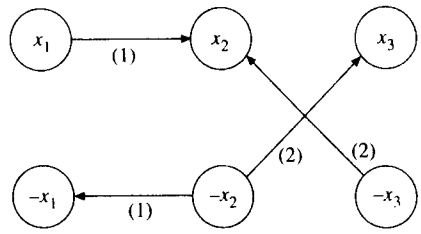


图8-30 指派图

考虑子句(1) $-x_1 \quad \vee \quad x_2$

假定指派x_1为T，为了使得句子为T，那么必须指派x_2也为T，在指派图中由从x_1到x_2的边表明。另一方面，假定$-x_2$指派为T，为了使得句子为T，那么必须指派$-x_1$也为T。在指派图中由从$-x_2$到$-x_1$的边表明。这就是为什么每条子句有两条边，这两条边是对称的。也就是，如果从结点x到结点y有一条边，那么从$-y$到$-x$也会有一条边。每条边对应满足该子句的可能指派。

基于上面的推理，有下面对指派图中的顶点指派真值的规则：

(1) 如果结点A指派为T，那么所有从结点A可达的结点都应指派为T。

(2) 如果结点A指派为T，那么结点$-A$同时指派为F。

例如，考虑图8-30。如果指派$-x_2$为T，那么必须指派$-x_1$和x_3都为T。同时，必须对其他所有结点指派为F，如图8-31所示。一种以上能满足所有子句的指派图指派真值的方法是可能的。例如，对于图8-30，可以指派$-x_3$和x_2为T，x_1为F。很容易明白这两个($-x_1$，$-x_2$，x_3)和($-x_1$，x_2，$-x_3$)都满足子句。

因为有一条边邻接结点A，当且仅当变量A在子句中出现，在任何时候结点A指派为T，所有相对于邻接结点A的边将都是满足的。例如，假定在图8-30中结点x_2指派为T，由于边邻接结点x_2对应于子句1和2，因此两个子句现在都是满足的。

考虑另一个例子，假定子句是

$-x_1 \quad \vee \quad x_2$ (1)

$x_1 \quad \vee \quad x_2$ (2)

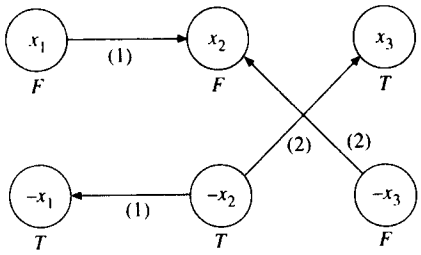


图8-31 对指派图指派真值

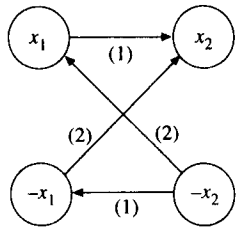


图8-32 指派图

对应上面的子句集的指派图如图8-32所示。在本例中，不能指派$-x_2$为T。如果那样的化，会遇到两个问题：

(1) x_1和$-x_1$都将指派为T，因为x_1和$-x_1$都是从$-x_2$可达的，这是不允许的。

(2) 由于$-x_2$指派为T，那么x_1将指派为T。这最终导致x_2指派为T，这也是不允许的，因为不能对$-x_2$和x_2都指派为T。

但我们可以对x_2指派为T，满足如图8-32所示的子句1和2。因为x_1可指派任意值。在关心可满足性的地方，对x_2指派为T后，x_1的指派不再重要。

考虑下面的例子：

$$x_1 \quad \vee \quad x_2 \tag{1}$$

$$-x_1 \quad \vee \quad x_2 \tag{2}$$

$$x_1 \quad \vee \quad -x_2 \tag{3}$$

$$-x_1 \quad \vee \quad -x_2 \tag{4}$$

对应的指派图如图8-33所示。

在本例中很容易看到对于图中结点没有指派真值的方法。每个指派都导致矛盾。假如指派x_1为T，那么有从x_1到$-x_1$的路径，这导致$-x_1$指派为T。假如指派$-x_1$为T，由于从$-x_1$到x_1也有路径，致使x_1指派为T。因此，不能对x_1指派任何真值。

没有办法对此图中的结点指派真值，这决非偶然。不能这么做是因为该子句集是不可满足的。

一般地，2SAT子句集是可满足的当且仅当对某个x，它的指派图中不能同时包含从x到$-x$的路径和从$-x$到x的路径。读者很容易了解图8-33的指派图含有这样的路径，而在图8-31和图8-32不包含这样的路径。

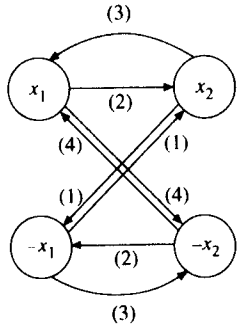


图8-33 对应一个不可满足子句集的指派图

假定在指派图中某个x，有从x到$-x$的路径和从$-x$到x的路径。显然，不能对x指派真值，因此这个输入子句一定是不可满足的。

假定在指派图中所有x不存在从x到$-x$的路径和从$-x$到x的路径。将证明这样的输入子句集一定是可满足的。在这种情况下推理，可以对指派图中的结点成功指派真值。下面给出用于发现这样指派的算法。

步骤1. 在指派图中选择结点A，它没有指派真值，并且不能通向$-A$。

步骤2. 给A和所有从结点A可达的结点指派真值T，对每个结点x指派为T，结点$-x$指派为F。

步骤3. 如果有结点没有指派，转向步骤1；否则，退出。

仍有一点必须注意。上面的算法对于某个x，是否导致x和$-x$都指派为T的情况？这是不可能的。如果结点y通向结点x，由于指派图的对称性，那么一定有从$-x$到$-y$的边。因此，y是这样的结点，将通向$-y$。所图8-34所示，这在步骤1是不被选择的。

因为对于所有的x，从x到$-x$的路径和从$-x$到x的路径不同时存在。在算法中，总是有结点可选。换句话说，上面的算法总是可以产生满足输入子句集的指派。因此，子句集是可满足的。

已经说明为了确定2SAT问题实例的可满足性，只需要对某个x确定，对应的指派图是否包含从x到$-x$的路径和从$-x$到x的路径。这等价于确定在指派图中是否存在$x\to\cdots\to-x\to\cdots x$形式的环。直到目前，我们没有讨论如何设计该算法。对读者来说，设计确定在指派图中是否存在这样的环将变得很容易。

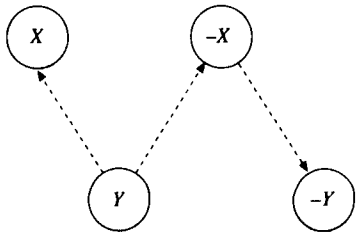


图8-34 指派图中边的对称性

总之，2SAT问题是一个P问题。

8.9 注释与参考

在计算机科学领域中，NP完全性的概念可能是最难理解的概念之一，涉及此概念的最重要的论文是Cook在1971年撰写的，它证明了可满足性问题的重要意义。在1972年，Karp证明了许多组合问题是NP完全的（Karp，1972）。这两篇论文被认为是里程碑式的。从那时起，大量问题被证明是NP完全的。由于有大量这方面的论文，所以由AT&T的David Johnson维护这方面的论文数据库。他也定期为关于NP完全性的算法学报撰写文章。他的文章都以“NP完全性专栏：持续发展指南”为题，这些文章对算法感兴趣的计算机科学家是非常重要和有意义的。

完全地专心致志于NP完全性方面非常好的书是Garey and Johnson(1979)。该书介绍了NP完全性的发展历史，一直作为NP完全性方面的百科全书。在本书中，我们对于库克定理没有给出正式的证明，这是一个痛苦的决断。我们认识到即使给出正式的证明，对于读者也是很难掌握其真正含义的。取而代之，我们通过例子说明非确定型算法如何转换成布尔公式，使得当且仅当转换的布尔公式可满足，那么非确定型算法以“yes”终止。库克定理的形式证明可在最初的库克论文（Cook，1971）中找到。读者还可以参考：Horowitz and Sahni (1978)、Garey and Johnson(1979)以及Mandrioli and Ghezzi(1987)。

注意对于NP完全性只是最坏情况下的分析。如果一个问题是NP完全的，那么并不劝导人们研究在平均情况下解决该问题的有效算法。例如，开发了许多解决可满足性问题的算法，它们中许多是基于本章讨论的消解原理。消解原理是由Robinson所发明（Robinson，1965），并在文献Chang and Lee(1973)中讨论。最近，找到了几个在平均情况下是多项式的可满足性问题算法（Chao，1985；Chao and Franco，1986；Franco，1984；Hu，Tang and Lee，1992；Purdom and Brown，1985）。

另一个有名的NP完全问题是旅行商问题，可通过归约哈密顿回路问题到旅行商问题来建立其NP完全性（Karp，1972）。有一本专门介绍旅行商问题的书（Lawler，Lenstra，Rinnooy Kan and Shmoys，1985）。针对简单多边形的艺术陈列馆问题也是NP完全的（Lee and Lin，1986），还有一本完全介绍该问题的定理和算法的书（O'Rourke，1987）。

在文献（Cook，1971）中发现3可满足性问题是NP完全的。色数问题的NP完全性是由文献Karp(1972)所建立。对于子集和、分割和装箱问题的NP完全性可参阅Horowitz and Sahni (1978)。VLSI离散布局问题由Lapaugh(1980)证明是NP完全的。

2可满足性问题由Cook(1971)和Even，Itai and Shamir(1976)证明是多项式时间的问题，该问题在Papadimitriou(1994)中也有讨论。

还有几个有名的问题疑为NP完全的，Garey and Johnson(1978)给出了一个列表。其中之一即线性规划问题，后来由Khachian(1979)证明是多项式算法，可参阅文献Papadimitriou and Steiglitz(1982)。

最后，还有另一个概念称为平均完全的（average complete）。如果一个问题是平均完全的，那么它恰好在平均情况下是困难的。瓷砖问题（tiling problem）（Berger，1966）由文献Levin (1986)证明是平均完全的。Levin(1986)可能是已出版的最短的论文，仅有一页半长，但却是最难领会的论文。如果你不能理解它，不要恼火，很多的计算机科学家都不理解这篇论文。

8.10 进一步的阅读资料

NP完全性是最兴奋的研究主题之一，许多看似容易的问题被发现是NP完全的。我们鼓励读者阅读Johnson在Journal of Algorithm撰写的关于该领域进展的文章。我们推荐下面的论文，许多是最近出版的，而在任何教科书都找不到：Ahuja (1988)；Bodlaender (1988)；Boppana，

Hastad and Zachos (1987); Cai and Meyer (1987); Chang and Du (1988); Chin and Ntafos (1988); Chung and Krishnamoorthy (1994); Colbourn, Kocay and Stinson (1986); Du and Book (1989); Du and Leung (1988); Fellows and Langston (1988); Flajolet and Prodiuger (1986); Friesen and Langston (1986); Homer (1986); Homer and Long (1987); Huynh (1986); Johnson, Yanakakis and Papadimitriou (1988); Kirousis and Papadimitriou (1988); Krivanek and Moravek (1986); Levin (1986); Megido and Supowit (1984); Monien and Sudborough (1988); Murgolo (1987); Papadimitriou (1981); Ramanan, Deogun and Lin (1984); Wu and Sahni (1988); Sarrafzadeh (1987); Tang, Buehrer and Lee (1985); Valiant and Vazirani (1986); Wang and Kuo (1988) and Yannakakis (1989)。

对于更新的成果，可参阅Agrawal, Kayal and Saxena (2004); Berger and Leighton (1998); Bodlaender, Fellows and Hallet (1994); Boros, Crama, Hammer and Saks (1994); Caprara (1997a); Caprara (1997b); Caprara (1999); Feder and Motwani (2002); Ferreira, de Souza and Wakabayashi (2002); Foulds and Graham (1982); Galil, Haber and Yung (1989); Goldberg, Goldberg and Paterson (2001); Goldstein and Waterman (1987); Grove (1995); Hochbaum (1997); Holyer (1981); Kannan and Warnow (1994); Kearney, Hayward and Meijer (1997); Lathrop (1994); Lent and Mahmoud (1996); Lipton (1995); Lyngso and Pedersen (2000); Ma, Li and Zhang(2000); Maier and Storer(1977); Pe'er and Shamir(1998); Pierce and Winfree(2002); Storer(1977); Thomassen(1997); Unger and Moult(1993) and Wareham (1995)。

习题

8.1 确定下面的陈述正确与否。

(1) 如果一个问题是NP完全的，那么在最坏情况下，它不能被任何多项式算法解决。

(2) 如果一个问题是NP完全的，那么在最坏情况下，找不出解决它的任何多项式算法。

(3) 如果一个问题是NP完全的，那么在最坏情况下，将来也是不可能找到解决它的任何多项式算法。

(4) 如果一个问题是NP完全的，那么在平均情况下，不可能找到解决它的多项式算法。

(5) 如果能证明一个NP完全问题的下界是指数的，那么就可以证明NP≠P。

8.2 确定下面每个子句集的可满足性。

(1) $-X_1 \vee -X_2 \vee X_3$

X_1

$X_2 \vee X_3$

$-X_3$

(2) $X_1 \vee X_2 \vee X_3$

$-X_1 \vee X_2 \vee X_3$

$X_1 \vee -X_2 \vee X_3$

$X_1 \vee X_2 \vee -X_3$

$-X_1 \vee -X_2 \vee X_3$

$X_1 \vee -X_2 \vee -X_3$

$-X_1 \vee X_2 \vee -X_3$

$-X_1 \vee -X_2 \vee -X_3$

(3) $-X_1 \vee X_2 \vee X_3$

$X_1 \vee X_2$

$$
(4)\quad \begin{array}{ccccc} X_3 & & & & \\ X_1 & \vee & X_2 & \vee & X_3 \end{array}
$$

$$
(5)\quad \begin{array}{ccc} X_1 & & \\ X_2 & & \\ -X_1 & \vee & X_2 \\ -X_2 & \vee & X_3 \\ -X_3 & & \end{array}
$$

8.3 我们都知道如何证明一个问题是NP完全的，但如何证明一个问题不是NP完全的呢？

8.4 完成在本章描述过的严格覆盖问题的NP完全性证明。

8.5 完成在本章描述过的子集和问题的NP完全性证明。

8.6 考虑下面的问题：已知两个输入变量a和b，如果$a>b$，那么返回“YES”；否则返回“NO”。设计解决该问题的非确定型多项式算法。并将该问题转换为一个布尔公式，使得当且仅当转换来的布尔公式可满足，那么该算法返回“YES”。

8.7 最大团判定问题（maximal clique decision problem）：最大团是一个图的最大完全子图。最大团的规模是在其中的顶点数。团判定问题是对于某个k，确定是否存在最小规模为k的最大团。通过将可满足性问题归约到它，证明最大团判定问题是NP完全的。

8.8 顶点覆盖判定问题（vertex cover decision problem）：一个图的顶点集S是该图的顶点覆盖，当且仅当该图所有的边至少邻接S中的一个顶点。顶点覆盖判定问题是确定一个图是否有最坏的k个顶点的顶点覆盖。证明顶点覆盖判定问题是NP完全的。

8.9 旅行商判定问题（traveling salesman decision problem）：通过证明哈密顿回路问题可多项式归约到旅行商判定问题，证明该问题是NP完全的。哈密顿回路判定问题的定义可在关于算法的任何教科书中找到。

8.10 独立集判定问题（independent set decision problem）：已知图G和整数k，独立集判定问题是确定是否存在k个顶点的集合S，使得S中没有两个顶点是通过一条边连通的。证明独立集判定问题是NP完全的。

8.11 瓶颈旅行商判定问题（bottleneck traveling salesman decision problem）：已知一个图及一个数M，瓶颈旅行商判定问题是确定在该图中是否存在一个哈密顿回路，使得该回路的最长边小于M。证明瓶颈旅行商判定问题是NP完全的。

8.12 证明8可着色$\propto$4可着色$\propto k$可着色。

8.13 单调子句可满足性问题（clause-monotone satisfiability problem）：如果一个公式的每个子只包含正变量或者负变量，那么这个公式是单调的。例如，

$$F = (X_1 \vee X_2)\ \&\ (-X_3)\ \&\ (-X_2 \vee -X_3)$$

是一个单调公式。证明判定一个单调公式是否可满足的问题是NP完全的。

8.14 对于8维匹配问题（8-dimensional matching problem）的NP完全性，阅读文献Papadimitriou and Steiglitz (1982)中的定理15.7。

第 9 章 近似算法

在前面各章所介绍的每个算法都给出最优解。显然我们要为此付出代价：产生这些最优解消耗的时间可能会非常长。如果问题是NP完全问题，那么产生最优解的算法过程将会有一个不现实的时间消耗。

一种折衷方法是使用启发式的解。词语“启发式”可以解释为“有根据的推测”。因此，启发式算法将会是一种基于有根据的推测的算法。因为是启发式推测算法，所以不能保证它会产生最优解。事实上，通常也不会产生最优解。

本章将介绍几个启发式算法。当然这些启发式算法不能保证产生最优解，只能保证由非最优解产生的误差可以预先估计。所有这样的算法称为近似算法（approximation algorithms）。

9.1 顶点覆盖问题的近似算法

已知图$G = (V, E)$，如果图中每条边与V中一个顶点集S的某个顶点邻接，那么称S为G的一个顶点覆盖（node cover）。一个顶点覆盖S的规模是S中顶点的数目。顶点覆盖问题（node cover problem）是找出$G = (V, E)$的最小顶点覆盖。例如，$\{v_1, v_2, v_4\}$是如图9-1所示的顶点覆盖，$\{v_2, v_5\}$是最小规模的顶点覆盖。

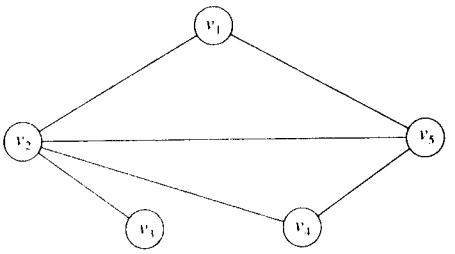


图9-1 一个例图

顶点覆盖问题已被证明是NP难的。对该问题介绍一个近似算法，即算法9-1。

算法9-1 顶点覆盖问题的近似算法

输入：图$G = (V, E)$。
输出：G的一个顶点覆盖S。
步骤1. 令$S = \phi$及$E' = E$。
步骤2. while $E' \neq \phi$
　　从E'中任意选取一条边(a, b)。
　　令$S = S \cup \{a, b\}$，E''是E'与a或b邻接的边集合，并令$E' = E' - E''$。
　end while
步骤3. 输出S。

现在考虑如图9-1所示的图G。算法9-1中的步骤1$S = \phi$，$E' = \{(v_1, v_2), (v_1, v_5), (v_2, v_3), (v_2, v_4), (v_2, v_5), (v_4, v_5)\}$。由于$E' \neq \phi$，执行步骤2中的循环。假如此次从$E'$中挑选边$(v_2, v_3)$，那么在循环结束时，有$S = \{v_2, v_3\}$，$E' = \{(v_1, v_5), (v_4, v_5)\}$。接下来，由于$E' \neq \phi$，执行步骤2中的第二次循环。假如此次从$E'$中挑选出边$(v_1, v_5)$，那么在第二次循环结束时，有$S = \{v_1, v_2, v_3, v_5\}$，$E' = \phi$。最终，输出$G$的顶点覆盖为$\{v_1, v_2, v_3, v_5\}$。

上述算法的时间复杂度是$O(|E|)$。将会看到算法9-1找出G的顶点覆盖的规模至多是G的最小顶点覆盖规模的两倍。令$M*$表示G的最小规模的顶点覆盖，M表示由算法9-1找出的G的顶点覆盖的规模，L表示由算法选出的边总数。由于在步骤2的每一次循环中挑出一条边，且每

条边都关联两个顶点，所以有$M = 2L$。由于在步骤2中不可能选出两条边共享任何一个端顶点，所以至少需要L个顶点覆盖这L条边，那么有$L \leqslant M*$。最终，$M = 2L \leqslant 2M*$。

9.2 欧几里得旅行商问题的近似算法

欧几里得旅行商问题（ETSP）是找出通过平面上n点集合S的最短闭回路，这个问题已被证明是NP难的。因此，不太可能存在有效的最坏情况下欧几里得旅行商问题的算法。

在本节将描述欧几里得旅行商问题的近似算法，找出一条是最优解3/2倍范围内的回路。就是说，如果最优回路长度为L，那么该近似回路的长度不会超过$(3/2)L$。

近似方案的基本思想是构造一个顶点集的欧拉回路，然后用它来寻找近似回路。一个图的欧拉回路是图的一个回路，在该回路中对图的每个顶点的访问不必是一次，而对图的每条边的访问只能是一次。当图包含一条欧拉回路时，称该图为欧拉图。欧拉证明了图G是欧拉图，当且仅当G是连通的，并且G中所有顶点的度数均为偶数。考虑下面的图9-2，显然图9-2a所示的图是一个欧拉图，而图9-2b不是。稍后将会看到找到欧拉回路之后，只需通过简单地遍历欧拉图，并且绕过之前访问过的顶点，就可以找到一条哈密顿回路。

因此，构造一个点集S的连通图G（也就是，使用S中的点作为G中的顶点），使得G中每个顶点的度都是偶数，就可以构造一条欧拉回路，然后找到一条哈密顿回路来近似最优旅行商回路。

近似方案由以下三步组成：首先，构造S的一棵最小生成树。例如，考虑图9-3。对于有8个点的点集$S_1 = \{P_1, P_2, \cdots, P_8\}$，$S_1$的一棵最小生成树如图9-3所示。

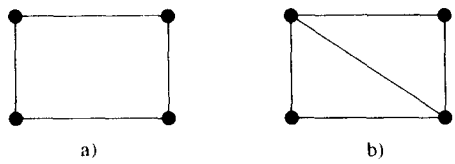


图9-2 有欧拉回路与没有欧拉回路的图

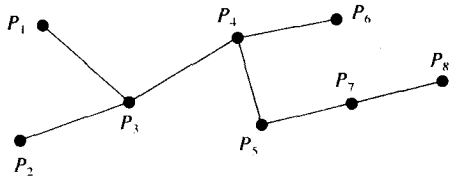


图9-3 8个点的最小生成树

尽管最小生成树是连通的，但它可能包含奇数度的顶点。事实上，可以证明在任何树中，总有偶数个这样的顶点。例如，在图9-3中的$\{P_1, P_2, P_3, P_4, P_6, P_8\}$点集中的每一个顶点的度都是奇数。第二步是构造一个欧拉图，也就是必须加入更多的边使得所有的顶点都是偶数度。这个问题变成了如何找到这样的边集？在该算法中，对奇数度的点集使用最小欧几里得加权匹配。已知平面中的一个点集，最小欧几里得加权匹配问题（minimum weighted matching problem）通过线段成对地添加点，使得总长度最小。如果可以找到奇数度顶点的最小欧几里得加权匹配，扩充匹配的边集到最小生成树的边集中，那么产生的图是一个欧拉图。例如，顶点P_1, P_2, P_3, P_4, P_6和P_8的最小欧几里得加权匹配如图9-4所示。

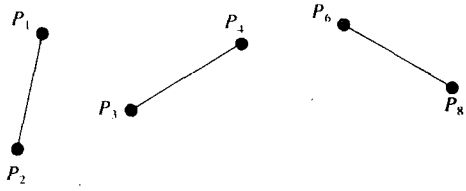


图9-4 6个顶点的最小欧几里得加权匹配

在把图9-4中的边加入图9-3中的最小生成树之后，所有顶点的度数都是偶数的。算法的第三步在结果图外构造一条欧拉回路，然后由欧拉回路得到哈密顿回路。上述8个顶点集的欧拉回路是$P_1-P_2-P_3-P_4-P_5-P_7-P_8-P_6-P_4-P_3-P_1$（注意$P_3$和$P_4$遍历了两次），如图9-5a所示。绕过$P_4$和$P_3$，直接将$P_6$与$P_1$相连，得到一条哈密顿回路。因此，得到的近似回路是$P_1-P_2-P_3-P_4-P_5-P_7-P_8-P_6-P_1$，如图9-5b所示。

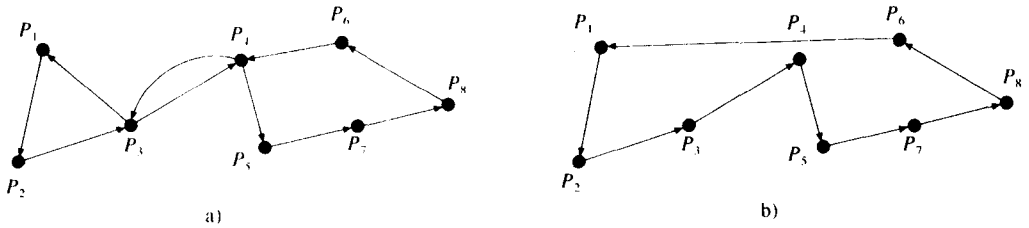


图9-5 一个欧拉回路以及结果近似回路

算法9-2 ETSP的近似算法

输入：平面上n个点的集合S。

输出：S的一个近似旅行商回路。

步骤1. 找出S的最小生成树T。

步骤2. 在T中的奇数度顶点集上找到最小欧几里得加权匹配M，令$G = M \cup T$。

步骤3. 找到G的欧拉回路，然后通过绕过所有已经遍历过的顶点，通过遍历该欧拉回路找到哈密顿回路作为ETSP的近似路径。

对该算法的时间复杂度分析是容易的。步骤1可以在$O(n \log n)$时间内执行完，而在$O(n^3)$时间内可以找到最小欧拉加权匹配，欧拉回路可以在线性时间内构造，由该欧拉回路得到哈密顿回路也可以在随后的线性时间内得到。因此，上述算法的时间复杂度是$O(n^3)$。因为在此讨论的主要目标是说明近似算法，所以不给出寻找最小匹配和欧拉回路算法的具体细节。

令L为S的最优旅行商回路。如果从L中移走一条边，得到路径L_p，它同样是S的一棵生成树。令T表示S的一棵最小生成树，由于T是S的最小生成树，那么有$\text{length}(T) \leqslant \text{length}(L_p) \leqslant \text{length}(L)$。令$\{i_1, i_2, \cdots, i_{2m}\}$为$T$中奇数度的顶点下标的集合，它们在集合中安排成与在最优旅行商回路L中的次序一样。考虑奇数度顶点的两个匹配$M_1 = \{[i_1, i_2], [i_3, i_4], \cdots, [i_{2m-1}, i_{2m}]\}$和$M_2 = \{[i_2, i_3], [i_4, i_5], \cdots, [i_{2m}, i_1]\}$。由于边满足三角不等式，显然有$\text{length}(L) \geqslant \text{length}(M_1) + \text{length}(M_2)$。令$M$为$\{i_1, i_2, \cdots, i_{2m}\}$的最优欧几里得加权匹配。因此，两个匹配长度较短的比$\text{length}(M)$的长度长，所以有$\text{length}(M) \leqslant \text{length}(L)/2$。由于$G = T \cup M$，那么$\text{length}(G) = \text{length}(T) + \text{length}(M) \leqslant \text{length}(L) + \text{length}(L)/2 \leqslant 3/2\text{length}(L)$。

由于上面的解是哈密顿回路，比G的长度短，因此，有一个欧拉旅行商问题的近似算法，产生一个近似的回路，在时间$O(n^3)$内最优解的3/2倍范围之内。

9.3 特殊瓶颈旅行商问题的近似算法

旅行商问题也可以定义在图上：已知一个图，要求从某个顶点开始，遍历图中所有的顶点并回到开始顶点，找出这样的最短闭合回路。该问题已被证明是NP难的。

在本节将考虑另一种类型的旅行商问题。仍然关心闭合回路。然而，不要求整个回路的长度最短，而是要求整个回路的最长边最短。也就是说，要找出一条最长边为最短的回路。因此这是一个最小最大问题（mini-max problem），称这种类型的旅行商问题为瓶颈旅行商问题（bottleneck salesperson problem），因为它通常属于与运输学问题相关的一类瓶颈问题。事实上，可以看到，之前讨论的许多问题可以修改成为瓶颈问题。例如，最小生成树问题有一个瓶颈问题的变形。在这个变形中，将找到其最长边为最短的生成树。

再次如同前两节一样，这里将给出一个近似算法。在介绍该算法之前，声明以下的假设：

(1) 研究的图是一个完全图（complete graph）。也就是，每一对顶点之间都有一条边。

(2) 所有的边都遵守三角不等式规则。因此，瓶颈旅行商问题是一种特例，可以证明这种

特殊瓶颈旅行商问题是NP难的。

在介绍近似算法之前，提出一个不是多项式级的算法。该算法将产生瓶颈旅行商问题的最优解。然后，将该算法修改为近似算法。

首先把图$G = (V, E)$中的边排序成非递减序列$|e_1| \leqslant |e_2| \leqslant \cdots \leqslant |e_m|$。令$G(e_i)$表示从$G = (V, E)$中删除所有比$e_i$长的边而得到的图。考虑$G(e_1)$，如果$G(e_1)$包含一条哈密尔顿回路，那么该哈密顿回路就是所要找的最优解；否则，考虑$G(e_2)$。重复此过程直到某个$G(e_i)$包含了一条哈密顿回路，然后返回该哈密顿回路作为特殊瓶颈旅行商问题的最优解。

用一个例子解释上面的思想。参见图9-6所示。为了找到一条有最长边为最短的哈密顿回路，可以尝试从$G(AC)$开始，AC的长度是12。因此，如图9-7所示，$G(AC)$仅包含那些长度小于或等于12的边。在$G(AC)$中没有哈密顿回路。尝试$G(BD)$，因为BD是下一条最长边。$G(BD)$如图9-8所示，在$G(BD)$中有一条哈密顿回路，也就是A-B-D-C-E-F-G-A。可以推断找出了最优解，以及最优解的值是13。

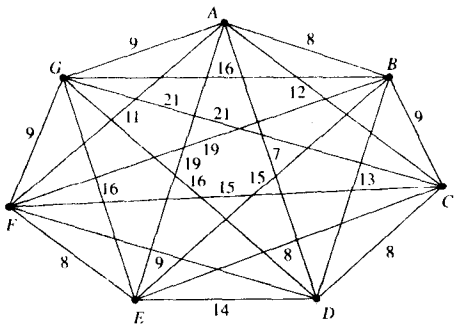


图9-6 一个完全图

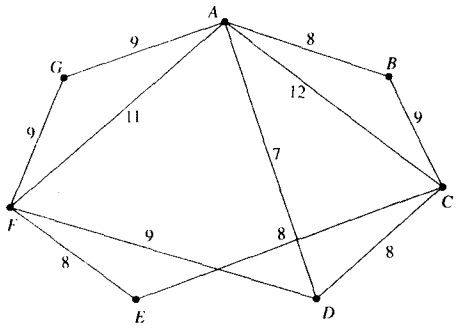


图9-7 图9-6的$G(AC)$

由于寻找哈密尔顿回路是一个NP难的问题，所以该算法不是多项式级的。下面将介绍一个近似算法该近似算法仍具有上述算法的思想。在介绍这个近似算法之前，首先介绍双连通（biconnectedness）的概念。图A是双连通的，当且仅当其每一对顶点至少属于一个公共的简单回路。

例如，在图9-9中，G_b是双连通的，而G_a不是。因为顶点A和顶点C不在任何简单回路中。另一方面，在G_b中，每一对顶点都属于至少一个公共简单回路中。接下来定义图的幂（power of a graph）概念。如果$G = (V, E)$是一个任意图，t是一个正整数，令图G的t次幂表示为$G^t = (V, E^t)$，其中在G^t中有一条边(u, v)，无论何时在图G中都有一条从u到v至多t条边的路径。

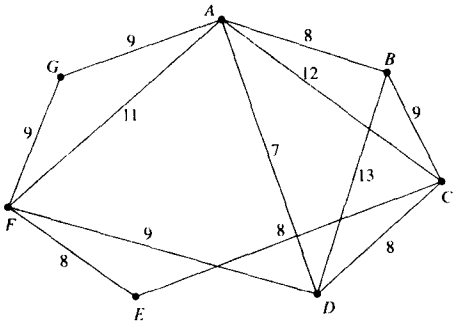


图9-8 图9-6的$G(BD)$

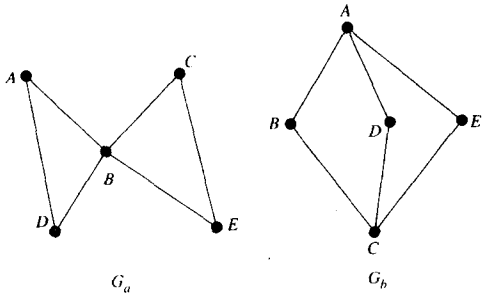


图9-9 双连通的例子

双连通图有一个非常重要的性质：如果图G是双连通的，那么G^2必有一条哈密顿回路。

显然，在图9-10是双连通的。例如，顶点D和F在同一个简单回路D-C-F-E-D中。现在考虑图9-11所示的G^2。在图9-11的图中有一条哈密顿回路，即A-B-C-D-F-E-G-A。

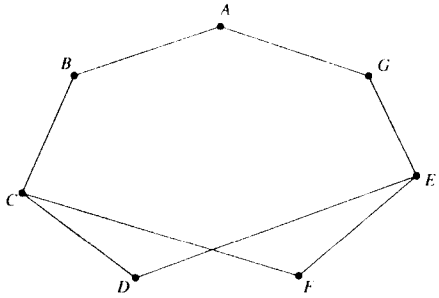


图9-10 一个双连通图

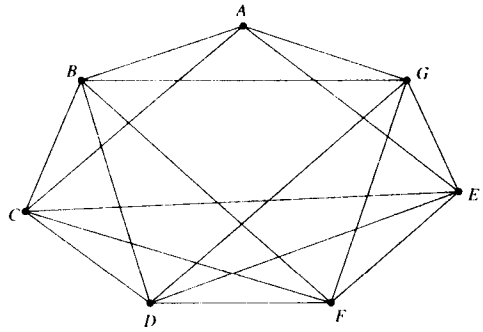


图9-11 图9-10所示图的G^2

基于此性质，每个双连通图G，G^2都有一条哈密顿回路，可以用下面的方法给出瓶颈旅行商问题的近似解。如前述，将G的边排序为非递减序列：$|e_1|\leqslant|e_2|\leqslant\cdots\leqslant|e_m|$。令$G(e_i)$只包含长度小于或等于边$e_i$长度的边。从$G(e_1)$开始考虑，如果$G(e_1)$是双连通的，那么构造$G(e_1)^2$；否则，考虑$G(e_2)$。重复此过程直到找到第一个使$G(e_i)$是双连通的$i$。然后构造$G(e_i)^2$。该图必定包含一条哈密顿回路，可将此哈密顿回路作为近似解。

近似算法正式描述如下：

算法9-3 特殊瓶颈旅行商问题的近似算法

输入：完全图$G=(V,E)$，其中所有的边都满足三角不等式。

输出：G中的一条回路，其最长的一条边不大于G的特殊瓶颈旅行商问题最优解值的两倍。

步骤1. 将边排序为$|e_1|\leqslant|e_2|\leqslant\cdots\leqslant|e_m|$。

步骤2. $i:=1$。

步骤3. 如果$G(e_i)$是双连通的，那么构造$G(e_i)^2$，并且寻找$G(e_i)^2$中的一条哈密顿回路，将其作为输出返回；否则，转向步骤4。

步骤4. $i:=i+1$，转向步骤3。

现在通过一个例子来说明该算法。参见图9-12所示，它表明了$G(FE)(|FE|=8)$，图G如图9-6所示。$G(FE)$不是双连通的。因为$|FG|=9$，所以考虑$G(FG)$。如图9-13所示$G(FG)$是双连通的，然后构造$G(FG)^2$，如图9-14所示。在这个图中有一条哈密顿回路，即A-G-E-F-D-C-B-A。在这条回路中，最长边的长度是16。必须注意到这并不是最优解，由于在图9-6中的瓶颈旅行商问题的最优值是13，它比16小。

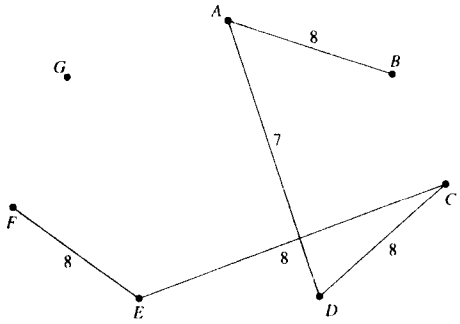


图9-12 图9-6的$G(FE)$

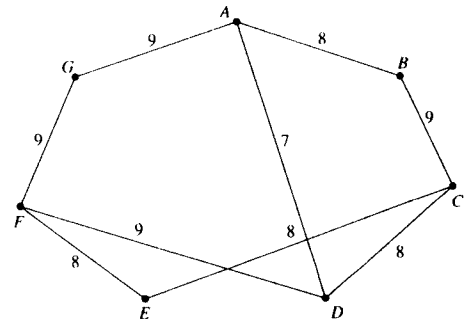


图9-13 图9-6的$G(FG)$

下面讨论与近似算法相关的三个问题：

(1) 近似算法的时间复杂度是多少？

(2) 近似解的界是什么？

(3) 在特定意义下，该界是最优的吗？

首先回答第一个问题。双连通性的判定可以通过多项式算法求解。进而，如果G是双连通的，那么存在一个解决G^2的哈密顿问题的多项式算法。因此，该近似算法是多项式级的。

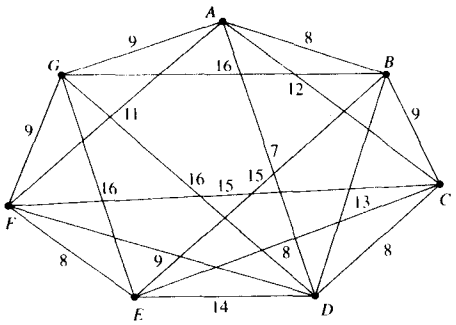


图9-14 $G(FG)^2$

现在回答第二个问题：近似解的界是什么？考虑下面的问题：$G=(V,E)$的边子图是G的具有顶点集V的子图。在G的所有边子图中，找出一个双连通的最长边为最短的边子图。显然，用上面的近似算法找到的第一个$G(e_i)$是双连通的，它必定是该问题的一个解。令e_{op}为特殊瓶颈旅行商问题最优解的最长边。显然，由于任意旅行商问题的解都是双连通的，故有$|e_i|\leqslant|e_{op}|$。在找到$G(e_i)$之后，构造$G(e_i)^2$。由于所有的边都服从三角不等式，所以$G(e_i)^2$中最长边的长度不大于$2|e_i|$。因而由近似算法产生的最优解的值不大于$|e_i|\leqslant 2|e_{op}|$。也就是，虽然近似算法并没有产生最优解，但是其解的值仍然限定在最优解的两倍范围之内。

下面通过所给的例子来检验上述的结论。在上面的例子中，$|e_{op}|=13$。由近似算法产生的解是16。由于$16\leqslant 2\cdot 13=26$，所以在这种情况下，结论是正确的。

最后，回答第三个问题：该近似解的界是最优的吗？即，是否有另一个多项式近似算法产生的解的界小于2倍？例如，对于所有的问题实例，一个多项式近似算法产生最长边小于或等于1.5倍最优解的近似解，这可能吗？

现在将说明这很可能是不可能的。如果有这样的近似算法，并且是多项式级的，那么可以用这个算法解决NP完全问题。就是说，如果有一个产生的界小于两倍最优解的多项式近似算法，那么有NP = P。我们使用的特殊NP完全问题是哈密顿回路决策问题（Hamiltonian cycle decision problem）。

假设对于特殊瓶颈旅行商问题存在多项式算法A，使得A确保对于每条边满足三角不等式的完全图G_C产生值小于$2v*$的解，其中$v*$是G_C的特殊瓶颈旅行商问题最优解的值。对于任意图$G=(V,E)$，可以将G变换为一个完全图G_C，并且定义G_C的边值为

$$C_{ij}=\begin{cases}1 & \text{如果}(i,j)\in E\\ 2 & \text{否则}\end{cases}$$

显然，上面c_{ij}的定义满足三角不等式。现在来解决G_C的瓶颈旅行商问题。显然

$$v^*=\begin{cases}1 & \text{如果}G\text{是哈密顿图}\\ 2 & \text{否则}\end{cases}$$

等价地，$v^*=1$当且仅当G是哈密顿图。

也就是说，如果能够解决G_C的瓶颈旅行商问题，那么可以解决G的哈密顿回路决策问题。简单地检测v^*的值，G是哈密顿图当且仅当$v^*=1$。下面将说明虽然事实上算法A仅是瓶颈旅行商问题的近似算法，但可以使用A来解决哈密顿回路问题。

根据假设，算法A可以产生一个值小于$2v^*$的近似解。考虑G是哈密顿图的情况，那么近

似算法A将产生一个值V_A小于$2v^* = 2$的近似解。

也就是，如果G是哈密顿图，由于边的权值只有1和2，那么$V_A = 1$。因此，认为可以使用算法A解决G的哈密顿回路决策问题。如果A是多项式级的，因为哈密顿回路决策问题是NP完全问题，那么有NP = P。

上面的讨论说明存在这样的算法是不大可能的。

9.4 特殊瓶颈加权k供应商问题的近似算法

特殊瓶颈加权k供应商问题（special bottleneck weighted k-supplier problem）是近似算法的另一个例子。在该问题中，已知一个所有边均满足三角不等式的完全图，边的权值可看作是距离，所有顶点分在两个集合中，供应商集合V_{sup}以及消费者集合V_{cust}。此外，每个供应商顶点i有一个权值w_i，表示在这个顶点建立供应中心的代价。选择一个总权值最多为k的供应商集合，使得供应商与消费者之间的最长距离为最短。

考虑图9-15，有9个顶点，即$V = \{A, B, C, \cdots, I\}$，$V_{sup} = \{A, B, C, D\}$及$V_{cust} = \{E, F, G, H, I\}$。供应商$A$, B, C和D的代价分别是1, 2, 3和4。注意在图9-15中显示一个完全图，在图9-15中所有消失的边的代价比图9-15中的最大代价还大，但是所有边的代价值仍满足三角不等式。

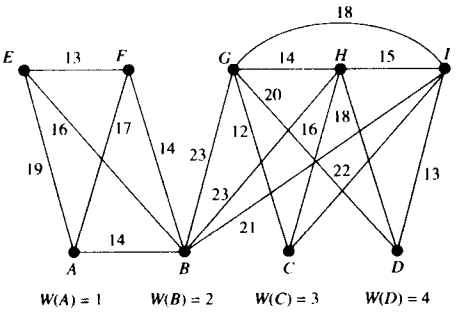


图9-15 一个加权k的供应商问题实例

假设$k = 5$。图9-16显示了几个可行解，最优解如图9-16d所示，其解的值为20。

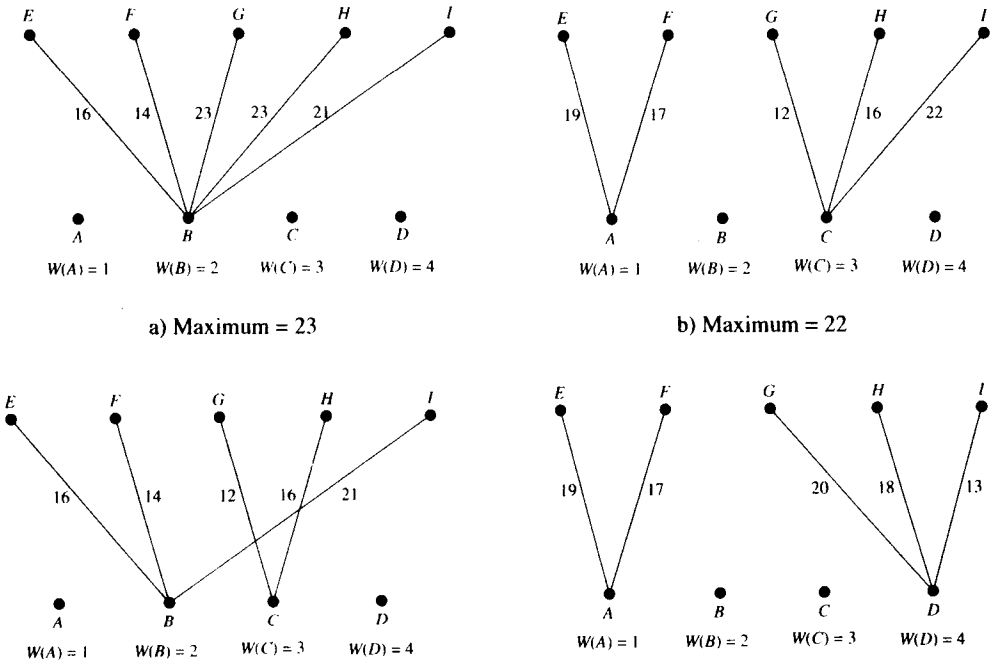


图9-16 图9-15问题实例的几个可行解决方案

将要介绍的近似算法与9.3节用来解决特殊瓶颈旅行商问题的近似算法是相似的。为求解该特殊瓶颈加权k供应商问题，将边排序为非递减序列：$|e_1|\leqslant|e_2|\leqslant\cdots\leqslant|e_m|$。正如之前所做的，令$G(e_i)$表示从$G=(V,E)$中删除所有比$e_i$长的边而得到的图。理想地，该算法将测试$G(e_1)$，$G(e_2)$，…，直到遇到第一个$G(e_i)$，使得$G(e_i)$包含加权$k$供应商问题的一个可行解，这个可行解一定是最优的。遗憾的是，如我们容易想到的，确定一个图是否包含加权k供应商问题的可行解是一个NP完全问题。因此，必须修改这个方法来得到一个近似算法。

在近似算法中，将使用一个测试过程。此测试过程满足下面的性质：

(1) 如果测试返回“YES”，那么输出一个归纳解作为近似算法的解。

(2) 如果测试返回“NO”，那么可以确定$G(e_i)$不含任何可行解。

如果测试过程没有满足上面的条件，并且图中的边满足三角不等式关系，那么可以很容易证明近似算法将会产生一个近似解，该近似解的值小于或等于最优解值的三倍。

令V_{op}和V_{ap}分别表示最优解和近似解的值。根据性质（2），有$|e_i|\leqslant V_{op}$。

此外，由性质（1），原因将在后面解释，得到

$V_{ap}\leqslant 3|e_i|$

因此，有$V_{ap}\leqslant 3V_{op}$。

现在解释测试如何。想象任意包含可行解的$G(e_i)$。每一个消费者顶点必须至少与一个供应商顶点连接，如图9-17所示，v_1，v_2，v_3和v_4是供应商顶点，所有其他的顶点为消费者顶点。每一个v_i都有几个消费者顶点与它相连。考虑图9-17所示图$G(e_i)$的$G(e_i)^2$。

$G(e_i)^2$如图9-18所示。由于$G(e_i)$包含一个可行解，每个消费者顶点至少与一个供应商顶点相连，$G(e_i)^2$将包含几个团。在一个团中，每一对顶点是连通的。

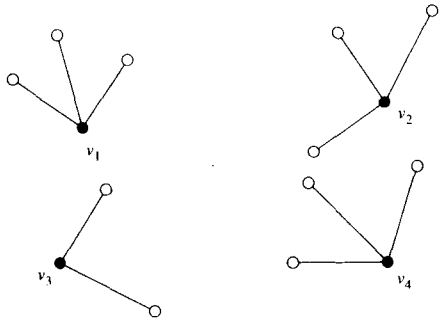


图9-17 包含一个可行解的$G(e_i)$

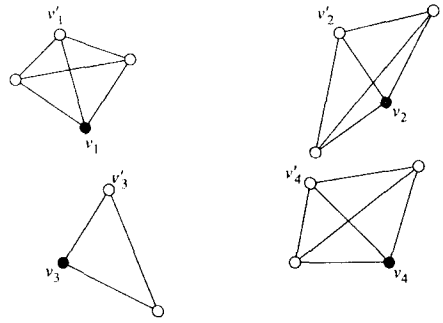


图9-18 图9-17的G^2

可以从每个团中选一个消费者顶点，此结果是一个最大的独立集。图$G=(V,E)$的独立集（independent set）是使E的每条边e至多邻接V'中一个顶点的子集$V'\subset V$。图的一个最大独立集（maximal independent set）是一个不完全包含在任何独立集中的独立集。如图9-18所示，$\{v_1', v_2', v_3', v_4'\}$是一个最大独立集。在找到最大独立集之后，对于每个消费者顶点v_i'，寻找一个与它相连的最小权值的供应商顶点。这个反向建立的供应商集可以证明是$G(e_i)^3$的一个可行解。

测试过程（关于$G(e_i)$和$G(e_i)^2$的）由下面步骤组成：

(1) 如果在V_{cust}中有顶点v，与$G(e_i)$中的任何供应商顶点均不相邻，则返回“NO”。

(2) 否则，在$G(e_i)^2$的V_{cust}中找到一个最大独立集S。

(3) 对于S中的每个顶点v'，找到$G(e_i)$中权值最小的相邻供应商顶点v_i。令S'为这些顶点的集合。

(4) 如果S'中的总权值$w(v_i)$小于或等于k，那么返回“YES”；否则，返回“NO”。

如果测试结果是“NO”，显然$G(e_i)$不包含任何可行解。如果测试结果是“YES”，那么可以使用归纳的供应商集合作为近似算法的解。因此，近似算法如算法9-4所示。

算法9-4 解决特殊瓶颈k供应商问题的近似算法

输入：完全图$G=(V,E)$，其中$V=V_{cust}\cup V_{sup}$，$V_{cust}\cap V_{sup}=\phi$，与$V_{sup}$中每个顶点$v_i$相关的权值$w(v_i)$，$G$中所有的边均满足三角不等式。

输出：特殊瓶颈k供应商问题的近似解，近似解的值不大于最优解的值的3倍。

步骤1. 将E中的边排序为$|e_1|\leqslant|e_2|\leqslant\cdots\leqslant|e_m|$，其中 $m=\binom{n}{2}$。

步骤2. $i:=0$。

步骤3. $i:=i+1$。

步骤4. 如果在V_{cust}中存在某个顶点（是消费者顶点）与$G(e_i)$中的任何供应商顶点均不相邻，转向步骤3。

步骤5. 在$G(e_i)^2$的V_{cust}中找出最大独立集S。对于S中的每个顶点v_j'，找到一个供应商顶点v_j，v_j有最小的权值，并与$G(e_i)$中的v_j'相邻，并令$S'=\bigcup_j v_j$。

步骤6. 如果$\sum w(v_i)>k$，其中$v_j\in S'$，那么转向步骤3。

步骤7. 将S'作为供应商集合输出。

步骤8. 对于V_{cust}中的每个顶点v_j，令DIST(j)表示v_j与S'之间的最短距离。令$V_{ap}=\max_j(\mathrm{DIST}(j))$，并输出$V_{ap}$。

下面通过一个例子说明该算法，参见图9-15。图9-19显示了图9-15中的$G(HI)$。由于H不与$G(HI)$中的任何供应商相邻，所以测试的结果是“NO”。

在图9-20中显示$G(HC)$。由于HC是下一条最短的边，图9-21显示了$G(HC)^2$。假设在$G(HC)^2$的V_{cust}中，选择$S=\{E,G\}$，那么$S'=\{B,C\}$。$W(B)+W(C)=5\leqslant k=5$。因此，$S'=\{B,C\}$是解。从$S'=\{B,C\}$中，可以找到每个消费者顶点的最近顶点，结果如图9-22所示。$DIST(E)=16$，$DIST(F)=14$，$DIST(G)=12$，$DIST(H)=16$，$DIST(I)=21$。因此，$V_{ap}=21$。

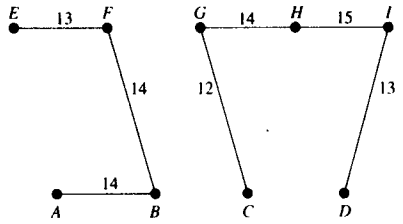


图9-19 图9-15的$G(HI)$

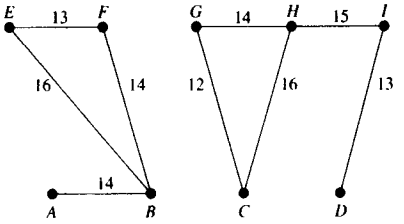


图9-20 图9-15的$G(HC)$

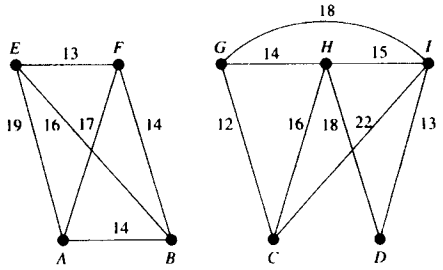


图9-21 $G(HC)^2$

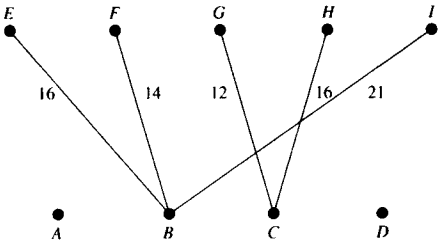


图9-22 由$G(HC)^2$得到的归纳解

正如之前所做的，要回答下面三个问题：

(1) 近似算法的时间复杂度是多少？

(2) 近似解的界是什么？

(3) 该界是最优的吗？

为了回答第一个问题，可以简单地说明该近似算法是由寻找最大独立集需要的步骤决定的，这是一个多项式级的算法。因此，这个近似算法是多项式算法。

现在回答第二个问题。通过使用近似算法得到解值的界是什么？令e_i为近似算法输出的解，V_{op}为特殊瓶颈k供应商问题最优解的值。显然，$|e_i| \leqslant V_{op}$。此外，应注意在近似算法步骤8中得到的近似解的值为V_{ap}。对于S中的每一个消费者顶点v_j，在初始供应商顶点集中找一个最近邻点v_i，这些最近邻点形成S'。必须注意到连接顶点v_j和在S'中它的最近邻点的边不必是在$G(e_i)^2$中的边。例如，考虑图9-22，BI不是$G(HC)^2$中的边。假设用C代替B作为I的供应商顶点。那么应该注意到CI也不是$G(HC)^2$中的边。但是，$V_{ap} = |BI|$必须小于等于$|CI|$。就是说，$|CI|$是$|BI|$的上界。因此，查询CI属于哪个$G(HC)^d$是很重要的。在下面将证明CI属于$G(HC)$、$G(HC)^2$或者$G(HC)^3$。

原因如下：对于算法挑选的不在最大独立集中的每个消费者顶点v_j，令v_j^m为最大独立集中直接与v_j连接的顶点。这样的v_j^m必定是存在的；否则，最大独立集就不是最大独立集了。如图9-23所示，v_j^m必须是一个消费者顶点，并且与算法步骤5中确定的S'中的供应商顶点v_i直接相连。现在，边(v_j, v_j^m)必定在$G(e_i)$中，或在$G(e_i)^2$中。此外，如步骤5所规定的，(v_j^m, v_j)必须在$G(e_i)$中。因此，边(v_i, v_j)必定在$G(e_i)$、$G(e_i)^2$或者$G(e_i)^3$中。也就是说，如果$d \geqslant 3$，边(v_i, v_j)在$G(e_i)^d$中。在近似算法的最后，v_j可能与某一v_i'连接，而不是v_i。(v_i', v_j)必定比(v_i, v_j)短；否则，就不会挑选v_i'。因此，

$$V_{ap} = |(v_i', v_j)| \leqslant |(v_i, v_j)| \leqslant 3|e_i| \leqslant 3V_{op}$$

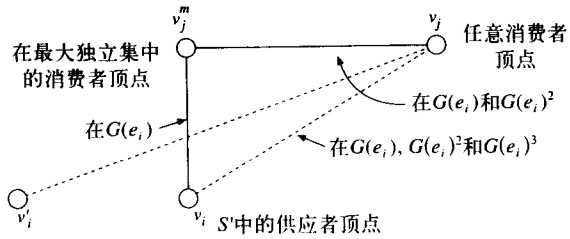


图9-23 特殊瓶颈k供应商问题近似算法界的说明

最后的问题是关心这个限界系数3的最优性。将证明要降低这个限界系数是不大可能的。如果可能，那么NP = P。本质上，要证明，如果存在可以产生特殊瓶颈k供应商问题近似解的多项式近似算法D，其解的值可以确保小于$a \cdot V_{ap}$，其中a<3且V_{op}是最优解的值，那么P = NP。

为证明此点，利用的NP完全问题是寻找集合问题（hitting set problem）。可以证明，如果上述的近似算法D存在，那么可以用算法D解决该寻找集合问题，这隐含着P = NP。

寻找集合问题（hitting set problem）定义如下：已知有限集S和S的子集的合集C以及一个正整数k。是否存在子集$S' \subseteq S$，其中$|S'| \leqslant k$，对于C中的任何S_i有$S' \cap S_i \neq \phi$。

例如，令

$S = \{1, 2, 3, 4, 5\}$

$C = \{\{1, 2, 3\}, \{4, 5\}, \{1, 2, 4\}, \{3, 5\}, \{4, 5\}, \{3, 4\}\}$

以及$k = 3$。

那么$S' = \{1, 3, 4\}$有这样的性质，$|S'| = 3 \leqslant k$，以及对于C中的每个S_i有$S' \cap S_i \neq \phi$。

为了说明此目的，将称寻找集合问题为k寻找集合问题，以强调参数k。

给定一个k寻找集合问题的实例，将这个例子转换一个特殊瓶颈k供应商问题的实例。假

设$S=\{a_1, a_2, \cdots, a_n\}$，那么令$V_{sup}=\{v_1, v_2, \cdots, v_n\}$，其中每个$v_i$对应一个$a_i$。假设$C=\{S_1, S_2, \cdots, S_m\}$，那么令$V_{cust}=\{v_{n+1}, v_{n+2}, \cdots, v_{n+m}\}$，其中$v_{n+i}$对应$S_i$。定义每个供应商顶点的权值为1，$v_i$与$v_j$之间的距离定义如下：

$d_{ij}=2$ 如果$\{v_i, v_j\}\subseteq V_{sup}$，或者$\{v_i, v_j\}\subseteq V_{cust}$

$=3$ 如果$v_i\in V_{sup}$，$v_j\in V_{cust}$，且$a_i\notin S_j$

$=1$ 如果$v_i\in V_{sup}$，$v_j\in V_{cust}$，且$a_i\in S_j$

直接证明这些距离满足三角不等式，以及*k供应商问题有一个代价为1的解，当且仅当存在一个k寻找集合S'*。此外，如果没有k寻找集合，那么最优k供应商的代价必定是3。因此，如果一个近似算法确保近似解V_{ap}，其中$V_{ap}<3V_{op}$，那么$V_{ap}<3$，当且仅当$V_{op}=1$，并且有一个k寻找集合。就是说，可以用这个多项式近似算法解决是NP完全问题的k寻找集合问题，这蕴涵了NP = P。

9.5 装箱问题的近似算法

*已知在$L=\{a_i \mid 1\leqslant i\leqslant n,\ 0<a_i\leqslant 1\}$中的n件物品清单，将它们放在单位容量的若干箱子中，那么，装箱问题就是确定包含所有n件物品的箱子最小的数量。*如果将不同大小的物品看作是在标准处理器上执行不同作业的时间长度，那么该问题变为在固定时间内完成所有作业，所使用最少处理器的问题。例如，令$L=\{0.3, 0.5, 0.8, 0.2, 0.4\}$，如图9-24所示，至少需要三个箱子来装这5件物品。

由于可以方便地将分割问题转化为这个问题，那么装箱问题是NP难的。下面将介绍装箱问题的近似算法。

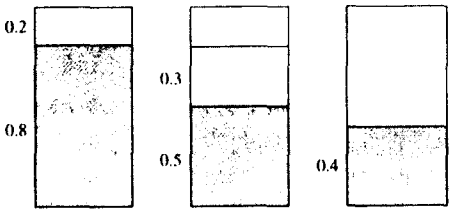


图9-24 装箱问题的例子

近似算法称为最先适应算法(first-fit algorithm)。令所要的箱子标以1，2，…，m下标，B_1表示第一个箱子的容量。最先适应算法将物品放入箱子，以下标递增的方式，一次一件物品。为了放物品a_i，最先适应算法总是将它放入下标最小的箱子中，已经放入该箱子中物品大小的总和不能超过1减去a_i的大小。

最先适应算法是很自然的，并且易于实现。剩下的问题是算法如何近似装箱问题的最优解。

令物品a_i的大小表示为$S(a_i)$。每个箱子有一个单位容量，问题实例I的最优解的大小表示为$OPT(I)$，大于等于所有物品大小之和的上限，即

$$OPT(I)\geqslant\left\lceil\sum_{i=1}^{n}S(a_i)\right\rceil$$

令$FF(I)$为最先适应算法中所需箱子的数量。$FF(I)=m$。$FF(I)$的上界可推导如下：选任意箱子B_i，令$C(B_i)$为放到箱子B_i中a_j的尺寸之和。那么

$$C(B_i)+C(B_{i+1})>1$$

否则，B_{i+1}中的物品会放入B_i中。将所有的m个非空箱子加起来，有

$$C(B_1)+C(B_2)+\cdots+C(B_m)>m/2$$

也就是说

$$FF(I)=m<2\left\lceil\sum_{i=1}^{m}C(B_i)\right\rceil=2\left\lceil\sum_{i=1}^{m}S(a_i)\right\rceil$$

因此，推导出

$FF(I)<2OPT(I)$

9.6 直线m中心问题的最优近似算法

已知平面上n个点的集合，直线m中心问题（rectilinear m-center problem）是寻找覆盖所有这n个点的m个直线方形，使得这些方形的最大边长最短。直线方形是边垂直或平行于欧几里得平面x轴的方形。因为对于直线m中心问题的解中可以扩大较小的方形到最大边长的方形而不影响目标，假设在一个解中所有m个方形的边长相等。边长称为解的大小。

图9-25给出直线5中心问题的例子。用“+”标记中心点被5个直线方形所覆盖。图中所示的5个直线方形形成其最优解。

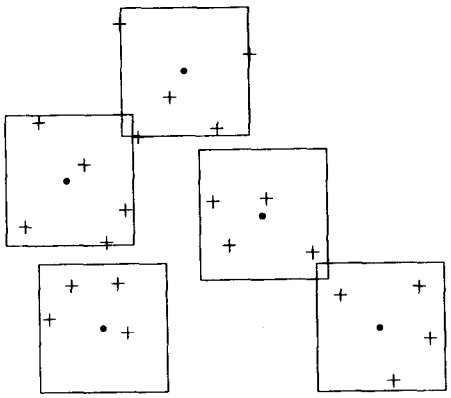


图9-25 直线5中心问题的例子

直线m中心问题已被证明是NP难的。此外已证明，除非NP = P，那么解决该问题的任何多项式时间近似算法的误差比≥2。在本节中，将介绍一个误差比精确地等于2的近似解。

注意，关于$P=\{p_1, p_2, \cdots, p_n\}$的直线$m$中心问题的任意最优解，解的大小必定等于输入点间的直线距离之一。也就是说，最优解的大小必定等于$L_\infty(p_i, p_j)$之一，$1\leqslant i<j\leqslant n$，其中$L_\infty((x_1, y_1), (x_2, y_2)) = \max\{|x_1-x_2|, |y_1-y_2|\}$。如果存在覆盖所有$n$个点，长为$r$的$m$个方形，那么数$r$是可行的。一个解决直线$m$中心问题的直接方法如下：

(1) 计算所有可能的$L_\infty(p_i, p_j)$，$1\leqslant i<j\leqslant n$。

(2) 排序上面计算得到的距离，将其表示为：$D[1]\leqslant D[2]\leqslant\cdots\leqslant D[n(n-1)/2]$。

(3) 对于$D[i]$进行折半查找，$i = 1, 2, \cdots, n(n-1)/2$，找到最小的索引值$i_0$，使得$D[i_0]$是可行的，且$D[i_0]$的大小是可行解。对于查找的每个距离$D[i]$，要测试$D[i]$是否可行。

测试距离仅为$O(\log n)$。然而，对于每个$D[i]$，是否存在覆盖所有n个点具有边长为$D[i]$的m个方形仍然是一个NP完全问题。因此，正如所希望的，至少对于时间来说，上面算法的界是指数级的。

在本节中，将介绍用在近似算法中的“松弛”测试子程序。已知平面上n个点的集合，以及两个输入参数m和r，该松弛测试子程序产生边长为$2r$，而不是r的m个方形。此时，测试是否m个方形覆盖这n个点。如果失败，那么将会返回“失败”。如果成功，那么返回大小为$2r$的可行解。随后将看到“失败”确保没有边长为r的m个方形可以覆盖这n个点。

后面将给出这个子程序的细节。令Test(m, P, r)表示这个松弛子程序。近似算法是以上面具有松弛算法中的简单最优算法来代替测试子程序的算法。

算法9-5 直线m中心问题的近似算法

输入：有n个点的集合P及中心数m。

输出：$SQ[1], \cdots, SQ[m]$：直线m中心问题的一个可行解，该解的规模小于或等于最优解的两倍。

步骤1. 计算所有的点对之间的直线距离，并将其升序排列为$D[0] = 0$，$D[1], \cdots, D[n(n-1)/2]$。

步骤2. $LEFT := 1$，$RIGHT := n(n-1)/2$。

步骤3. $i := \lceil (RIGHT + LEFT)/2 \rceil$。

步骤4. If Test(m, P, $D[i]$)不是“失败”then
　　$RIGHT := i$
　else
　　$LEFT := i$。
步骤5. If $RIGHT = LEFT+1$ then
　　Return Test(m, P, $D[RIGHT]$)
　else
　go to步骤3。

现在描述松弛测试子程序Test(m, P, r)如下：

(1) 在剩余的没有被覆盖的点中找出x值最小的点，以该点为中心画一个边长为$2r$的方形。

(2) 在解中添加这个方形，并且移出所有已经被这个方形覆盖的点。

(3) 将步骤1和步骤2重复m次。如果成功找到覆盖所有输入点的m个方形，那么返回这m个方形作为一个可行解；否则，返回“失败”。

算法9-6　算法Test(m, P, r)

输入：点集P，中心数m，大小r。
输出：“失败”，或者$SQ[1]$，…，$SQ[m]$，覆盖P的大小为$2r$的m个方形。
步骤1. $PS := P$
步骤2. For $i := 1$ to m do
　If $PS \neq \phi$ then
　　$P := PS$中具有最小x值的点
　　$SQ[i] :=$ 中心为p，大小为$2r$的方形
　　$PS := PS-\{$被$SQ[i]$覆盖的点$\}$
　else
　　$SQ[i] := SQ[i-1]$。
步骤3. If $PS = \phi$ *then*
　　return $SQ[1]$，…，$SQ[m]$
　else
　　return “失败”。

再次考虑图9-25中的直线5中心问题。令r为图9-25中方形的边长。对于该实例，对参数r应用松弛测试子程序，先找到x值最小的点p_1，然后画出以边长为$2r$、p_1为中心的方形S_1，如图9-26所示。然后移出被S_1覆盖的所有点。在剩下的点中，找到x值最小的点p_2，再次画出以边长为$2r$、p_2为中心的方形S_2，然后移出被S_2覆盖的所有点，如图9-27所示。将上面的步骤重复5次。最后，得到一个可行解，如图9-28所示。在这个例子中，4个方形就足够了。

接下来证明近似算法的误差率为2。主要声明如下：*如果r是可行的，那么松弛测试子程序*Test(m, P, r)总是会返回一个大小为$2r$的可行解。假设上面的说法是正确的。令$r*$是最优解的大小。由于$r*$是可行的，根据上面的声明，子程序Test(m, P, $r*$)将返回一个可行解。由于近似算法至少会在r终止，这样的Test(m, P, r)返回一个大小为$2r$的可行解，得到$r \leqslant r*$。因此，这个可行解的大小为$2r$，小于等于$2r*$。也就是说，误差率为2。

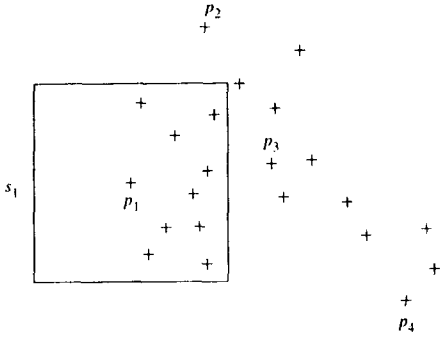


图9-26　松弛测试子程序的第一次应用

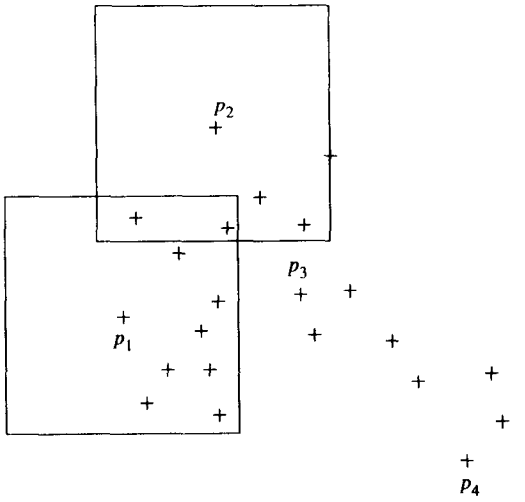


图9-27 松弛测试子程序的第二次应用

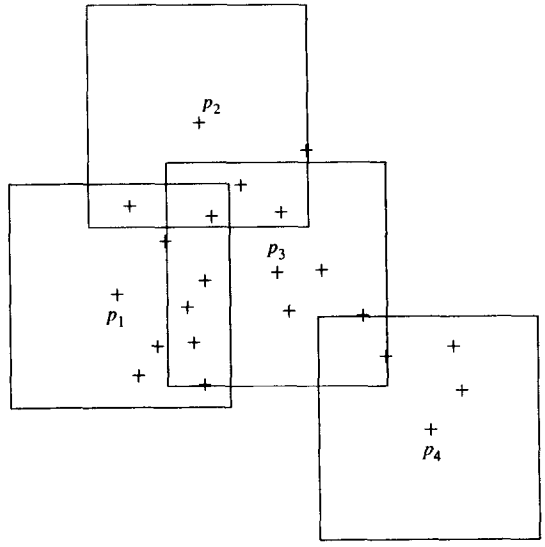


图9-28 直线5中心问题的一个可行解

下面，证明上述的说法。令S_1，S_2，…，S_m为大小为r的可行解，S_1'，…，S_m'为在子程序Test(m，P，r)中产生的边长为$2r$的m个方形。如果S_1'，…，S_m'互不区别，那么在子程序Test的步骤2中$PS = \phi$。也就是说，在这种情况下，S_1'，…，S_m'总是覆盖所有点，并形成一个可行解。下面考虑S_1'，…，S_m'是相互区别的情况。令S_i'的中心为p_i'，$i = 1, 2, \cdots, m$。也就是S_i'中的任意点到p_i'的直线距离都小于等于r。这样，选择S_i'，p_j'不被S_1'，…，S_{j-1}'所覆盖。因此，对于所有$i \neq j$，有$L_\infty(p_i', p_j') > r$。由于对$i = 1, \cdots, m$，S_i的边长为r，S_i中任何两个点之间的直线距离小于等于r。所以，任何S_j至多包含一个p_i'，$i = 1, \cdots, m$。另一方面，由于形成的可行解S_1，S_2，…，S_m的并包含了p_1'，…，p_m'。因此，p_i'属于不同的S_j，也就是$p_i' \in S_i$。对于$1 \leqslant i \leqslant m$，由于$S_i$的大小为$r$，对于所有的$p \in S_i$，有$L_\infty(p, p_j') \leqslant r$。这样，$S_i \subset S_i'$（图9-29）。因此，$S_1'$，…，$S_m'$包含所有的点，可以得出上面的声明。根据上面的说法，也可以有下面的等价描述：

如果Test(m，P，r)没有返回一个可行解，那么r是不可行的。

在算法9-5中，步骤1和步骤2所用的时间为$O(n^2 \log n)$，步骤3执行时间为$O(\log n)$。测试中步骤3到步骤5的循环时间为$O(mn)$。因此，我们得到误差率为2的直线m中心问题的近似算法的时间复杂度为$O(n^2 \log n)$。

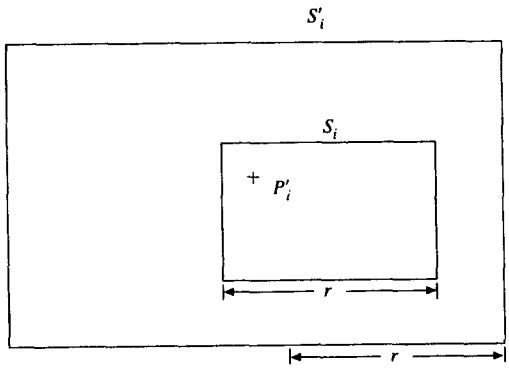


图9-29 $S_i \subset S_i'$的解释

9.7 多序列比对问题的近似算法

在第7章中只研究了两个序列间的比对。很自然地，扩展该问题成为超过2个序列的多序列比对问题（multiple sequence alignment problem）。考虑下面涉及3个序列的情况。

S_1 = ATTCGAT

S_2 = TTGAG

S_3 = ATGCT

这三个序列的一个非常好的比对表示如下：

S_1 = ATTCGAT
S_2 = -TT-GAG
S_3 = AT--GCT

注意，序列中的每一对之间的比对都是非常好的。

多序列比对问题类似于两序列的排序问题。代替基本得分函数$\sigma(x, y)$，必须定义一个包含多个变量的得分函数。假设有三个序列，不是考虑a_i和b_j，现在必须考虑a_i、b_j和c_k。这就是说，必须定义一个$\sigma(x, y, z)$，还必须找到$A(i, j, k)$。当确定了$A(i, j, k)$时，需要考虑下面

$A(i-1, j, k)$，$A(i, j-1, k)$，$A(i, j, k-1)$，$A(i-1, j-1, k)$
$A(i, j-1, k-1)$，$A(i-1, j, k-1)$，$A(i-1, j-1, k-1)$

在2序列间定义一个线性得分函数。已知k个输入序列，成对的多序列比对问题得分总和是找到这k个序列的比对，使得其中所有成对的序列比对得分总和最大。如果输入的序列数量k是可变的，那么这个问题已被证明是一个NP难的。因此，没有太大希望使用多项式算法来解决这个成对的多序列比对问题的总和，所以近似算法是必需的。在本节中，将介绍一个由Gusfield提出的近似算法，来解决成对的多序列比对问题的和。

考虑下面的两个序列：

S_1 = GCCAT
S_2 = GAT

这两个序列之间的一种可能的比对是

S_1' = GCCAT
S_2' = G--AT

对于这个比对，有三个准确的匹配和两个不匹配。如果$x = y$，那么定义$\sigma(x, y) = 0$；如果$x \neq y$，那么定义$\sigma(x, y) = 1$。对于比对$S_1' = a_1', a_2', \cdots, a_n'$和$S_2' = b_1', b_2', \cdots, b_n'$，由该比对导出的两个序列之间的距离定义为

$$\sum_{i=1}^{n} \sigma(a_i', b_i')$$

读者可以容易地理解这个记为$d(S_i, S_j)$的距离函数有下面的特性：

(1) $d(S_i, S_i) = 0$

(2) $d(S_i, S_j) + d(S_i, S_k) \geqslant d(S_j, S_k)$

第二个性质称为三角不等式。

需要强调的另一点是对于2序列比对，比对(-，-)永远不会出现。但是，在多序列比对中，可能有“-”与“-”的匹配。考虑如下的S_1，S_2和S_3：

S_1 = ACTC
S_2 = AC
S_3 = ATCG

首先比对S_1和S_2如下：

S_1 = ACTC

S_2 = A--C

然后假设比对S_1和S_3如下：

S_1 = ACTC-
S_3 = A-TCG

三个序列最终比对如下：

S_1 = ACTC-
S_2 = A--C-
S_3 = A-TCG

这一次，“-”与“-”匹配了两次。当S_3与S_1比对时，“-”添加到已经比对的S_1中。

已知两个序列S_i和S_j，最小的比对距离表示为$D(S_i, S_j)$。现在考虑下面的四个序列。首先找到与其他所有序列的最短距离序列

S_1 = ATGCTC
S_2 = AGAGC
S_3 = TTCTG
S_4 = ATTGCATGC

将这四个序列按对进行比对。

S_1 = ATGCTC

$D(S_1, S_2) = 3$

S_2 = A-GAGC

S_1 = ATGCTC

$D(S_1, S_3) = 3$

S_3 = TT-CTG

S_1 = AT-GC-T-C

$D(S_1, S_4) = 3$

S_4 = ATTGCATGC

S_2 = AGAGC

$D(S_2, S_3) = 5$

S_3 = TTCTG

S_2 = A--G-A-GC

$D(S_2, S_4) = 4$

S_4 = ATTGCATGC

S_3 = -TT-C-TG-

$D(S_3, S_4) = 4$

S_4 = ATTGCATGC

然后得到

$$D(S_1, S_2) + D(S_1, S_3) + D(S_1, S_4) = 3 + 3 + 3 = 9$$
$$D(S_2, S_1) + D(S_2, S_3) + D(S_2, S_4) = 3 + 5 + 4 = 12$$
$$D(S_3, S_1) + D(S_3, S_2) + D(S_3, S_4) = 3 + 5 + 4 = 12$$

$D(S_4, S_1) + D(S_4, S_2) + D(S_4, S_3) = 3 + 4 + 4 = 11$

可以看出S_1相对于其他所有序列具有最短距离，称S_1与其他所有序列最相似，称这个序列为序列的中心（center of sequencen）。下面正式地定义这个概念。

已知有k个序列的集合S，序列集的中心是使下面最小的序列。

$$\sum_{X \in S \setminus \{S_i\}} D(S_i, X)$$

共有$k(k-1)/2$对序列，每一对通过动态规划方法比对，显然找到中心花费多项式时间，近似算法如算法9-7所示。

算法9-7　寻找成对的多序列比对问题总和近似解的2近似算法

输入：k个序列。
输出：性能比不大于2的k序列比对。
步骤1. 找到这k个序列的中心。不失一般性，可以假设S_1是中心。
步骤2. 令$i = 2$。
步骤3. While $i \leqslant k$
　　在S_i和S_1间找到最优比对。
　　必要的话，在已比对的序列S_1，S_2，…，S_{i-1}中添加空格。
　　$i = i + 1$
　End while
步骤4. 输出最终的比对。

考虑上面讨论的4个序列：

S_1 = ATGCTC
S_2 = AGAGC
S_3 = TTCTG
S_4 = ATTGCATGC

如前所述，S_1是中心。比对S_2与S_1如下：

S_1 = ATGCTC
S_2 = A-GAGC

通过S_3与S_1比对添加S_3。

S_1 = ATGCTC
S_3 = -TTCTG

因此，比对变为：

S_1 = ATGCTC
S_2 = A-GAGC
S_3 = -TTCTG

通过S_4与S_1比对添加S_4。

S_1 = AT-GC-T-C
S_4 = ATTGCATGC

此次，添加空格到已比对的S_1中。因此，必须在比对S_2和S_3中添加空格。最终的比对如下：

S_1 = AT-GC-T-C

S_2 = A--GA-G-C
S_3 = -T-TC-T-G
S_4 = ATTGCATGC

可以看到，这是一个相对于S_1所有序列比对的典型近似算法。令$d(S_i, S_j)$表示S_i和S_j之间由近似算法推导出的距离，令 $App=\sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} d(S_i,S_j)$，$d*(S_i, S_j)$表示$S_i$和$S_j$之间的由最优多序列比对计算的距离，令 $Opt=\sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} d*(S_i,S_j)$。现在证明$App\leqslant Opt$。

在正式证明之前，$d(S_1, S_i)=D(S_1, S_i)$。由上面的例子可以检验出。

S_1 = ATGCTC
S_2 = A-GAGC

因此，$D(S_1, S_2)=3$。在算法的结束处，S_1和S_2比对如下：

S_1 = AT-GC-T-C
S_2 = A--GA-G-C

那么，$d(S_1, S_2)=3=D(S_1, S_2)$。由于$\sigma(-, -)=0$，所以这个距离没有改变。

下面是$App\leqslant 2Opt$的证明：

$$
\begin{aligned}
App &= \sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} d(S_i,S_j) \\
&\leqslant \sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} d(S_i,S_1)+d(S_1,S_j) \qquad \text{三角不等式} \\
&= 2(k-1)\sum_{i=2}^{k} d(S_1,S_i) \qquad (d(S_1,S_i)=d(S_i,S_1))
\end{aligned}
$$

对于所有的i，由于$d(S_1, S_i)=D(S_1, S_i)$，有

$$App\leqslant 2(k-1)\sum_{i=2}^{k} D(S_1,S_i) \tag{9-1}$$

现在找出 $Opt=\sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} d*(S_i,S_j)$。首先注意$D(S_i, S_j)$为一个最优2序列比对归纳出的距离。因此，

$D(S_i, S_j)\leqslant d*(S_i, S_j)$

以及

$$
\begin{aligned}
Opt &= \sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} d*(S_i,S_j) \\
&\geqslant \sum_{i=1}^{k}\sum_{\substack{j=1\\j\neq i}}^{k} D(S_i,S_j)
\end{aligned}
$$

但是S_1是中心。因此，

$$
\begin{aligned}
Opt &\geqslant \sum_{i=1}^{k}\sum_{\substack{j=1\\ j\neq i}}^{k} D(S_i,S_j) \\
&\geqslant \sum_{i=1}^{k}\sum_{j=2}^{k} D(S_1,S_j) \\
&= k\sum_{j=2}^{k} D(S_1,S_j)
\end{aligned}
\tag{9-2}
$$

考虑式（9-1）和式（9-2），得到$App \leqslant 2Opt$。

9.8 对换排序问题的2近似算法

在本节中，将介绍通过比较两个染色体的对换算法排序。我们可以简单地把染色体看作是一个基因序列，基因在序列中的次序是非常重要的。因此，在本节中，将每个基因用一个整数表示，染色体用整数序列表示。

两个染色体的比较是很重要的，提供给我们对于物种在遗传学上亲疏远近的内部了解。如果两个染色体很相似，那么它们在遗传学来讲是相近的；否则不是相近的。这里的问题是如何测量两个染色体的相似度。本质上说，通过某些操作将一个染色体转换为另一个的难易程度来量度两个染色体的相似度。在本节将介绍一种操作，也就是对换（transposition）。

由于将一个数字序列转换为另一个序列，不失一般性，可以假设目标序列为1，2，…，n。对换在不改变两个子串次序的情况下，交换任意长度的两个相邻子串。下面是描述这个操作的一个例子。

染色体X：3 <u>1 5</u> <u>2 4</u> → 染色体Y：3 2 4 1 5

通过将一个序列转换为另一个序列所需的最小操作数来定义两个序列的相似度。由于目标序列总是1，2，…，n，可将这个问题看为一个排序问题。但这并非我们所熟悉的普通排序问题而是找到要求排序一个序列的特定操作的最小数目。因此，这个排序问题是一个优化问题。

考虑序列1 4 5 3 2。可以通过对换将这个序列排序为：

1 <u>4 5</u> <u>3 2</u>

1 <u>3</u> <u>2</u> 4 5

1 2 3 4 5

现在开始介绍对换排序。首先，尽管现在讨论排序，可是输入是不同于一般的排序输入。它必须满足下面的条件：

(1) 输入序列不能包含两个相同的数字。例如，不能包含两个5。

(2) 在输入序列中不能出现负数。

(3) 如果序列中出现了i和j，并且$i<k<j$，那么k必须出现在序列中。也就是说，不允许5和7出现了，而6却没有出现。

总之，可以简单地定义输入为1，2，…，n的一个排列。也就是，输入的染色体是由排列$\pi=\pi_1\pi_2\cdots\pi_n$表示的。将排列扩大为包括$\pi_0=0$和$\pi_{n+1}=n+1$，其原因将在后面介绍清楚。例如，一个典型的输入排列为0 2 4 1 3 5。

对于排列π，对换记为$\rho(i, j, k)$（对于所有$1\leqslant i<j\leqslant n+1$，以及所有的$1\leqslant k\leqslant n+1$，有$k\notin[i, j]$），交换排列中的子串$\pi_i$，$\pi_{i+1}$，…，$\pi_{j-1}$和$\pi_j$，$\pi_{j+1}$，…，$\pi_{k-1}$。例如，排列0 7 2 3 6 1 5 4 8中的$\rho(2, 4, 6)$是交换子串(2 3)和(6 1)，结果为0 7 6 1 2 3 5 4 8。已知排列π以及对换ρ，ρ

对π的应用记为$\rho \cdot \pi$。

对换排序问题正式定义如下：已知两个排列π和σ，对换排序问题（sorting by transposition problem）是找到一系列对换ρ_1，ρ_2，…，ρ_t，使得$\rho_t \cdots \rho_2 \cdot \rho_1 \cdot \pi = \sigma$，并且$t$是最小的。$t$称为$\pi$和$\sigma$之间的对换距离（transposition distance）。如前所述，不失一般性，假设σ总是以(0，1，2，…，n，$n+1$)形式的一致排列。接下来，将介绍对换排序问题的2近似算法，该算法由Bafna和Pevzner于1998年提出。注意对换排序问题的复杂度到目前是未知的。

在排列中，对于所有的$0 \leqslant i \leqslant n$，如果$\pi_{i+1} \neq \pi_i + 1$，那么在$\pi_i$与$\pi_{i+1}$之间有一个分界点（breakpoint）。例如，对于排列0 2 3 1 4 5添加分界点之后为0，2 3，1，4 5。一个有序的排列不包含分界点。没有分界点的排列称为一致排列（identity permutation）。因此，我们将输入排列排序为一致排列。

由于一致排列是没有分界点的排列，所以对排列排序相当于减少分界点数。令$d(\pi)$表示将π转换为一个一致排列所需要的最小对换数。每一次对换至多减少三个分界点，因此，$d(\pi)$的通常下界为

（π中分界点的个数）/3

π的循环图表示为$G(\pi)$，是顶点集{0，1，2，…，n，n+1}的有向边着色图，边集定义如下。对于所有的$1 \leqslant i \leqslant n+1$，灰色的边是由$i-1$到$i$的边，黑色的边是从$\pi_i$到$\pi_{i-1}$的边。图9-30是循环图的一个例子，虚线和实线的弧分别代表灰色和黑色的边。

着色图的交替环（alternating cycle）是相邻的每对边是不同颜色的环。对于$G(\pi)$中的每个顶点，每一条入边与唯一的不同颜色的出边成对。因此，$G(\pi)$的边集可以分解为交替环。图9-30是循环图分解为交替环的例子，该交替环如图9-31所示。

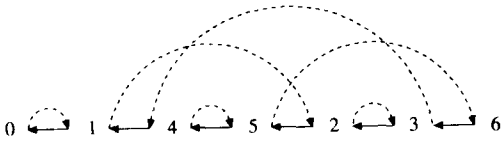


图9-30 排列0 1 4 5 2 3 6的循环图

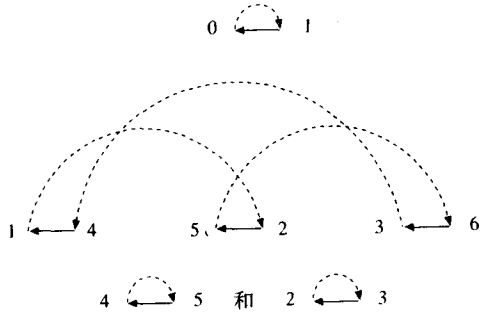


图9-31 循环图分解为交替环

用k环（k-cycle）表示包含k条黑色边的交替环。如果k>2，称k环是长的，否则称为短的。长交替环和短交替环的两个例子如图9-32所示。图9-33是一致排列的循环图。

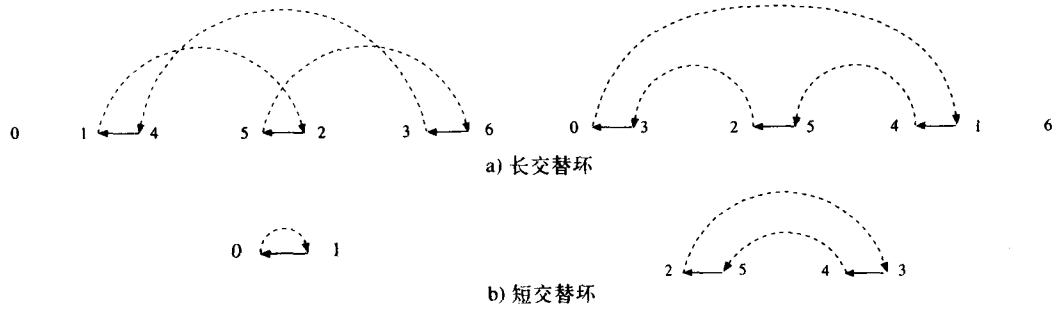


图9-32 长交替循环和短交替循环

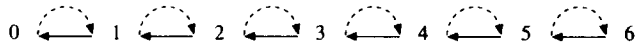


图9-33 一致排列的循环图

如图9-33所示的一致排列中，每个顶点x通过一条灰色的边连接到另一个顶点y，且顶点y通过黑色的边连接回点x，称这种类型的循环图是规则的（regular）。对换排序问题是将一个不规则的循环图转换为规则的循环图。

由于总在处理交替环，所以将交替环简称为环。在$G(\pi)$中至多有$n+1$个环，并且仅有$n+1$个环的排列是一致排列。对于排列π，将$G(\pi)$中环的个数表示为$c(\pi)$。因此，将π排序是将环的个数由$c(\pi)$增加到$n+1$。同样对于排列π将由对换ρ产生的环个数的变化表示为$\Delta c(\rho)=c(\rho\pi)-c(\pi)$。

考虑下面的排列：

0 <u>3 4</u> <u>1 2</u> 5

假设将子串(3 4)和(1 2)对换，如前面所表示的，得到：

0 1 2 3 4 5

读者可以明显地看到上面的对换是非常理想的。最初的排列有三个分界点。对换之后，没有分界点。这可以通过交替环图加以解释，假设对换为$\rho(i, j, k)$，并且涉及$G(\pi)$排序的相应的顶点，即π_{i-1}，π_i，π_{j-1}，π_j，π_{k-1}，π_k在一个环内如图9-34所示。

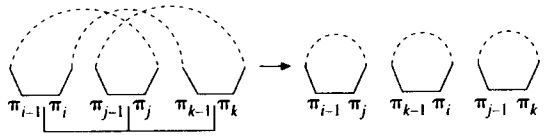


图9-34 对换$\Delta c(\rho)=2$的一个特例

如图9-34所示，这类特殊的对换将环的个数增加了两个。因此，得到更好的下界$d(\pi)=\frac{n+1-c(\pi)}{2}$。任何一个产生与下界$\frac{n+1-c(\pi)}{2}$相等的对换距离的对换排序算法一定是一个最优算法。到目前为止，仍然没有这样的算法。当然，可能下界还是不够高。下面，我们将得到一个2近似算法。

图9-34的例子是非常可取的，这是一个增加两个环的对换。理想地，我们希望在任何时刻排序中都有这种循环。遗憾的是，并不总是这样的。我们必须处理其他的情况。

首先为$G(\pi)$中黑色的边分配一个从1到$n+1$的数，并称对换$\rho(i, j, k)$作用于黑色边i，j和k。图9-35是为黑色的边分配数字的例子。

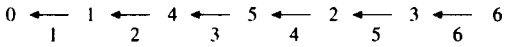


图9-35 $G(\pi)$的黑色边被标记的排列

注意，根据从i_1到i_k对黑色边的访问，环可以用（i_1，…，i_k）来表示，其中i_1是环中最右边的黑色边。例如，图9-36所示的$G(\pi)$中有4个交替环。环b中的最右黑色边是黑色边6，所以环b可表示为（6，2，4）。

有两种不同的环，即非定向环（non-oriented cycle）和定向环（oriented cycle）。对于所有的$k>1$，如果i_1，i_2，…，i_k是递减序列，那么环$C=(i_1, \cdots, i_k)$是非定向的；否则C是定向的。图9-37是非定向和定向环的两个例子。

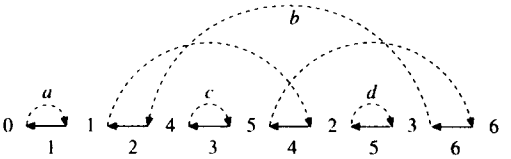
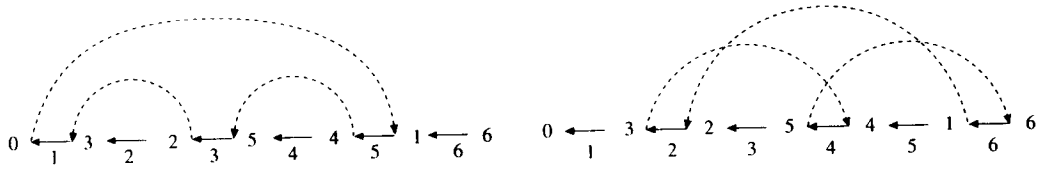


图9-36 $G(\pi)$包含4个循环的排列

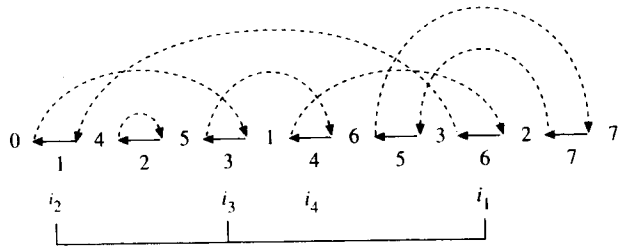
如果$\Delta c(\rho)=x$，我们称对换ρ为x迁移（x-move）。假设循环$C=(i_1, \cdots, i_k)$是定向环，那么可以证明在C中存在i_t，$i_t>i_{t-1}$，$3\leqslant t\leqslant k$，

以及对换$\rho(i_{t-1}, i_t, i_1)$使得$\rho \cdot \pi$产生包含顶点$\pi_{i_{t-1}-1}$和π_{i_t}的1环以及其他环。因此，ρ是一个2-move对换。总之，在每个定向环中有一个2-move对换，图9-38是一个例子，输入排列为0 4 5 1 6 3 2 7。如图9-38a所示，有三个循环，循环（6，1，3，4）是定向环。如图9-38a所示，有对换$\rho = (i_2, i_3, i_1) = \rho(1, 3, 6)$。这个对换对应将子串（4 5）和（1 6 3）交换。在使用这个对换之后，排列变为0 1 6 3 4 5 2 7，并有5个环。如图9-38b，环的个数增加了2个。

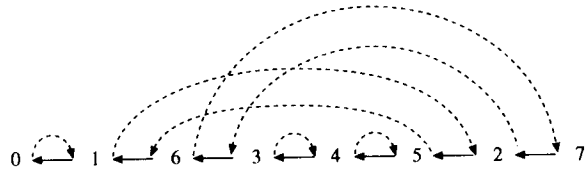


a) 非定向环(5，3，1)　　b) 定向环(6，2，4)

图9-37 定向和非定向环



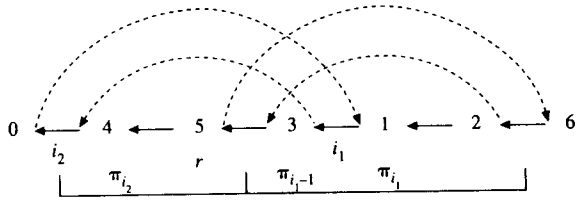
a) 排列0 4 5 1 6 3 2 7，$t = 3$，定向环(6，1，3，4)以及对换$\rho(i_2, i_3, i_1)$



b) 对换之后的环图

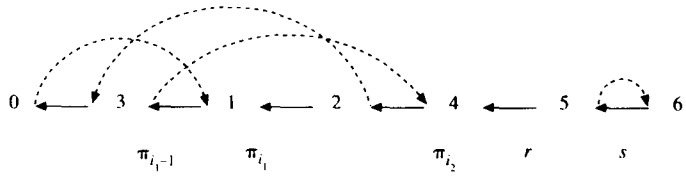
图9-38 允许2-move的定向环

考虑图9-39a，图中没有定向环。容易看出，必定存在多于一个的非定向环。假设环$C = (i_1, i_2, \cdots, i_k)$是非定向环，令在区间$[i_2, i_1-1]$中排列$\pi$中最大元素的位置为$r$，$\pi$中$\pi_{r+1}$的位置为$s$。可以证明$s>i_1$，且对换$\rho = (r+1, s, i_2)$是一个0-move，该对换允许一个2-move将非定向环C转换为定向环C'。因此，在非定向环的2-move之后有一个0-move。图9-39所示是一个例子。



a) 排列0 4 5 3 1 2 6，非定向环$C = (i_1, i_2)$，$r = 5$，$s = 6$，对换$\rho(3, 6, 1)$，$\pi_{i_1} = 1$，$\pi_{i_1-1} = 3$以及$\pi_{i_2} = 4$

图9-39 允许0-move，2-move的非定向环



b) 对换之后的环图

图9-39 （续）

从前面的讨论中看到任意排列π在2-move置换之后有一个2-move置换或者0-move置换。由此，可以得到对换距离的上界。对于对换排序有 $d(\pi)\leqslant\dfrac{n+1-c(\pi)}{2/2}=n+1-c(\pi)$。因此，有一个可以产生不大于$n+1-c(\pi)$的对换距离的算法。

由于下界 $d(\pi)\geqslant\dfrac{n+1-c(\pi)}{2}$ 以及上界$d(\pi)\leqslant n+1-c(\pi)$，所以，有性能比为2的对换排序近似算法，如算法9-8所示。

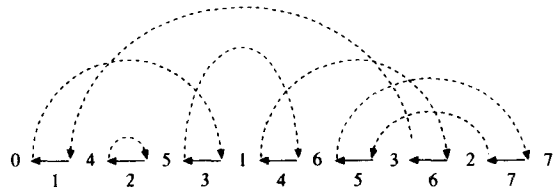
算法9-8 求解对换排序问题近似解的2近似算法

输入：两个排列π和σ。
输出：两个排列π和σ之间的最小距离。
步骤1. 对于排序排列π，重新标记两个排列为一致排列。
步骤2. 构造排列π的环图$G(\pi)$，令距离$d(\pi)=0$。
步骤3. While有定向环
　　执行一个2-move，$d(\pi)=d(\pi)+1$
　While有非定向环
　　在2-move之后执行一个0-move，$d(\pi)=d(\pi)+2$
步骤4. 输出距离$d(\pi)$。

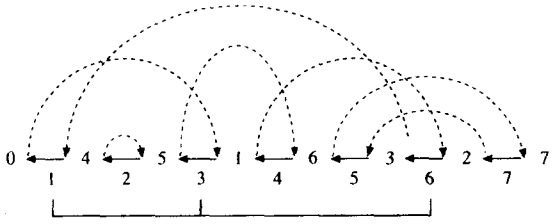
一个2近似算法的例子如图9-40所示。算法执行如下：

(1) 由于有一个定向环（6，1，3，4），执行对换ρ(6，3，1)，结果如图9-40c所示。

(2) 如图9-40b所示，在环图中不存在定向环，存在两个非定向环(6，2)以及(7，3)。在对换ρ(6，5，2)之后执行对换ρ(7，3，2)。对换ρ(7，3，2)的结果如图9-40d所示，对换ρ(6，5，2)的执行结果如图9-40e所示。

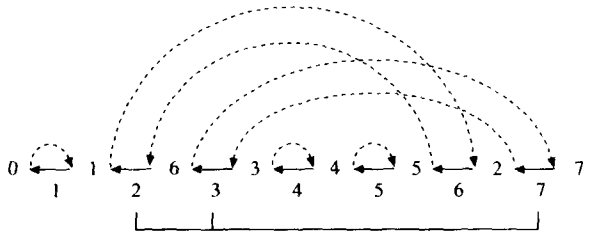


a) 排列4 5 1 6 3 2的环图，定向环（6，1，3，4），环（2）以及非定向环（7，5）

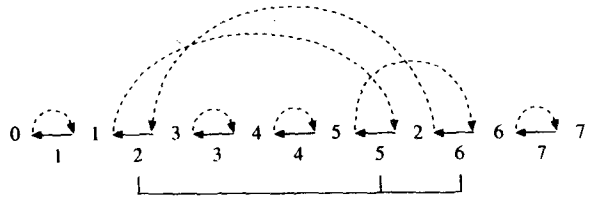


b) 定向环（6，1，3，4）的对换ρ(6，3，1)

图9-40 2近似算法的例子



c) 2-move之后的环图，两个非定向环（6，2）和（7，3），其他环（1），（4）和（5），以及非定向环的对换ρ(7，3，2)



d) c)之后的环图，定向环（6，2，5），其他环（1），（3），（4）和（7），以及定向环进行的对换ρ(6，5，2)

e) 三次对换2-move、0-move和2-move之后的最终环图，

图9-40 （续）

应用对换的整个过程如下所示：

0 <u>4 5</u> <u>1 6 3</u> 2 7

0 1 <u>6</u> <u>3 4 5 2</u> 7

0 1 <u>3 4 5</u> <u>2</u> 6 7

0 1 2 3 4 5 6 7

9.9 多项式时间近似方案

每一个近似算法都包含误差，我们当然希望误差越小越好。也就是说，我们希望有一族近似算法，使得每一个误差有一个相应的可以得到该误差的近似算法。因此，不管特定的误差有多小，都可以通过相应的近似算法得到这个误差。当然，由于小误差算法的时间复杂度必定大于大误差算法的时间复杂度，所以要对小的误差付出代价。不论误差多么小，时间复杂度都是多项式级，是一种理想情况。上面的讨论产生了多项式时间近似方案（polynomial time approximation scheme, PTAS）的概念。

令S_{OPT}为最优解的代价，S_{APX}为近似解的代价。定义误差率为

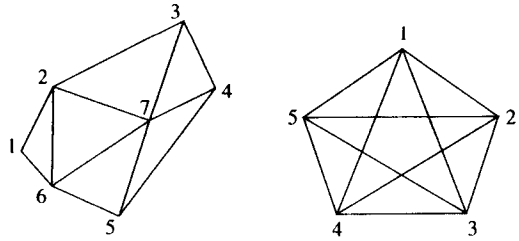
$$\varepsilon = S_{OPT} - S_{APX} / S_{OPT}$$

一个问题的PTAS表示为一个近似算法族，使得每个指定的误差率E有一个具有多项式时间复杂度的近似算法。例如，假设算法的时间复杂度为$O(n/E)$。那么不论E多小，都有一个相对于E的多项式时间近似算法，因为E是常量。

平面图上最大独立集问题的PTAS

在本节将介绍平面图上的最大独立集问题，有一类PTAS，其中每个算法的时间复杂度是$O(8^k kn)$，其中$k = \lceil 1/E \rceil - 1$。

首先定义平面图（planar graph）。称图为平面S上的，如果它画在S上，它的边只相交在端点上。一个图是平面图，如果它画在一个平面上。图9-41a是一个平面图，图9-41b不是平面图。



a）一个平面图　　b）一个非平面图

图9-41　图

平面图上的最大独立集问题是NP难的。因此，需要近似算法。首先定义一些术语。面（face）是一个区域，由嵌入的平面定义。没有边界的面称为外部面（exterrior face），其他面称为内部面（interior face）。例如，对于图9-42中的面，面6→7→11→12→6是一个内部面，而面1→2→3→4→5→1是一个外部面。在平面图中，可以对每一个结点分配一个层次。在图9-42中，仅有一个外部面1→2→3→4→5→1。因此，结点1，2，3，4和5是层1。随后，结点6，7，11，12，8，9，10，13，14是层2，结点15，16，17是层3。结点的层可在线性时间内计算出来。

如果一个图没有大于k的结点，那么称该图为k外平面图（k-outerplanar）。例如，图9-43包含一个2外平面图。

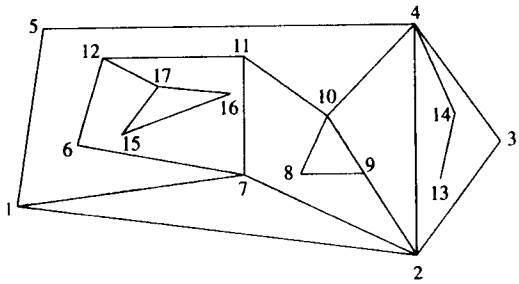


图9-42　平面图的嵌入

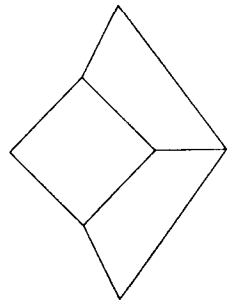


图9-43　2外平面图的例子

对于k外平面图，最大独立集问题的最优解可以通过动态规划方法，在时间$O(8^k n)$中找到，其中n是顶点的个数，在这里并不给出动态规划方法的细节。

已知任意平面图G，可以将它分解为k外平面图的集合。参见图9-44，置k为2。然后将层3、6和9中的所有结点组成类3，将层1、4和7中的结点组成类1，层2、5和8中的结点组成类2，如表9-1所示。

表9-1　结点的分组

层	1 4 7	（类1）
层	2 5 8	（类2）
层	3 6 9	（类3）

如果删除类3中的所有结点，即层3，6和9中的结点，那么得到的结果如图9-45所示。显然图9-45中的所有子图都是2外平面图。每一个2外平面图可以在线性时间内找到其最大独立集。此外，对于原始平面图来讲，这些最大独立集的并仍然是一个独立集（不必是最大的），因此可以作为一个近似解。

类似地，可以删除类1中的结点，即层1，4和7中的点，结果图中仍然由2外平面图集组成。采用相似的方法可以得到最初平面图的近似最大独立集。

现在，类1中结点的层都是同余为1的（mod k +1），类2中的结点的层都是同余为2的（mod k +1）。基于某个指定的k，近似算法运行如下：

算法9-9 在平面图上求解最大独立集问题的近似算法

步骤1. For 所有的$i=0$，1，…，k，do

(1.1) 令G_i为通过删除同余为$i(\bmod k+1)$的层上的所有结点得到的图，剩余的子图均是k外平面图。

(1.2) 对于每个k外平面图，找出它的最大独立集。令S_i为这些解的并集。

步骤2. 在S_0，S_1，…，S_k中，选择最大规模的S_i，并令其为近似解S_{APX}。

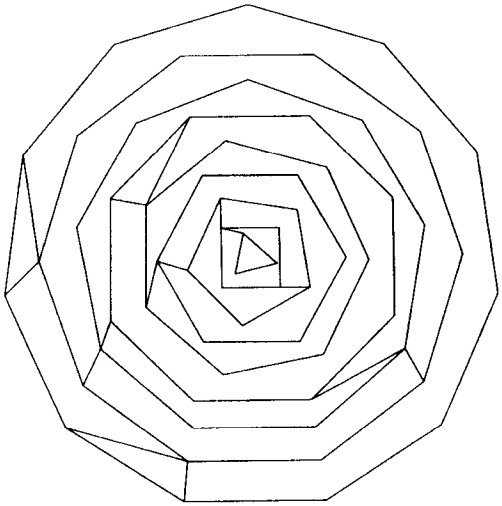


图9-44 9层的平面图

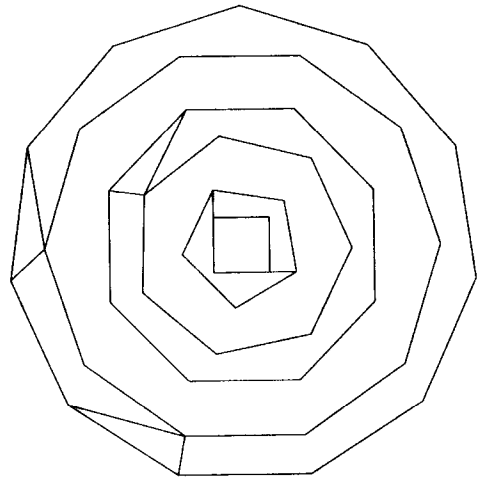


图9-45 删除3，6和9层上的结点得到的图

上面近似算法的时间复杂度显然是$O(8^k kn)$。下面将介绍k与误差率成反比。因此，平面图上的最大独立集问题有一个PTAS。

注意，已将所有的结点分为（$k+1$）个类；每一类对应一个同余为$i(\bmod k+1)$的层，$i=0$，1，…，k。每个独立集S的结点个数的平均数是$|S|/(k+1)$，其中$|S|$是这个独立集中结点的个数。因此，至少有一个r使得在S_{OPT}中至多有$\frac{1}{k+1}$个结点同余为$r(\bmod k+1)$。这意味着通过从S_{OPT}中删除类r中的结点得到解S_r，因为至多删除$|S_{OPT}|\frac{1}{k+1}$个结点，该解至少有$|S_{OPT}|\left(1-\frac{1}{k+1}\right)=|S_{OPT}|\frac{k}{k+1}$个结点。因此，

$$|S_r| \geqslant |S_{OPT}|\frac{k}{k+1}$$

根据算法，有

$$|S_{APX}| \geqslant |S_r| \geqslant |S_{OPT}|\frac{k}{k+1}$$

或

$$\varepsilon = \frac{|S_{OPT}-S_{APX}|}{|S_{OPT}|} \leqslant \frac{1}{k+1}$$

因此，如果置$k=\lceil 1/E \rceil - 1$，那么上面的公式为

$$\varepsilon \leqslant \frac{1}{k+1} = \frac{1}{\lceil 1/E \rceil} \leqslant E$$

这表明每个已知的误差界限E都有相对的k保证近似解在误差率范围内不同于最优解。此外，无论误差多么小，都可以在$O(8^k kn)$时间复杂度内找到达到该误差的算法，该算法相对于n来说是多项式的。

0/1背包问题的PTAS

0/1背包问题是一个NP难问题，在第5章中讨论过。实在令人惊奇的是对于该问题存在多项式时间的近似方案。

0/1背包问题定义如下：已知n个物品，第i个物品有利润p_i以及重量w_i。已知一个整数M，0/1背包问题是挑选这n个物品的子集，使得在总重量不超过M的约束下，取得的利润总和最大。在形式上，希望使$\sum\delta_i p_i$最大，其中$\delta_i = 1$或0，约束为$\sum\delta_i w_i \leqslant M$。

看一个例子。$M = 92$，$n = 8$，利润和重量如表9-2所示，物品的顺序是根据p_i/w_i的非递增次序确定的。

表9-2 8个物品的利润和重量

i	1	2	3	4	5	6	7	8
p_i	90	61	50	33	29	23	15	13
w_i	33	30	25	17	15	12	10	9
p_i/w_i	2.72	2.03	2.0	1.94	1.93	1.91	1.5	1.44

基本上，近似算法建立在误差率ε上。一旦给定ε，通过精心地计算，计算一个阈值，称为T。由这个阈值T，将所有n个物品分成两个子集：BIG和$SMALL$。BIG包含所有利润大于T的项，$SMALL$包含所有利润小于等于T的项。

在这个例子中，T为46.8。因此，$BIG = \{1, 2, 3\}$，$SMALL = \{4, 5, 6, 7, 8\}$。

在得到BIG与$SMALL$之后，将BIG中物品的利润标准化。这个例子中，标准化利润是

$p_1' = 9$

$p_2' = 6$

及 $p_3' = 5$。

现在尽量枚举出这个问题实例的所有可能解，有很多这样的解。我们考虑其中的两个：

解1：挑选物品1和2。标准化利润的总和为15，相应原始利润的总和是90 + 61 = 151，重量总和为63。

解2：挑选物品1、2和3。标准化利润总和为20，相应原始利润的总和是90 + 61 + 50 = 201，重量总和为88。

由于重量总和小于$M = 92$，我们可以从$SMALL$中向两个解中添加某些物品。这可以由贪心方法得到。

解1：对于解1，可添加物品4和6，利润总和为151+33+23 = 207。

解2：对于解2，并不能从$SMALL$中添加任何物品，因此利润总和为201。

当然还有很多这样的解。每个解都可以用贪心算法得到一个近似解。可以证明解1是最大的。因此，输出解1作为近似解。近似算法必须是一个多项式算法，这一点很重要。贪心法部分明显是多项式的。算法关键的部分是在BIG中为物品找到可行解的部分。这些可行解之一是0/1背包问题的一个最优解。如果算法的这部分是多项式的，是否意味着可以用多项式算法来解决0/1背包问题呢？

后面将证明，这部分实际上解决了一个特定的0/1背包问题。随后将介绍在该问题的实例中，利润被标准化在标准化利润之和小于$\lfloor(3/\varepsilon)^2\rfloor$的限度内，$\varepsilon$是误差率。也就是说，标准化

利润总和小于一个常数。只要满足这个条件，就有一个多项式时间算法，产生所有可能的可行解。因此，这里的近似算法是多项式的。

在给出了这个近似算法上层的概述后，现在给出该算法的细节。再次应用上述数据来描述算法。令ε为0.6。

步骤1：根据利润与重量比率p_i/w_i，将物品非递增排序（表9-3）

表9-3 排序物品

i	1	2	3	4	5	6	7	8
p_i	90	61	50	33	29	23	15	13
w_i	33	30	25	17	15	12	10	9
p_i/w_i	2.72	2.03	2.0	1.94	1.93	1.91	1.5	1.44

步骤2：计算数Q如下：

找到最大的d，使得

$W = w_1 + w_2 + \cdots + w_d \leqslant M$

如果 $d = n$ 或 $W = M$，那么

置$P_{APX} = p_1 + p_2 + \cdots + p_d$ 及$INDICES = \{1, 2, \cdots, d\}$

停止。

在这个例子中，$P_{OPT} = P_{APX}$

否则，置$Q = p_1 + p_2 + \cdots + p_d + p_{d+1}$

在本例中，$d = 3$，$Q = 90 + 61 + 50 + 33 = 234$。

Q的特性是什么？现在证明

$$Q/2 \leqslant P_{OPT} \leqslant Q$$

注意，$p_1 + p_2 + \cdots + p_d \leqslant P_{OPT}$

由于$W_{d+1} \leqslant W$，所以，p_{d+1}本身是一个可行解。

$$p_{d+1} \leqslant P_{OPT}$$

所以，$Q = p_1 + p_2 + \cdots + p_d + p_{d+1} \leqslant 2P_{OPT}$。

或者 $Q/2 \leqslant P_{OPT}$

由于P_{OPT}是一个可行解，而Q不是，可以得到$P_{OPT} \leqslant Q$。

因此，$Q/2 \leqslant P_{OPT} \leqslant Q$

后面将看到这对于误差分析是关键。

步骤3：计算标准化因子δ：

$\delta = Q(\varepsilon/3)^2$

在本例中，$\delta = 234(0.6/3)^2 = 234(0.2)^2 = 9.36$

那么计算参数g：

$$g = \lfloor Q/\delta \rfloor = \lfloor (3/\varepsilon)^2 \rfloor = \lfloor (3/0.6)^2 \rfloor = 25$$

置$T = Q(\varepsilon/3)$。

在这个例子中，$T = 234(0.6/3) = 46.8$。

步骤4：

步骤4.1：令$SMALL$合并所有利润小于等于T的物品，其他的物品放入BIG。

在本例中，$SMALL = \{4, 5, 6, 7, 8\}$

$BIG = \{1, 2, 3\}$

步骤4.2：对于BIG中的所有物品，根据下面的公式标准化其利润：$P_i' = \lfloor P_i / \delta \rfloor$。

在本例中，$P_1' = \lfloor 90/9.36 \rfloor = 9$

$P_2' = \lfloor 61/9.36 \rfloor = 6$

$P_3' = \lfloor 50/9.36 \rfloor = 5$

步骤4.3：用规模g初始化数组A。数组的每一个元素对应于p_i'的一个组合。每个元素$A[i]$包含三个字段I、P和W，分别代表组合的下标值、利润的总和以及重量的总和。

步骤4.4：对于BIG中的每个物品i，扫描这个表，对每一个数组元素执行下面的操作：对于元素$A[j]$，如果$A[j]$中总是非空，添加物品i将不会使重量总和超过限制M，那么检查相应的$A[j+p_i']$的重量，该重量与添加i到$A[j]$的组合相对应。如果$A[j+p_i']$为空，或者$A[j+p_i'] \cdot W$的重量大于$A[j] \cdot W + w_i$（是添加到$A[j]$中的i相对应的重量），那么用$A[j] \cdot I \cup \{i\}$更新这个数组元素，并且更新对应的利润和重量。

本例运行如下。

当$i = 1$，$p_i' = 9$，如表9-4所示。

表9-4 i = 1，p_i' = 9的数组A

p_i	I	P	W	p_i	I	P	W
0		0	0	13			
1				14			
2				15			
3				16			
4				17			
5				18			
6				19			
7				20			
8				21			
9	1	90	33	22			
10				23			
11				24			
12				25			

$i = 2$，$p_2' = 6$，如表9-5所示。

表9-5 i = 2，p_2' = 6的数组A

p_i	I	P	W	p_i	I	P	W
0		0	0	13			
1				14			
2				15	1, 2	151	63
3				16			
4				17			
5				18			
6	2	61	30	19			
7				20			
8				21			
9	1	90	33	22			
10				23			
11				24			
12				25			

$i = 3$，$p_3' = 5$，如表9-6所示。

表9-6 i = 3，p_3' = 5的数组A

p_i	I	P	W	p_i	I	P	W
0		0	0	13			
1				14	1, 3	140	58
2				15	1, 2	151	63
3				16			
4				17			
5	3	50	25	18			
6	2	61	30	19			
7				20	1, 2, 3	201	88
8				21			
9	1	90	33	22			
10				23			
11	2, 3	111	55	24			
12				25			

查看表9-6。每个非空数组元素表示一个可行解。例如，$A[j] = 5$对应仅为一件物品的选项，如物品3。其标准化利润的总和为5，利润总和为50，重量总和为25。对于$j = 25$，选择物品1和2。在这种情况下，其标准化利润为15，利润总和为151，重量总和为63。

步骤5：对于数组A的每个元素，用贪心算法将$SMALL$中的物品添加到对应的组合中。

表9-7 最终的数组A

p_i	I	P	W	添加	利润	p_i	I	P	W	添加	利润
0		0	0	4,5,6,7,8	113	13					
1						14	1,3	140	58	4,5	202
2						15	1,2	151	63	4,5	207
3						16					
4						17					
5	3	50	25	4,5,6,7,8	163	18					
6	2	61	30	4,5,6,7	161	19					
7						20	1,2,3	201	88		201
8						21					
9	1	90	33	4,5,6,7	190	22					
10						23					
11	2,3	111	55	4,5	173	24					
12						25					

再次查看表9-7的含义。对于$j = 15$，可以添加物品4和6，得到利润总和为207。对于$j = 20$，从BIG中可选择物品1，2和3。在这种情况下，由于将超过重量约束，所以不会从$SMALL$中添加任何物品。因此，重量总和为201。

步骤6：选择步骤5中最大的利润为近似解。本例中，选定$\Sigma p_i' = 15$对应的数组元素，其产生的最大总利润为207。

读者也许观察到，步骤4.4是一个穷举扫描步骤，这些物品的标准化利润为9，6和5，对应重量分别为33，30和25。从利润为9的物品开始，由于其重量小于M，可以只挑选这一个，其总重量为33，总标准化利润为9，然后扫描标准化利润为6的物品。只挑选这一物品。如果

这样挑选，其总标准化利润为6，总重量为30。如果挑选已经扫描过的两个物品，那么总的标准化利润为9 + 6 = 15，总的重量是33 + 30 = 63。因此，有下面4个可行解：

(1) 不选任何一个。

(2) 只选物品1，总的标准化利润为9，总重量为33。

(3) 只选物品2，总的标准化利润为6，总重量为30。

(4) 挑选两个物品1和2，总的标准化利润为15，总重量为63。

直觉上，这一步花费的时间是指数级的：$2^{|BIG|}$。这就出现了一个关键的问题：为什么这一步是多项式的？

下面，将证明数组A的大小不会大于g。

令数组A的最大元素有标准化利润$p_{i_1}' + p_{i_2}' + \cdots + p_{i_j}'$。由于这对应可行解，所以有$p_{i_1} + p_{i_2} + \cdots + p_{i_j} \leqslant P_{OPT} \leqslant Q$，$p_{i_j}' = \lfloor p_{i_j} / \delta \rfloor$。

考虑$p_{i_1}' + p_{i_2}' + \cdots + p_{i_j}'$。

$$
\begin{aligned}
& p_{i_1}' + p_{i_2}' + \cdots + p_{i_j}' \\
&= \lfloor p_{i_1} / \delta \rfloor + \lfloor p_{i_2} / \delta \rfloor + \cdots + \lfloor p_{i_j} / \delta \rfloor \\
&\leqslant \lfloor (p_{i_1} + p_{i_2} + \cdots + p_{i_j}) / \delta \rfloor \\
&\leqslant \lfloor Q / \delta \rfloor \\
&= g
\end{aligned}
$$

这就是为什么大小为g的数组是足够的。g是一个常数，与n无关。扫描代价时间至多为ng，这就是为什么步骤4是多项式的。

近似算法的时间复杂度如下：

步骤1：$O(n \log n)$

步骤2：$O(n)$

步骤4.1到4.2：$O(n)$

步骤4.3：$O(g)$

步骤4.4：$O(ng)$

步骤5：$O(ng)$

步骤6：$O(n)$

近似算法总的时间复杂度是

$$
\begin{aligned}
& O(n \log n)+O(n)+O(n)+O(g)+O(ng)+O(ng)+O(n) = O(n \log n)+O(ng) \\
& = O(n \log n)+O(n(3/\varepsilon)^2)
\end{aligned}
$$

因此，这个算法是一个多项式时间近似方案，其时间复杂度是ε的函数。我们所设置的误差率越小，时间复杂度就变得越大，但是它仍然是多项式的。

现在对该近似算法作误差分析。首先，需要介绍一些断言。

断言1：对于 $\forall i \in BIG$，$p_i' \geqslant 3/\varepsilon$

已知$p_i' = \lfloor p_i / \delta \rfloor = \lfloor p_i / Q(\varepsilon/3)^2 \rfloor$

但是 $\forall i \in BIG$，$p_i > Q(\varepsilon/3)$

所以，$p_i' \geqslant 3/\varepsilon$。

断言2：对于 $\forall i \in BIG$，$p_i'\delta \leqslant p_i \leqslant p_i'\delta(1 + \varepsilon/3)$。

注意 $p_i' = \lfloor p_i/\delta \rfloor$

所以，$p_i' \leqslant p_i/\delta$

因此，有$p_i'\delta \leqslant p_i$。

此外，根据断言1的$p_i' \geqslant 3/\varepsilon$，还有$p_i/\delta \leqslant p_i' + 1 = p_i'(1 + 1/p_i') \leqslant p_i'(1 + \varepsilon/3)$。

断言3：令P_{OPT}为0/1背包问题的一个最优利润，P_{Greedy}为对同一问题采用贪心法计算得到的利润。那么

$P_{OPT} - P_{Greedy} \leqslant \max\{p_i\}$。

证明：令b为贪心算法没有选择的第一个物品。根据贪心算法的选择，

$P_{Greedy} + p_b \geqslant P_{OPT}$

这样，$P_{OPT} \leqslant P_{Greedy} + P_b \leqslant P_{Greedy} + \max\{p_i\}$

所以，$P_{OPT} - P_{Greedy} \leqslant \max\{p_i\}$。

现在讨论近似算法的误差分析。

首先，P_{OPT}可以表示如下：

$P_{OPT} = p_{i_1} + p_{i_2} + \cdots + p_{i_k} + \alpha$

其中p_{i_1}，p_{i_2}，…，p_{i_k}是BIG中的物品，α是$SMALL$中的物品对应的利润总和。

令c_{i_1}，…，c_{i_k}的对应重量为p_{i_1}，p_{i_2}，…，p_{i_k}。考虑$p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}'$，由于$p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}'$是不唯一的，可能有另一个物品集合，即$j_1$，$j_2$，…，$j_h$，使得$p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}' = p_{j_1}' + p_{j_2}' + \cdots + p_{j_h}'$，以及$c_{j_1} + \cdots + c_{j_h} \leqslant c_{i_1} + \cdots + c_{i_k}$。也就是说，近似算法选择物品$j_1$，$j_2$，…，$j_h$，来代替$i_1$，$i_2$，…，$i_k$。

算法在执行到步骤6期间得到总和$P_H = p_{j_1} + p_{j_2} + \cdots + p_{j_h} + \beta$。

显然，根据P_{APX}的选择，

$P_{APX} \geqslant P_H$。

因此，

$P_{OPT} - P_{APX} \leqslant P_{OPT} - P_H$

根据断言2，有

$p_i'\delta \leqslant p_i \leqslant p_i'\delta(1 + \varepsilon/3)$

将其带入到公式$P_{OPT} = p_{i_1} + p_{i_2} + \cdots + p_{i_k} + \alpha$和$P_H = p_{j_1} + p_{j_2} + \cdots + p_{j_h} + \beta$，可以看到

$(p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}')\delta + \alpha \leqslant P_{OPT} \leqslant (p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}')\delta(1 + \varepsilon/3) + \alpha$

以及$(p_{j_1}' + \cdots + p_{j_h}')\delta + \beta \leqslant P_H \leqslant (p_{j_1}' + \cdots + p_{j_h}')\delta(1 + \varepsilon/3) + \beta$

由于$p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}' = p_{j_1}' + \cdots + p_{j_h}'$，可以得到

$P_{OPT} \leqslant (p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}')\delta(1 + \varepsilon/3) + \alpha$

以及$P_H \geqslant (p_{i_1}' + p_{i_2}' + \cdots + p_{i_k}')\delta + \beta$

因此，有

$$\begin{aligned}(P_{OPT} - P_H)/P_{OPT} &\leqslant ((p_{i_1}' + \cdots + p_{i_k}')\delta(\varepsilon/3) + \alpha - \beta)/P_{OPT} \\ &\leqslant \varepsilon/3 + (\alpha - \beta)/P_{OPT}\end{aligned} \tag{9-3}$$

必须进一步证明

$|\alpha - \beta| \leqslant Q(\varepsilon/3)$

这可推导如下：

α是在容量为$M - (c_{i_1} + c_{i_2} + \cdots + c_{i_k})$的$SMALL$中定义的物品的0/1背包问题中利润的总和。

β是在容量为$M - (c_{j_1} + c_{j_2} + \cdots + c_{j_k})$的$SMALL$中定义的物品的0/1背包问题由贪心算法得到的利润总和。令β'为在容量为$M - (c_{i_1} + c_{i_2} + \cdots + c_{i_k})$的$SMALL$中定义的物品的0/1背包问题由

贪心算法得到的利润总和。

情况1：$\alpha \geqslant \beta$

由于假设$c_{i_1}+c_{i_2}+\cdots+c_{i_k} \geqslant c_{j_1}+c_{j_2}+\cdots+c_{j_h}$，

$M-(c_{i_1}+c_{i_2}+\cdots+c_{i_k}) \leqslant M-(c_{j_1}+c_{j_2}+\cdots+c_{j_h})$

而且，β和β'通过贪心法在$SMALL$中得到，$SMALL$以非递增次序排序。因此，

$M-(c_{i_1}+c_{i_2}+\cdots+c_{i_k}) \leqslant M-(c_{j_1}+c_{j_2}+\cdots+c_{j_h})$隐含着$\beta' \leqslant \beta$。

所以，

$\alpha-\beta \leqslant \alpha-\beta'$

根据断言3，同样有

$\alpha-\beta' \leqslant \max\{P_i | i \in SMALL\}$

由于$SMALL$中每个物品的利润小于等于$Q(\varepsilon/3)$的性质，得到

$\alpha-\beta \leqslant Q(\varepsilon/3)$

情况2：$\alpha \leqslant \beta$

由于$(p_{j_1}'+\cdots+p_{j_k}')\delta+\beta \leqslant P_H \leqslant P_{OPT} \leqslant (p_{i_1}'+\cdots+p_{i_k}')\delta(1+\varepsilon/3)+\alpha$

所以，$\beta-\alpha \leqslant (p_{i_1}'+\cdots+p_{i_k}')\delta(\varepsilon/3)$

$\leqslant (p_{i_1}+\cdots+p_{i_k})(\varepsilon/3)$

$\leqslant P_{OPT} \cdot \varepsilon/3$

$\leqslant Q(\varepsilon/3)$

考虑情况1和情况2，可以得到

$$|\alpha-\beta| \leqslant Q(\varepsilon/3) \tag{9-4}$$

将式（9-4）代入式（9-3），得到

$(P_{OPT}-P_H) \leqslant \varepsilon/3+(\varepsilon/3)Q/P_{OPT}$

根据Q的选择，

$P_{OPT} \geqslant Q/2$

由不等式$P_{APX} \geqslant P_H$，得到$(P_{OPT}-P_{APX})/P_{OPT} \leqslant \varepsilon/3+2\varepsilon/3=\varepsilon$。

上面的等式表明这个近似算法是一个多项式时间近似方案。

9.10 最小路径代价生成树问题的2近似算法

最小路径代价生成树问题（minimum routing cost spanning tree problem，MRCT）类似于最小生成树问题。在最小生成树问题中，生成树中所有的边总代价是最小的。在最小路径代价生成树问题中，我们感兴趣的是路径的代价。对于树中任意两个结点u和v，在它们之间有一条路径，这条路径中所有的边总代价称为这对结点的路径代价。进一步规定已知的图是一个完全图，且所有的边代价满足三角不等式。因此，最小路径代价生成树问题定义如下：已知完全图G，其边的代价和满足三角不等式，最小路径代价生成树问题是找到G的一棵生成树，该生成树中结点间所有成对结点的路径代价总和最小。

参见图9-46，图9-46a中的完全图的一棵最小路径代价生成树如图9-46b所示。

令$RC(u, v)$表示树中结点u和v之间的路径代价，该树中所有成对结点的路径代价的总和是

$RC(a, b)+RC(a, c)+RC(a, d)+RC(b, c)+RC(b, d)+RC(c, d)$

$=1+2+2+1+1+2$

$=9$

这是所有可能生成树中最小的。

最小路径代价生成树问题是NP难的。在本节中，将给出该问题的一个2近似算法。在下一节中，将会进一步介绍这个问题的PTAS。

在介绍算法之前，首先指出，我们将每对结点计数两次。也就是说，从结点u到v之间的路径以及从结点v到u之间的路径都计数，这样做是为了简化讨论。

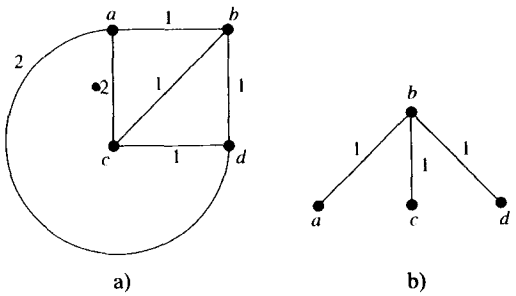


图9-46 完全图的最小路径代价生成树

2近似算法基于称为质心（centroid）的概念。一棵树的质心是这样一个结点，该结点的删除会导致每一个子图将包含不多于$n/2$个结点，其中n是树中所有结点的总数。参见图9-47，结点m是一个质心。

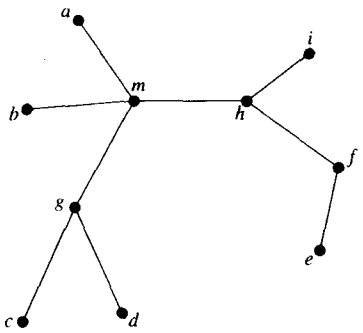


图9-47 树的质心

考虑树T中的任意结点v，T以其质心m为根。根据定义，m的每一棵子树包含不多于$n/2$个结点。考虑T中包含任意结点v的子树必定包含不多于$n/2$个结点。这意味着，在结点u与v之间至少有$n/2$条通过m的路径。例如，考虑图9-47中树的结点e。只有e与f、h和i之间的路径没有通过质心m，其他所有结点到e的路径都会通过m。因此，在这棵树的路径代价中，从任意结点v到m的路径长度将计数至少$2(n/2) = n$次。用$RC(u, v)$表示结点u和v之间路径的代价，树$C(T)$的代价是

$$C(T) \leqslant n \sum_{u} RC(u,m) \tag{9-5}$$

假设树T是一棵最小路径代价生成树，m是T的一个质心，那么上面的等式仍然成立。现在，用m来得到近似算法。如图9-48所示，定义1星树（1-star）是只有一个内部结点，而其他结点都是叶子结点的树。

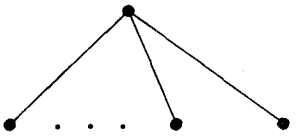


图9-48 1星树

近似算法是搜集所有到m的结点以形成1星树S，该树同样是一棵生成树。令$w(v, m)$表示原始图G在结点v与m之间边的权值。这棵星树的路径代价的总和记为$C(S)$，表示如下：

$$C(S) = (2n-1) \sum_{v} w(v,m) \tag{9-6}$$

因为三角不等式，所以$w(v, m)$小于T的$RC(v, m)$。因此，可以得到

$$C(S) = 2n \sum_{v} RC(v,m) \tag{9-7}$$

这就是

$$C(S) \leqslant 2C(T) \tag{9-8}$$

因此，构造的1星树S是一个最小路径代价生成树问题的2近似解。从最小路径生成树开始，但是并没得到任何最小路径代价生成树。如果这样做，就不需要这个问题的任何近似算法了。

在没有树的情况下，如何找到该树的质心m呢？已知完全图G，它只有n棵1星树。因此，可以产生一个穷举搜索来构造所有可能的n棵1星树，将具有最小代价的那个作为近似解。这棵1星树必须满足要求，近似算法的时间复杂度是$O(n^2)$。

可能读者会注意到，我们仅证明了满足要求的1星树的存在。穷举搜索可能会找到一棵1星树，其路径代价小于上面讨论的代价。

9.11 最小路径代价生成树问题的PTAS

在上节介绍了对于最小路径代价生成树问题的2近似算法。该近似算法是基于这样的思想：每个图G存在一棵1星树，其路径代价至多是最小路径代价生成树路径代价的两倍。由于仅有n棵1星树，所以可在多项式步数内找到近似解。

在本节将介绍最小路径生成树问题的PTAS。本质上讲，构建k星树，k越大，误差就越小。

一棵k星树是确有k个内部结点的树。图9-49是一棵3星树。

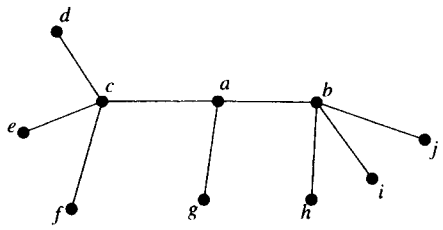


图9-49 一棵3星树

在本节剩余部分中，令$mcks(G, k)$表示图G的最小路径代价k星树的路径代价，$mrcst(G)$表示G的最小路径代价生成树的路径代价。将证明：

$$mcks(G,k) \leqslant \left(\frac{k+3}{k+1}\right) mrcst(G) \tag{9-9}$$

当$k = 1$时，构造一棵1星树，上面的公式变为：

$$mcks(G, 1) \leqslant 2mrcst(G) \tag{9-10}$$

这在上一节已证明。

等式（9-9）表示用k星树近似最优解的误差率为

$$E = \frac{2}{k+1} \tag{9-11}$$

对于$k = 2$，误差率为0.66；$k = 5$，误差率减小到0.33。给定一个误差界，用下面的等式挑选k：

$$k = \left\lceil \frac{2}{E} - 1 \right\rceil \tag{9-12}$$

因此，较大的k意味着较小的误差率。对于给定的误差界限，可以选出相应足够大的k以保证由k星树导出的误差不超过指定的误差界限。可以证明，找到一个给定完全图的最小路径代价k星树的时间复杂度为$O(n^{2k})$。对于每个k，不管它多大，近似算法的时间复杂度仍然是多项式的。也就是，对于最小代价生成树问题有PTAS。

为了找出最小路径代价k星树，需要一个称为δ分离（δ-separator）的概念，其中$0<\delta\leqslant 1/2$。已知图G，G的δ分离是G的一个最小子图，删除它将产生每个包含不多于δn个结点的子图。对于$\delta = 1/2$，δ分离仅包含一个结点，即质心（centriod），将在最后一节介绍。可以证明，在δ与k之间有如下关系式：

$$\delta = \frac{2}{k+3} \tag{9-13}$$

或者，反之，

$$k=\frac{2}{\delta}-3 \tag{9-14}$$

用式(9-14)代替式(9-9)，得

$$mcks(G,k)\leqslant\left(\frac{1}{1-\delta}\right)mrcst(G) \tag{9-15}$$

从本质上讲，推理过程如下：如果有误差率界限E，可以根据等式（9-12）挑选k，然后通过等式（9-13）确定δ。经过δ分离，可以确定满足误差界限要求的k星树。假设指定E为0.4。那么，用等式（9-12）选出$k=(2/0.4)-1=4$，找出$\delta=2/(4+3)=2/7=0.28$。

接下来先讨论$k=3$的情况来说明PTAS的基本概念。在这种情况下，通过等式（9-13）得到$\delta=2/(3+3)=1/3$。

假设已经找到最小路径代价生成树T。不失一般性，假设T的根为其质心m，那么至多有两棵包含多于$n/3$个结点的子树。令结点a和b是具有至少$n/3$个结点的最低层结点，忽略$m=a$或$m=b$的特殊情况。这两种特殊情况的处理相同，产生的结果也相同。令P表示T中从点a到点b的路径，因为没有质心m的子树含有多于$n/2$个结点，所以这条路径必须包含质心m。根据定义，P是(1/3)分离。接下来，证明基于P，如何构造最小路径代价3星树。3星树路径代价不会多于T的路径代价

$$\frac{k+3}{k+1}=\frac{3+3}{3+1}=\frac{3}{2}$$

首先，将所有结点划分为V_a，V_b，V_m，V_{am}和V_{bm}。集合V_a、V_b和V_m分别由在P中的最低祖先分别为a、b和m的结点组成，集合V_{am} (V_{bm})分别与在P中最低祖先分别为a和m之间（在b和m之间）的结点组成。

然后将P用具有边(a，m)和(b，m)的路径代替。将V_a，V_b和V_m中的每个结点v，分别与a、b和m连接。V_{am} (V_{bm})中的所有结点都连接到$a(b)$，或者都连接到m。因此，有4棵3星树，下面证明其中之一是满足要求的。一个典型的情况如图9-50所示，显示了所有的4棵3星树。

现在尽量找出最小路径代价生成树路径代价的公式。该树上的每个结点v有一个从v到P中结点的路径。令v到P中结点的路径长度记为$dt(v, P)$。由于P是一个(1/3)分离，所以在总的路径代价中，路径长度必定至少计数$n/3$次。对于P中的每一条边，由于在边的某侧至少有$n/3$个结点，所以在路径代价中边至少计数$(n/3)(2n/3)$次。令$w(P)$为P的总路径长度，那么有

$$mrcst(G)\leqslant(2n/3)\sum_{v}dt(v,P)+(2/9)n^2w(P) \tag{9-16}$$

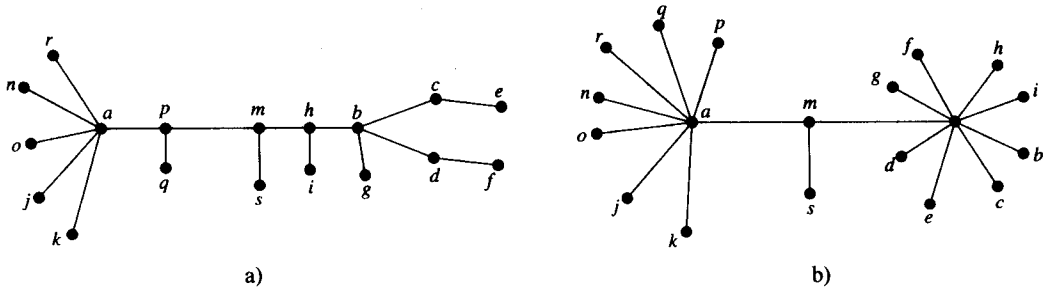
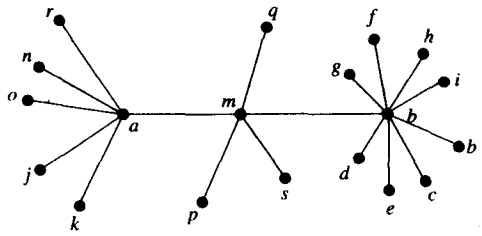
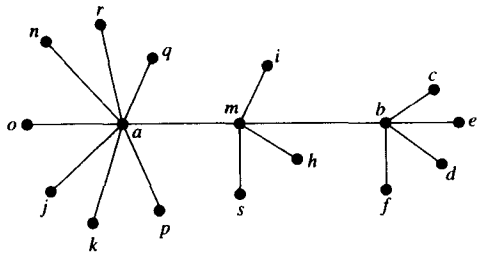


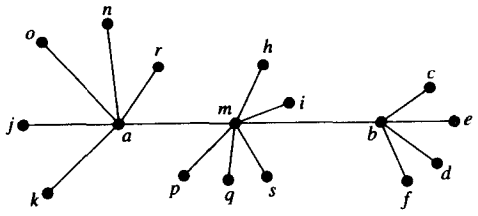
图9-50 树及其4棵3星树



c)



d)



e)

图9-50 （续）

现在，分析构造的3星树的路径代价。

(1) 对于最终的路径代价中的每条边(v, a)、(v, b)以及(v, m)计数$(n-1)$次。对于$V_a \cup V_b \cup V_m$中的每个结点v，唯一要说的是边的权值不大于$dt(v, P)$。

(2) 对于$V_{am} \cup V_{bm}$中每个结点，可以讲得更多。注意有4棵可能的3星树。V_{am}中的所有结点连接到结点a，或者连接到结点m。我们并不知道哪种情况更好，参见图9-51。假设v是最初连接到a和m之间的x结点，那么，得到下面的两个不等式：

$$w(v, a) \leqslant w(v, x) + w(a, x) \tag{9-17}$$

$$w(v, m) \leqslant w(v, x) + w(x, m) \tag{9-18}$$

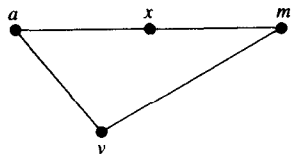


图9-51 结点v或者a或者到m的连接

因为$w(v, x) \leqslant dt(v, P)$，以及$w(a, x) + w(x, m) \leqslant dt(a, m)$，根据式（9-17）与式（9-18）的和，可以得到

$$w(v, a) + w(v, m) \leqslant 2dt(v, P) + dt(a, m)$$

或者

$$\frac{w(v,a)+w(v,m)}{2} \leqslant dt(v,P) + \frac{dt(a,m)}{2} \tag{9-19}$$

进而，可以得到：

$$\sum_v w(v,a) + \sum_v w(v,m) = \sum_v (w(v,a) + w(v,m)) \tag{9-20}$$

因此，$\sum_v w(v,a)$ 或者 $\sum_v w(v,m)$ 小于 $\sum_v \frac{w(v,a)+w(v,m)}{2}$。在$V_{am}$或$V_{bm}$中最多有$n/6$个结点。令$v$为$V_{am} \cup V_{bm}$中的结点，上面的推理得出在4棵3星树之一，从$V_{am} \cup V_{bm}$中所有结点出发的边权值之和不大于：

$$\frac{n}{6}\sum_{v}\left(dt(v,P)+\frac{1}{2}(dt(a,m)+dt(b,m))\right)$$

(3) 对于每条边(a, m)或(b, m)，在最终的路径代价中计数不多于$(n/2)(n/2) = (n^2/4)$次。

总之，在4棵3星树之一，其路径代价记为RC(3星，G)，满足下面的公式：

RC(3星，G)

$$\leqslant (n-1)\sum_{v}dt(v,P)+\left(\frac{n}{12}\right)(dt(a,m)+dt(b,m))+\left(\frac{1}{4}\right)n^2w(P)$$

$$\leqslant n\sum_{v}dt(v,P)+\left(\frac{1}{3}\right)n^2w(P)$$

由式（9-16），可以得到：

$$RC(3星，G)\leqslant \frac{3}{2}mrcst(G)$$

因此，证明了对于误差率 = 1/2，存在一棵最小路径代价生成树的路径代价不大于3/2的3星树。

上面证明了3星树的存在性。问题是如何找到这棵3星树，下面将加以解释。

想像在所有的n个结点中选取3个结点，表示为a、b和c。这三个结点是3星树中仅有的内部结点。假如已知一个3元组(i, j, k)，其中i，j，k是正整数，且$i+j+k = n-3$。这是说将连接i个结点到a，j个结点连接到b，k个结点连接到c，如图9-52所示。

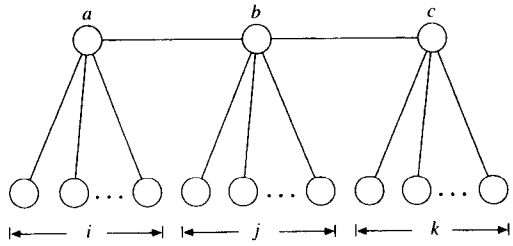


图9-52 有$(i+j+k)$个叶子结点的3星树

接下来的问题是：哪i个结点应该与结点a连接，哪j个结点应该与结点b连接，哪k个结点应该与结点c连接？为解决这个问题，必须解决二分图的匹配问题。在一个二分图中，有两个结点的集合，表示为X和Y。在这里，原始输入图是$G(V, E)$。对于二分图，$X = V-\{a, b, c\}$，Y包含i份的结点a，j份的b，k份的c。在X中的结点与Y中结点之间边的权值可以在原始输入距离矩阵中找到。在二分图上解决最小完美匹配问题（minimum perfect matching problem）。如果X中的结点v与Y中的结点u匹配，那么在3星树中v将连接到u。如果结点a，b和c以及整数i，j和k确定了，那么这棵3星树就是最优的3星树。这棵3星树的总路径代价可以相当容易地找到。叶子结点与内部结点之间的边计数了恰好$(n-1)$次。a和b之间的边计数了$(i(j+k)+j(i+k))$次，b和c之间的边计数了$[j(i+k)+k(i+j)]$次。叶子结点与内部结点之间的每条边计数了$(n-1)$次。

求解总路径代价不超过最小代价路径树3/2的3星树算法表示如下：

算法9-10 求解总路径代价不超过最小路径代价树3/2的3星树算法

输入：一个所有的边带权值，且所有权值满足三角不等式的完全图$G(V, E)$。

输出：总路径代价不超过G的最小路径代价树3/2的3星树。

令$RC = \infty$。

For 所有(a, b, c)，其中a、b和c是从V中挑选出 do

For 所有的(i, j, k)，其中$i+j+k = n-3$，i，j和k都是正整数 do

令$X = V-\{a, b, c\}$，Y包含i份的a、j份的b和k份的c。

在X和Y之间执行一个完美最小二分匹配。如果结点v在匹配中连接a、b或c，将这个结点v分别与a、b或c连接。这就建立了一棵3星树。

计算这棵3星树的总的路径代价Z。
如果Z小于RC，使$RC = Z$。
令这棵3星树为最优的3星树。

对上面算法的时间复杂度分析如下：挑选a，b和c有$O(n^3)$种可能的方法。挑选i，j和k有$O(n^2)$可能的方法。完美最小二分匹配问题可以在$O(n^3)$时间内解决。因此，解决近似3星树的时间复杂度为$O(n^8)$。

合适的k星树可以用相同的方式解决。我们不打算深入此细节。一般来讲，求解总的路径代价不超过最小路径代价树$(k+3)/(k+1)$的k星树的时间复杂度是$O(n^{2k+2})$。注意，$k=\left\lceil\frac{2}{E}-1\right\rceil$。因此，无论$E$多小，总有一个产生误差在该误差率界限中的$k$星树的多项式近似算法。总之，存在一个最小路径代价生成树问题的PTAS。

9.12 NPO完全性

在第8章中讨论了NP完全性的概念。如果一个判定问题是NP完全的，那么在多项式时间内解决是极不可能的。

在本节中，将介绍NPO完全性。将证明，如果一个优化问题是NPO完全的，那么极不可能存在一个多项式的近似算法，它产生一个有常数误差率的近似解。换句话说，NPO完全类问题表示不可能有好的近似算法的一类优化问题。

首先，定义NPO问题的概念。回顾一下，NP集由判定问题组成。NPO（non-deterministic polynomial optimization）集是由优化问题组构成的。现在处理优化问题，对可行的最优解感兴趣。如果一个优化问题在NPO中，那么该问题的可行解可以通过猜测和验证找出。假设该问题有可行解，那么猜测阶段总是会正确地对它定位。当然还应假设验证花费多项式级的步数。值得注意的是猜测阶段仅产生一个可行解，该可行解不必是最优的。

例如，旅行商问题是一个NPO问题，因为它是一个最优化问题，我们总是可以猜测一个回路并输出该回路的长度。当然，不能保证这个猜测的输出是最优解。

现在，定义NPO完全性的概念。首先定义规约，称为严格规约（strict reduction），如图9-53所示，定义如下：

已知NPO问题A和B，称(f, g)是从A到B的一个严格规约，如果

(1) 对于A中的每个实例x，$f(x)$是B中的一个实例。

(2) 对于B中$f(x)$的每个可行解y，$g(y)$是A中的一个可行解。

(3) $g(y)$相对于x的最优绝对误差小于等于y相对于B中$f(x)$的最优绝对误差。也就是$|g(y)-OPT_A(x)|\leqslant|y-OPT_B(f(x))|$。

在定义了NPO问题和严格规约之后，可以定义NPO完全问题了。一个NPO问题是NPO完全的，如果所有的NPO问题严格规约到该问题。

可以看出，如果问题A严格规约为问题B，并且B有一个相对于最优的误差小于ε的近似算法，那么可以用它来构建A的近似算法，其误差可以保证小于ε。如果一个最优问题A严格规约为一个优化问题B，那么事实上问题B有一个具有常数性能比的近似算法，这隐含着问题A也有一个具有常数性能比的近似算法。因此，

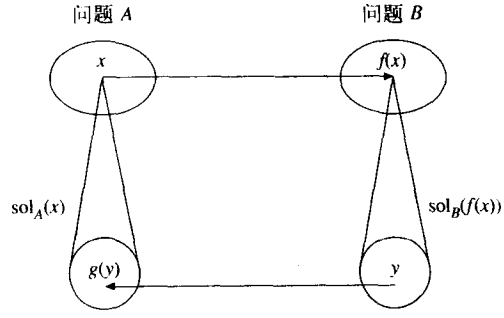


图9-53 严格规约的概念

如果一个NPO完全问题有任一常数比的近似算法，那么所有的NPO问题都有常数比的近似算法。因此，确定了问题A的难度。

在第8章中，介绍了第一个正式的NP完全问题：可满足性问题。在本节将介绍带权的可满足性问题（weingted satisfiability problem），并描述为什么带权可满足性问题是NPO完全的。

最小（最大）带权可满足性问题（minimum (maximum) weingted satisfiability problem）定义如下：已知布尔公式BF，其中每个变量x_i分配一个正的权值$w(x_i)$。我们的目标是找到满足BF所有变量的真值赋值，并使得下式最小（最大）

$$\sum_{x_i \text{为真}} w(x_i)$$

我们将忽略带权可满足性问题的NPO完全性的形式证明。正如在第8章所介绍的，应该举一些例子来说明最优化问题是如何严格规约到带权可满足性问题的。

例如，考虑找出在两个数a_1和a_2中的最大数问题。这个例子的严格规约$f(x)$如下：对于a_1（及a_2），将$x_1(x_2)$与$a_1(a_2)$相联系，$f(x)$将a_1（及a_2）映射到最大带权可满足问题的一个问题实例如下：

$$\text{BF}: (x_1 \vee x_2) \wedge (-x_1 \vee -x_2)$$

对于$i = 1, 2$，令$w(x_i) = a_i$。对于每个满足BF的真值指派，的确每个变量都是真的。如果$a_1(a_2)$是最大，那么指派给$x_1(x_2)$的真值指派为真。

函数$g(x)$定义如下

$g(x_1, -x_2) = a_1$

$g(-x_1, x_2) = a_2$

可见$f(x)$和$g(x)$构成了最大化问题与最大带权可满足性问题之间的一个严格规约。

例：最大独立集问题

最大独立集问题定义如下：已知图$G = (V, E)$，找出$V' \subseteq V$，使得V'是独立的，且$|V'|$最大。如果对于所有的$u, v \in S$，边$(u, v) \notin E$，那么称顶点集S是“独立的”。

参见图9-54。

这个例子的严格规约$f(x)$如下：

1. 对于每个顶点i，其邻接点为$j_1, j_2, \cdots, j_k$，f将它映射到子句

$x_i \rightarrow (-x_{j_1} \& -x_{j_2} \& \cdots \& -x_{j_k})$

2. 对于每个i，$w(x_i) = 1$。

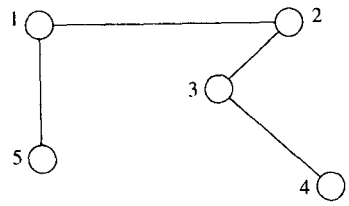


图9-54 最大独立集问题的例子

图9-54中的实例变换为

$$\begin{aligned}\phi = &(x_1 \rightarrow (-x_5 \& -x_2)) \\ &\&(x_2 \rightarrow (-x_1 \& -x_3)) \\ &\&(x_3 \rightarrow (-x_2 \& -x_4)) \\ &\&(x_4 \rightarrow (-x_3)) \\ &\&(x_5 \rightarrow (-x_1))\end{aligned}$$

函数g定义为：$g(X) = \{i | x_i \in X \text{且} x_i \text{为真}\}$。

可见对于满足ϕ的每个真值指派X，$g(X)$是一个独立集。在最大独立集与最大带权可满足性问题之间的严格规约就建立起来了。

证明NPO完全性的例

从上面的例子中，读者应该确信总是可以成功将任何NPO问题变换为带权可满足性问题

(WSAT)。这是我们本质上讲的“WSAT是NPO完全的”，例如，所有的NPO问题都可以严格规约到它。

现在证明一些问题是NPO完全的。应该提醒读者，当想证明一个问题A是NPO完全的，通常在下面两步内证明：

(1) 首先证明A是一个NPO问题。

(2) 然后证明某个NPO完全问题严格规约为A。

注意直到现在，只有一个NPO完全性问题：带权可满足性问题。为产生更多的NPO完全性问题，必须从它开始。也就是说，我们应该尽力证明带权可满足性问题严格规约到我们感兴趣的问题。

例：0-1整数规划

0-1整数规划问题（zero-one integer programming）定义如下：已知一个$m*n$的整数矩阵A，整数m矢量b，正整数n矢量c，找一个满足$Ax \geqslant b$，且使得cx最小的0-1 n矢量x。

当然，容易看出这个问题是一个NPO问题。对于任何非确定性猜测解X，可以在多项式时间内验证是否$Ax \geqslant b$，且对cx的验证也可在多项式级时间内计算。

在证明0-1整数规划问题的NPO完全性之前，首先要介绍最大或最小，带权3可满足性问题(W3SAT)。最大（最小）带权3可满足性问题定义（maxmum (minimum) weingted 3-satisfiability problem）如下：已知布尔公式ϕ是由子句所组成，其中每个子句只包含三个文字，以及与x_i有关的正整数权值$w(x_i)$，$i = 1, 2, 3$。找到满足ϕ的指派$\tau(x_i)$，如果存在这个指派，那么它将最大（最小）化下式

$$\sum_{\tau(x_i)\text{为真}} w(x_i)$$

令$w(x_1) = 3$，$w(x_2) = 5$，$w(x_3) = 2$。考虑下面的公式

$$\begin{aligned}&(-x_1 \vee -x_2 \vee -x_3)\\ &\&(-x_1 \vee -x_2 \vee -x_3)\\ &\&(-x_1 \vee -x_2 \vee x_3)\\ &\&(x_1 \vee x_2 \vee x_3)\end{aligned}$$

有4个指派满足上面的公式：

$(x_1, -x_2, x_3)$带权值为5

$(x_1, -x_2, -x_3)$带权值为3

$(-x_1, x_2, x_3)$带权值为7

以及$(-x_1, x_2, -x_3)$带权值为5。

指派$(-x_1, x_2, x_3)$ $((x_1, -x_2, -x_3))$是最大（最小）W3SAT问题的解。最大和最小W3SAT问题都是NPO完全的。为了证明0-1整数规划问题是NPO完全的，将证明最小W3SAT可以严格规约到它。

将最小W3SAT问题转换为0-1整数规划问题的函数f如下：

(1) 在最小W3SAT中的每个变量x对应0-1整数规划问题中的一个变量x。

(2) 变量值1代表“TRUE”，值0代表“FALSE”。

(3) W3SAT中的每个子句转换为一个不等式。运算符“or”转换为“+”，$-x$转换为$1-x$。由于整个子句必须为真，所以应使它“$\geqslant 1$”。

(4) 令$w(x_i)$（$i = 1, 2, 3$）表示变量x_i的权值，那么将下面的函数最小化：

$$\sum_{i=1}^{3} w(x_i)x_i$$

权值为$w(x_1)=3$，$w(x_2)=5$和$w(x_3)=5$的最小W3SAT问题$(x_1 \vee x_2 \vee x_3)\&(-x_1 \vee -x_2 \vee x_3)$可转换为下面的0-1整数规划问题：

最小化 $3x_1+5x_2+5x_3$，其中$x_i=0$，1

约束为 $x_1+x_2+x_3 \geqslant 1$，$1-x_1+1-x_2+x_3 \geqslant 1$

最小W3SAT问题解的一个为真的指派$(-x_1, -x_2, x_3)$对应于矢量(0，0，1)，该矢量是0-1整数规则问题的一个解。因此，带权3可满足性问题严格规约为0-1整数规划问题。所以，0-1整数规划问题是NPO完全的。

例：图上的最长和最短哈密顿回路问题

已知一个图，一条哈密顿回路是遍历图中所有结点仅一次的回路。图上的最长（最短）哈密顿回路问题（longest (shortest) Hamiltonian cycle problem）定义为：已知一个图，找到这个图的最长（最短）哈密尔顿回路。这两个问题都是NP难的。最短哈密尔顿回路问题也称为旅行商问题。下面将证明这两个问题都是NPO完全的。显然这两个问题都是NPO问题，NPO完全性可以通过证明最大或最小W2SAT问题严格规约到它来建立。

首先，已知W3SAT问题的实例ϕ，将证明：

1. 存在将ϕ映射到图$f(\phi)$上的函数f，使得W3SAT是可满足的，当且仅当$f(\phi)$有一条哈密顿回路。

2. 存在将图$f(\phi)$上的哈密尔顿回路C映射到对ϕ的可满足的真值指派$g(C)$的函数g。

3. C的代价与$g(C)$的代价相等。

最后一个性质保证了$g(C)$对ϕ的最优解的绝对误差等于C相对于$f(\phi)$最优解的绝对误差。也就是，$|g(C)-OPT(\phi)| = |C-OPT(f(\phi))|$。因此，$(f, g)$是一个严格规约。

现在说明W3SAT问题实例转换到图的例子。

(1) 对于每个子句C_i，有一个C_i分支，如图9-55所示。边e_1，e_2和e_3对应于子句C_i中的三个文字。例如，如果C_i包含$-x_1$、x_2和$-x_3$，那么e_1，e_2和e_3分别对应于$-x_1$、x_2和$-x_3$。C_i分支有一个性质，对于任何从$c_{i,1}$到$c_{i,4}$的哈密顿路径，在e_1、e_2，e_3中至少有一条边不会被遍历，如图9-56所示。后面将解释未遍历边的意义。

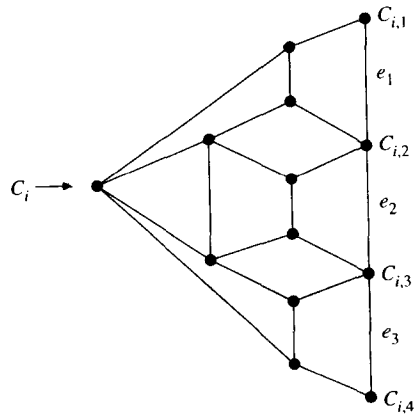


图9-55 对应于一个3子句的C分支

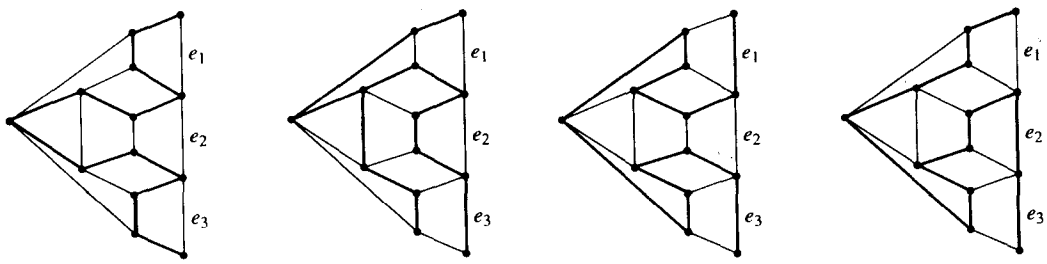


图9-56 e_1，e_2，e_3中至少有一条边不被遍历的C分支

(2) 对于每个变量V_i，有一个V_i分支，如图9-57所示。对于一条哈密顿路径，x_i或者$-x_i$会被遍历。

(3) 通过配件H连接V分支到C分支，如图9-58所示。如果图$G=(V, E)$包含如图9-58a所示的配件$H=(W, F)$，使得$V-W$中没有结点与$W-\{u, v, u', v'\}$中的任何结点相邻，那么看出G中的任何哈密顿回路必定遍历图9-58b或者图9-58c中的H。

(4) 对于一个已知有子句C_1，C_2，…，C_m的布尔公式，对应这些子句的C分支以边$(c_{i,4}, c_{i+1,1})$ $(i=1, 2, \cdots, m-1)$串联。

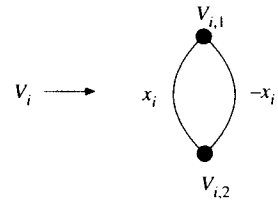
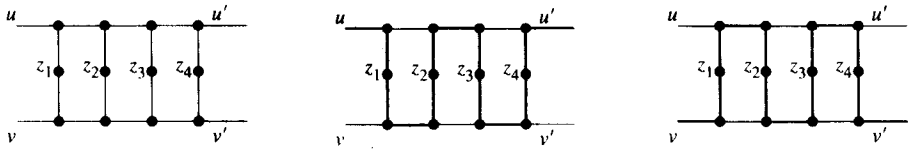


图9-57 对应于一个变量的组件



a) b) from u to u' c) from v to v'

图9-58 配件

(5) 对于一个已知有变量x_1，x_2，…，x_n的布尔公式，它们对应的V_i分支以边$(v_{i,2}, v_{i+1,1})$ $(i=1, 2, \cdots, n-1)$串联。

(6) 有两条特殊的边$(c_{1,1}, v_{1,1})$和$(c_{m,4}, v_{n,2})$。

(7) 如果在子句C中出现$x(-x)$，那么左(右)边对应于变量x的V分支的$x(-x)$，将会连接到对应于C分支中对应于子句C的变量x的边。

一条哈密顿回路的例子如图9-59所示。注意任何哈密顿回路必须包含从$v_{1,1}$到$c_{1,1}$的路径。从$c_{1,1}$有两个选择，直接到$c_{1,2}$或者其他的路径。在本例中，并没有遍历C_1分支中对应于x_1的边。在整个回路中，C_1分支中的x_1和x_3以及C_2分支中的$-x_2$都没有被遍历，这对应指派$(x_1, -x_2, x_3)$。

通常有下面的规则：在一条哈密顿回路中，如果在C分支中对应$x_i(-x_i)$的边没有被遍历，那么在相应的指派中$x_i(-x_i)$为真。

根据上面的规则，如果在回路中子句C_1中对应于x_1的边没有被遍历，那么在赋值中，置x_1为真。读者可能想知道：子句C_2中对应于$-x_1$的边是否被遍历呢？确实是这种情况；否则，将会产生矛盾，并且没有相应的指派。这样可通过验证在V_1分支中的$-x_1$连接看出。可以证明对于G中的任何哈密尔顿回路，如果在某个C_j分支中对应于$x_i(-x_i)$的边未遍历，那么在这个C_k分支中对应于$-x_i(x_i)$的边会被遍历。这意味着G中的每一条哈密顿回路有ϕ的一个相应指派。

还应该指派G中边的权值。指派$w(x_i)$为V_i分支的左边，$i=1, 2, \cdots, n$，其他的边为0。可以证明图G有一条权值为W的哈密顿回路，当且仅当布尔公式ϕ有权值W的可满足指派。

我们证明了在最大或最小W2SAT问题中存在一个严格规约，将其规约为图上最长或最短哈密顿回路问题。由于最大和最小W3SAT问题的NPO完全性，现在最长和最短哈密顿回路问题的NPO完全都建立起来了。

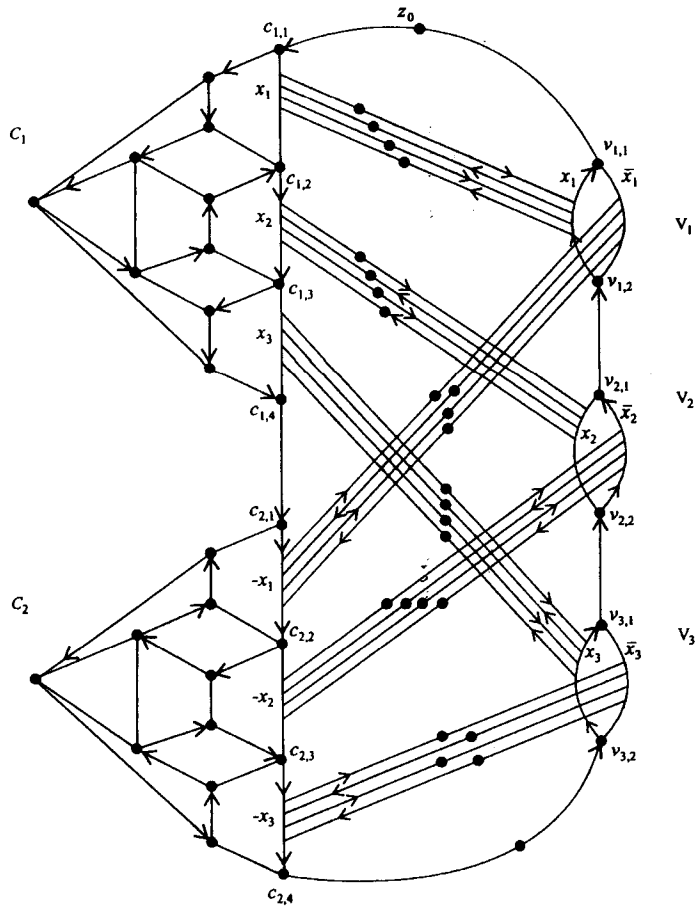


图9-59 指派为$(-x_1, -x_2, x_3)$的$(x_1 \vee x_2 \vee x_3)$和$(-x_1 \vee -x_2 \vee -x_3)$

9.13 注释与参考

欧几里得旅行商问题的NP难度是由文献Papadimitriou(1977)所证明。文献Rosenkrantz, Stearns and Lewis(1977)证明了欧几里得旅行商问题的近似长度可以小于最短回路的两倍。文献Christofides(1976)讲述了最小生成树、匹配以及旅行商问题，以得到长度在最优解3/2内的欧几里得旅行商问题的近似算法。在9.3节和9.4节中讨论的瓶颈问题最初分别在文献Parker and Pardin(1984)，Hochbaum and Shmoys(1986)中出现。装箱近似算法在文献Johnson(1973)中出现。直线m中心问题的近似算法由文献Ko，Lee and Chang(1990)给出。

术语NPO完全性首先出现在文献Ausiello，Crescenzi and Protasi(1995)中，但是这个概念最初在Orponen and Mannila(1987)中介绍。在文献Orponen and Mannila(1987)中，0−1整数规划问题、旅行商问题和最大带权3可满足性问题都证明是NPO完全的。最长哈密顿路径问题的NPO完全的证明可以在文献Wu，Chao and Lee(1998)中找到。

旅行商问题很难有好的近似算法，这在文献Sahni and Gonzalez(1976)中有证明。他们证明了如果旅行商问题有任何常数的近似率，那么NP = P。注意这是对于一般情况。对于特殊情况是有好的近似算法的。

求解0/1背包问题PTAS的一个聪明方法由Ibarra and Kim(1975)发现。Baker发展了平面图

上的几个问题的PTAS(Baker, 1994)。如果将其限制在平面（Grigni, Koutsoupias, Papadimitriou, 1995）或者欧几里得中（Arora, 196），那么旅行商问题是有PTAS的。最小路径代价生成树问题的PTAS方案由文献Wu, Lancia, Bafna, Chao, Ravi and Tang(2000)给出。另一个感兴趣的PTAS方案可以在文献Wang and Gusfield(1997)中找到。

还有另一类问题，记为MAX SNP完全问题，由Papadimitriou and Yannakakis(1991)提出。这类问题可以由一个严格句法形式Fagin(1974)表示，它有一个常数比率的近似算法。已经证明这类的任何问题可以有一个PTAS，当且仅当NP = P。有几个问题，如MAX 3SAT问题、矩阵旅行商问题、最大叶子生成树问题以及无序标记树问题，都被证明是这一类的。更多细节内容可参见Papadimitriou and Yannakakis(1991)；Papadimitriou and Yannakakis(1992)；Galbiati, Maffioli and Morzenti(1994)以及Zhang and Jiang(1994)。

9.14 进一步的阅读资料

对换排序问题的近似算法可以在文献Bafna and Pevzner(1998)中找到。对于多序列比对问题可参考文献Gusfield(1997)。文献Horowitz and Sahni(1978)和Papadimitriou and Steiglitz(1982)都讨论了近似算法。这个主题的回顾还可以在文献Garey and Johnson(1976)中找到。近几年出版了许多近似算法，其中的许多都与NP完全性相关。推荐下面的文章进一步阅读：Alon and Azar(1989)；Baker and Coffman Jr.(1982)；Bruno, Coffman and Sethi(1974)；Cornuejols, Fisher and Nemhauser(1977)；Friesen and Kuhl(1988)；Hall and Hochbaum(1986)；Hochbaum and Maass(1987)；Hsu(1984)；Johnson(1974)；Johnson, Demars, Ullman, Garey and Graham(1987)；Krarup and Pruzan(1986)；Langston(1982)；Larson(1984)；Mehlhorn(1988)；Moran(1981)；Murgolo(1987)；Nishizeki, Asano and Watanabe(1983)；Reghavan(1988)；Sahni(1977)；Sahni and Gonzalez(1976)；Tarhio and Ukkonen(1988)；Vaidya(1988)以及Wu, Widmayer and Wong(1986)。

最近的研究成果可以在以下文献中找到：Agarwal and Procopiuc(2000)；Akutsu and halldorsson(1994)；Akutsu and Miyano(1997)；Akutsu, Arimura and Shimozono(2000)；Aldous(1989)；Amir and Farach(1995)；Amir and Keselman(1997)；Amir and Landau(1991)；Arkin, Chiang, Mitchell, Skiena and Yang(1999)；Armen and Stein(1994)；Armen and Stein(1995)；Armen and Stein(1996)；Armen and Stein(1998)；Arora, Lund, Motwani, Sudan and Szegedy(1998)；Arratia, Goldstein and Gordon(1989)；Arratia, Martin, Reinert, and Waterman(1996)；Arya, Mount, Netanyahu, Silverman and Wu(1998)；Avrim, Jiang, Li, Tromp and Yannakakis(1991)；Baeza-Yates and Perleberg(1992)；Baeza-Yates and Navarro(1999)；Bafna and Pevzner(1996)；Bafna, Berman and Fujito(1999)；Bafna, Lawler and Pevzner(1997)；Berman, Hannenhalli and Karpinki(2001)；Blum(1994)；Blum, Jiang, Li, Tromp and Yannakakis(1994)；Bonizzoni, Vedova and Mauri(2001)；Breen, Waterman and Zhang(1985)；Breslauer, Jiang and Jiang(1997)；Bridson and Tang(2001)；Cary(2001)；Chang and Lamp(1992)；Chang and Lawler(1994)；Chen and Miranda(2001)；Chen(1975)；Christos and Drago(1998)；Chu and La(2001)；Clarkson(1994)；Cobbs(1995)；Czumaj, Gasieniec, Piotrow and Rytter(1994)；Dinitz and Nutov(1999)；Drake and Hougardy(2003)；Esko(1990)；Even, Naor and Zosin(2000)；Feige and Krauthgamer(2002)；Frieze and Kannan(1991)；Galbiati, Maffioli and Morrzenti(1994)；Galil and Park(1990)；Goemans and Williamson(1995)；Gonzalo(2001)；Gusfield(1994)；Hochbaum and Shmoys(1987)；

Ivanov(1984); Jain and Vazirani(2001); Jiang and Li(1995); Jiang, Kearney and Li(1998); Jiang, Kearney and Li(2001); Jiang, Wang and Lawler(1996); Jonathan(1989); Jorma and Ukkonen(1988); Kannan and Warnow(1995); Karkkainen, Navarro and Ukkonen(2000); Kececioglu(1991); Kececioglu and Myers(1995a); Kececioglu and Myers(1995b); Kececioglu and Sankoff(1993); Kececioglu and Sankoff(1995); Kleinberg and Tardos(2002); Kolliopoulos and Stein(2002); Kortsarz and Peleg(1995); Krumke, Marathe and Ravi(2001); Landau and Schmidt(1993); Landau and Vishkin(1989); Laquer(1981); Leighton and Rao(1999); Maniezzo(1998); Mauri, pavesi and Piccolboni(1999); Myers(1994); Myers and Miller(1989); Parida, Floratos and Rigoutsos(1999); Pe'er and Shamir(2000); Pevzner and Waterman(1995); Promel and Steger(2000); Raghavachari and Veerasamy(1999); Slavik(1997); Srinivasan(1999); Srinivasan and Teo(2001); Stewart(1999); Sweedyk(1995); Tarhio and Ukkonen(1986); Tarhio and Ukkonen(1988); Tong and Wong(2002); Trevisan(2001); Turner(1989); Ukkonen(1985a); Ukkonen(1985b); Ukkonen(1990); Ukkonen(1992); Vazirani(2001); Vyugin and V'yugin(2002); Wang and Gusfield(1997); Wang and Jiang(1994); Wang, Jiang and Gusfield(2000); Wang, Jiang and Lawler(1996); Wang, Zhang, Jeong and Shasha(1994); Wateman and Vingron(1994); Wright(1994); Wu and Manber(1992); Wu and Myers(1996)以及Zelikovsky(1993)。

习题

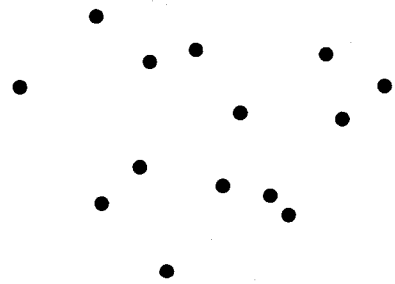
9.1 一个解决装箱问题的简单在线近似算法是将一件物品放入第i个箱子中，否则放入第$i+1$个箱子中。这个算法称为最先适应(FF)算法。证明FF算法得出的箱子数量不多于最优解箱子个数的两倍。

9.2 证明如果物品的序列看成是1/2，1/2n，1/2，1/2n，…，1/2（一共有4n−1项），那么FF算法实际上使用了真正所需箱子个数的两倍。

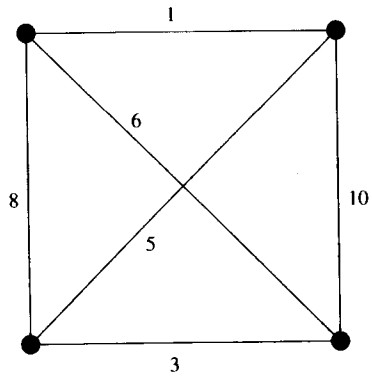
9.3 证明并不存在任何对于旅行商问题的多项式时间的近似算法，使得由近似算法产生的误差界限为$\varepsilon \cdot TSP$，其中ε是任意常数，TSP表示最优解。
（**提示**：证明哈密顿回路问题可以归约到这个问题。）

9.4 使用近似凸包算法求解下面点集的近似凸包。

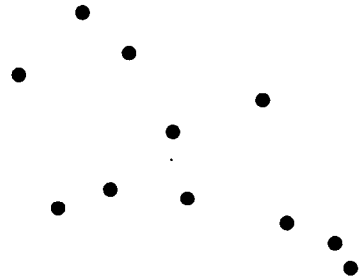
9.5 设有一个点集稠密地分布在一个圈上，使用近似欧几里得旅行商算法求解这个点集的近似回路，得到的结果是最优解吗？



9.6 考虑如下图所示方形中的四个点。使用本章所介绍的算法近似求解瓶颈旅行商问题，得到的这个解是最优解吗？



9.7 对直线m中心问题使用近似算法找出下面点集的直线2中心问题的近似解。



9.8 考虑下面的瓶颈最优问题。已知平面上的一个点集，找出k个点，使得在这k个点中最短距离最大，这个问题由Wang and Kuo(1988)证明是一个NP完全问题。设法找出解决这个问题的近似算法。

9.9 阅读文献Horowitz and Sahni(1978)关于调度独立任务近似算法的12.3节。对下面的调度问题应用最长处理时间(LPT)规则：有三台处理器和七项任务，其中任务时间为$(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (14, 12, 11, 9, 8, 7, 1)$。

9.10 编写一个实现旅行商问题近似算法的程序，以及求解旅行商问题最优解的分支限界算法的程序。比较它们的结果，总结出你的结论，使用这样的近似算法值得吗？

第10章 分摊分析

在本章中，我们将关注一系列操作的时间复杂度。假如考虑一系列操作OP_1，OP_2，…，OP_m，想确定这个操作序列可能花费的最长时间。人们的立即反映可能是考查每种操作OP_i的最坏情况时间复杂度。然而，操作序列OP_1，OP_2，…，OP_m可能的最长时间是每种操作OP_i花费的最坏时间t_i之和，并不总是正确。原因相当简单，我们并不希望最坏情况频繁地发生。

为了更加明确，想象我们每个月到银行存一次钱，只要银行里有存款，就可到百货公司去消费。现在立即就明白在某个月的消费依赖于前几个月的存款。如果在十月份花钱大手大脚，那么在十月之前的几个月应是非常节俭的。另外，十月以后，由于所有存款花完了，又必须再存钱，想奢侈地花钱又得等几个月了。

上面的讨论说明了一点，操作是相互关联的，不能假定它们是相互独立的。

“分摊”（amortized）意味着现在我们完成许多存款，使得在将来能够消费。为什么“分摊”与算法分析有联系呢？后面将会明白，在很多情况下，“分摊”意味着对数据结构所做的操作似乎浪费了时间。然而，对数据结构的这种操作在后面可以得到补偿。也就是采取这些行动之后，后面的事情将变得更容易些。我们年轻时努力地工作，就是在年老时收获回报。

10.1 使用势能函数的例子

对我们来说势能是一个熟知的概念。为了利用水的能量，我们必须提升水，使其“势能”变高。只有当水的势能达到一定时，才可利用水。我们在银行里存的钱也可看作是势能，只有在有了足够的存款之后才可能消费。

在本节中，将说明如何利用势能函数的概念进行分摊分析。考虑操作序列OP_1，OP_2，…，OP_m，其中每个OP_i由多个元素出栈和一个元素入栈组成。假如每个入栈和出栈花费一个时间单位，令t_i表示OP_i花费的时间，那么总的时间是

$$T = \sum_{i=1}^{m} t_i$$

每个操作的平均时间是

$$t_{ave} = \sum_{i=1}^{m} t_i / m$$

我们的任务是确定t_{ave}。

读者可以看出t_{ave}是不容易找出的。在介绍使用势能函数概念找出t_{ave}之前，先举一个例子：

i	1	2	3	4	5	6	7	8
S_1	1 push	1 push	2 pops 1 push	1 push	1 push	1 push	2 pops 1 push	1 pop 1 push

t_i	1	1	3	1	1	1	3	2
	$t_{ave} = (1 + 1 + 3 + 1 + 1 + 1 + 3 + 2)/8 = 13/8 = 1.625$							
i	1	2	3	4	5	6	7	8
S_2	1 push	1 pop 1 push	1 push	1 push	1 push	1 push	5 pops 1 push	1 push
t_i	1	2	1	1	1	1	6	1
	$t_{ave} = (1 + 2 + 1 + 1 + 1 + 1 + 6 + 1)/8 = 14/8 = 1.75$							
i	1	2	3	4	5	6	7	8
S_3	1 push	1 push	1 push	1 push	4 pops 1 push	1 push	1 push	1 pop 1 push
t_i	1	1	1	1	5	1	1	2
	$t_{ave} = (1 + 1 + 1 + 1 + 5 + 1 + 1 + 2)/8 = 13/8 = 1.625$							

读者一定注意到，对于每种情况都有$t_{ave} \leqslant 2$。实际上，t_{ave}是很难找出的。但我们能够证明t_{ave}确有一个上界为2。得到这个结论是因为每次OP_i后，不仅完成了一次操作，而且改变了栈中的内容，栈中元素数量增加或者减少。如果增加，那么执行多次出栈操作的能力也增加。在某种意义上，当一个元素入栈时，我们增加了势能，易于完成许多出栈操作。另一方面，由于从栈中弹出一个元素，就以某种方式减少了势能。

令$a_i = t_i + \phi_i - \phi_{i-1}$，其中$\phi_i$表示操作$OP_i$之后栈的势能函数。这样，$\phi_i - \phi_{i-1}$是势能的改变。那么

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} t_i + \sum_{i=1}^{m} (\phi_1 - \phi_{i-1})$$

$$= \sum_{i=1}^{m} t_i + \phi_m - \phi_0$$

如果$\phi_m - \phi_0 \geqslant 0$，那么$\sum_{i=1}^{m} a_i$代表$\sum_{i=1}^{m} t_i$的一个上界。也就是

$$\sum_{i=1}^{m} t_i / m \leqslant \sum_{i=1}^{m} a_i / m \tag{1}$$

现在直接定义ϕ_i为第i次操作后栈中元素数，容易得到$\phi_m - \phi_0 \geqslant 0$。假定在执行$OP_i$之前，栈中有$k$个元素，$OP_i$由$n$次出栈和1次入栈组成，即

$$\phi_{i-1} = k$$
$$\phi_i = k - n + 1$$
$$t_i = n + 1$$
$$a_i = t_i + \phi_i - \phi_{i-1}$$
$$= n + 1 + (k - n + 1) - k$$
$$= 2$$

从上面的等式，很容易证明

$$\sum_{i=1}^{m} a_i / m = 2$$

所以，根据(1)，$t_{ave}=\sum_{i=1}^{m}t_i/m\leqslant\sum_{i=1}^{m}a_i/m=2$。也就是$t_{ave}\leqslant 2$。

实际上，对于这种情况不必用势能函数推导上界2。当然不需要。对m次操作，至多有m次push入栈和m次pop出栈，因为出栈的数量不可能超过入栈的数量。因此，至多可能有$2m$个动作，平均时间不可能超过2。

尽管对于如此简单的例子可以不必使用势能函数，但对其他情况却需要使用，正如在下一节将要看到的。

最后，需要提醒读者$a_i\leqslant 2$不应解释为$t_i\leqslant 2$。t_i的值可以非常高，远超过2。然而，如果t_i在当前很高，那么在未来将是很小的。当前t_i很高的事实意味着栈中元素在将来要大幅地减少。或者用另一说法，如果它是高的，势能将会急剧降低。这与自然现象类似。如果要利用大量的水，那么资源的势能将会降低，将来就没有可利用的水了。

10.2 斜堆的分摊分析

在斜堆（skew heap）中，基本的操作称为“合并”（meld）。参见图10-1，为合并这两个斜堆，先将两棵斜堆的右路径合并成一个，然后将左分支连接到合并路径中的结点，如图10-2所示。在这一步之后，除了最低的结点，交换合并路径中的每个结点的左右孩子。

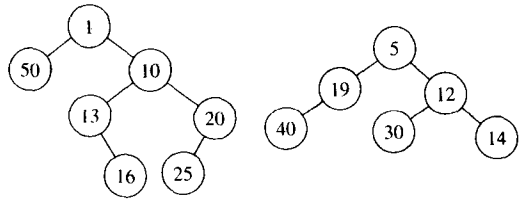


图10-1 两棵斜堆

图10-3显示了交换后的结果。正如所看到的，尽管不能保证右路径最短，但是合并后的斜堆趋向于有最短的右路径。然而，实现合并操作是很容易的，从分摊意义上，这能产生好的性能。

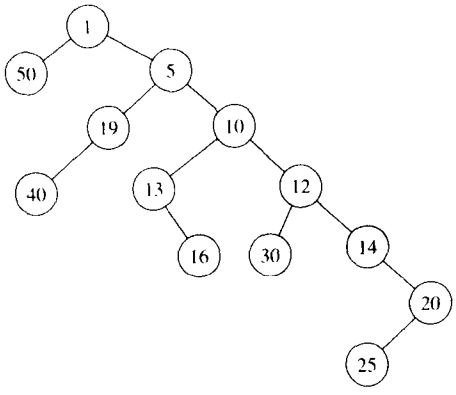


图10-2 图10-1中两棵斜堆的合并

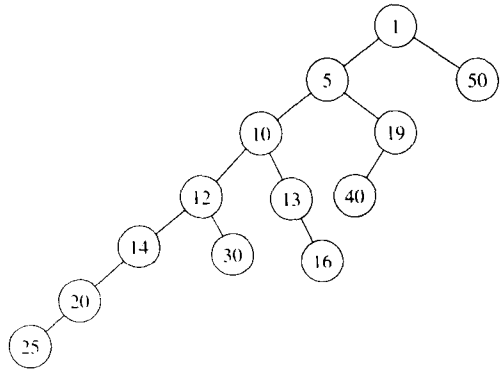


图10-3 图10-2中斜堆的最左、最右路径的交换

象斜堆这样的数据结构需要许多操作，实际上对斜堆可以实现以下的操作：

(1) find-min(h)：找出斜堆h中的最小者。

(2) insert(x，h)：将元素x插入斜堆h中。

(3) delete-min(h)：从斜堆h中删除最小者。

(4) meld(h_1，h_2)：合并两个斜堆h_1和h_2。

前三个操作通过合并即可完成。例如，对于插入元素x的操作，可将x看作只有一个元素的斜堆，然后合并这两个斜堆。类似地，当删除最小元素时，它总是斜堆的根，我们可以有效地将该斜堆拆分成两个斜堆，然后再合并新生成的两个斜堆。

由于所有的操作都基于合并操作，可以将一个由删除、插入和合并组成的操作序列看作一个合并操作序列。所以，只需分析合并操作。在分析前，注意合并操作一个有趣的性质：*如果在合并上花费很多时间，那么由于交换的影响，右路径将变得非常短。因此，合并操作并不再是非常耗时的，势能就简单地消失了。*

为了完成分摊分析，再次令

$$a_i = t_i + \phi_i - \phi_{i-1}$$

其中t_i是操作OP_i的耗时，ϕ_i和ϕ_{i-1}分别是操作OP_i之后与之前的势能。另外还有

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} t_i + \phi_m - \phi_0$$

如果$\phi_m - \phi_0 \geqslant 0$，那么$\sum_{i=1}^{m} a_i$看作任意$m$次操作所花费实际时间的上界。

现在的问题是定义一个好的势能函数。令$wt(x)$表示包括结点x的子孙数，重结点（heavy node）x是非根结点，且$wt(x)>wt(p(x))/2$，其中$p(x)$是x的父结点。反之，结点x是轻结点（light node）。

考虑图10-2所示的斜堆，结点的权值表示在表10-1中。

表10-1　图10-2中结点的权值

结点	权值	重/轻	结点	权值	重/轻
1	13		19	2	轻
5	11	重	20	2	重
10	8	重	25	1	轻
12	5	重	30	1	轻
13	2	轻	40	1	轻
14	3	重	50	1	轻
16	1	轻			

要注意每个叶结点必定是轻结点。另外，如果某个结点是轻的，那么至少与它的兄弟结点比较时，它没有太多的子孙。例如，在图10-2中的结点13和19都是轻结点。它们的兄弟结点是重结点。此树或该斜堆向许多重结点方倾斜。注意在图10-2中的斜堆，所有在右路径中的结点都是重结点，该路径是条长路径。另一方面，考虑图10-3，该斜堆的右路径没有重结点，它是条短路径。

如果某个结点是另一个结点的右（左）孩子，那么称该结点为右（左）结点（right(left) node）。右（左）重结点是右（左）结点为重的。*我们可以将斜堆的右重结点数作为势能函数。*因此，对于图10-2中的斜堆，势能是5，图10-3中的斜堆的势能是0。

现在调查一个非常重要的问题。合并前与后的势能是多少，或者合并前与后在右重结点数量间的差别有多大？

首先应注意，为了成为重结点，结点必须比它的兄弟结点有更多的子孙。因此，可以容易地明白，*在任何结点的孩子中，最多有一个是重的。*基于上面的陈述，在图10-4中可以看到连接到左路径中右重结点数与左路径中的轻结点数相关。

由于只有轻结点的兄弟结点可能是连接到左路径的重结点，连接到左路径的右重结点数总是小于或等于斜堆左路径重轻结点数。现在的问题是估计路径中轻结点数。

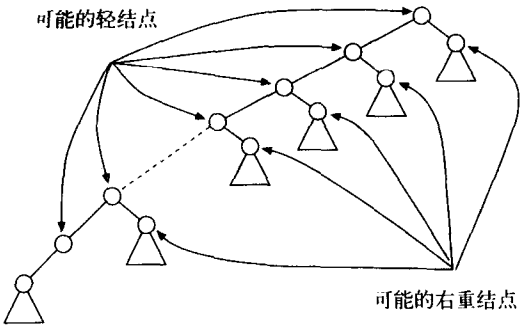


图10-4 可能的轻结点和重结点

为了估计路径中的轻结点数，要注意，如果x是一个轻结点，那么x的权值一定小于或等于其父结点$p(x)$的一半。如果y是x的子孙，且y是轻结点，那么y的权值一定小于或等于$p(x)$权值的四分之一。换句话说，考虑从结点x到y的路径中的轻结点序列，容易看到这些轻结点的权值快速地减少，而且，对斜堆的任何路径，此路径中的轻结点数一定小于$\lfloor \log_2 n \rfloor$。这意味着关联到左路径的右重结点数一定小于$\lfloor \log_2 n \rfloor$。

注意$a_i = t_i + \phi_i - \phi_{i-1}$，其中$t_i$是操作$OP_i$的耗时，$\phi_i$表示操作$OP_i$之后的势能。

t_i的值与右路径的长度有关，参见图10-5，令$K_1(K_2)$是在$h_1(h_2)$中右路径的重结点数，$n_1(n_2)$是$h_1(h_2)$中的结点数，$L_1(L_2)$等于$h_1(h_2)$中右路径上的轻结点数加重结点数。并且，t_i由等于$2 + L_1 + L_2$合并路径上的结点数确定。（其中的2是指h_1和h_2的根。）

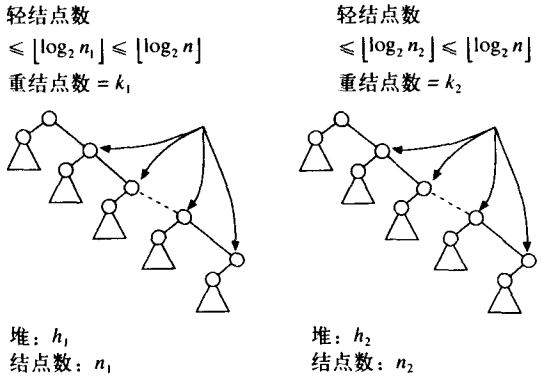


图10-5 两棵斜堆

所以，
$$
\begin{aligned}
t_i &= 2 + L_1 + L_2 \\
&= 2 + h_1\text{右路径中的轻结点数} \\
&\quad + h_1\text{右路径中的重结点数} \\
&\quad + h_2\text{右路径中的轻结点数} \\
&\quad + h_2\text{右路径中的重结点数} \\
&\leqslant 2 + \lfloor \log_2 n_1 \rfloor + K_1 + \lfloor \log_2 n_2 \rfloor + K_2 \\
&\leqslant 2 + 2\lfloor \log_2 n \rfloor + K_1 + K_2。
\end{aligned}
$$

其中$n = n_1 + n_2$。

现在计算a_i的值。在合并之后，右路径中的$K_1 + K_2$个右重结点消失了，而产生了小于$\lfloor \log_2 n \rfloor$关联到左路径的右重结点。所以

$$\phi_i - \phi_{i-1} \leqslant \lfloor \log_2 n \rfloor - K_1 - K_2 \qquad \text{对于} i = 1, 2, \cdots, m$$

那么，得到

$$
\begin{aligned}
a_i &= t_i + \phi_i - \phi_{i-1} \\
&\leqslant 2 + 2\lfloor \log_2 n \rfloor + K_1 + K_2 + \lfloor \log_2 n \rfloor - K_1 - K_2 \\
&= 2 + 3\lfloor \log_2 n \rfloor
\end{aligned}
$$

所以，$a_i = O(\log_2 n)$，且$\sum t_i/m = O(\log_2 n)$。

10.3 AVL树的分摊分析

在本节中，将对AVL树进行分摊分析。如果每个结点的子树高度差至多是1，那么该二叉树是一棵AVL树，图10-6显示一棵AVL树。正如所看到的，每个结点的子树高度差至多是1，在这种定义下此树是平衡的。

对于以v为根的子树T，表示为$H(T)$的高度定义成从根v到叶子的最长路径长度。令$L(v)$（$R(v)$）是以v为根的左（右）子树，那么对于每个结点v，定义其高度平衡因子$hb(v)$为

$$hb(v) = H(R(v)) - H(L(v))$$

对于一棵AVL树，$hb(v)$只等于0，+1或−1。

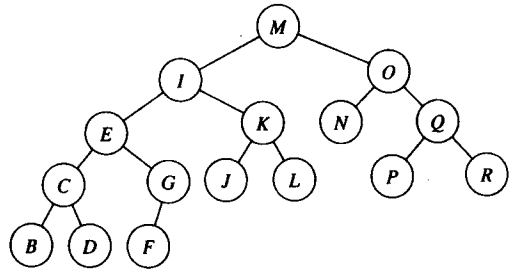


图10-6 一棵AVL树

对于图10-6的AVL树，一些结点的高度平衡因子如下：

$hb(M) = -1$　　$hb(I) = -1$

$hb(E) = 0$　　$hb(C) = 0$

$hb(B) = 0$　　$hb(O) = +1$

由于AVL树是二叉树，要往树中增添一项，那么该新的数据项将成为一个叶子结点。这个叶子结点会导致一条特殊的路径，其上所有结点的平衡因子都需改变。图10-7显示了图10-6中的AVL树所有结点的平衡因子。增添一项A，新树如图10-8所示。我们看到，新树不再是一棵AVL树，必须要调整平衡。将图10-8与图10-7做比较，看到只有M，I，E，C，B，A这条路径上的结点的平衡因子需要改变。

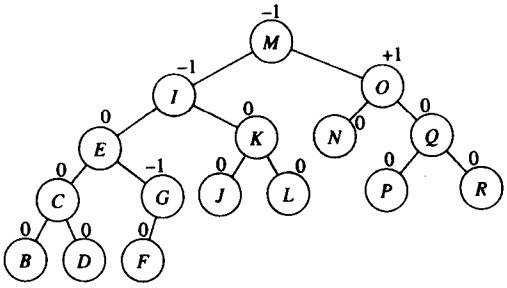


图10-7 标记高度平衡因子的AVL树

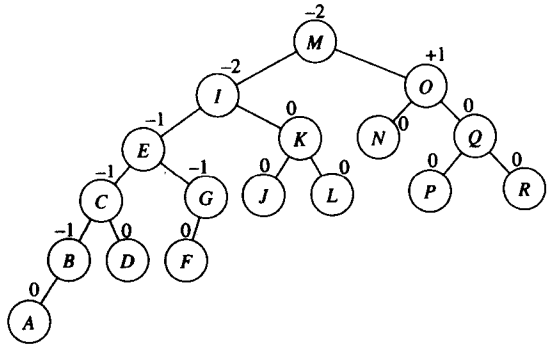


图10-8 增添A的新AVL树

令V_k为增添到AVL树中的新叶子结点，V_0，V_1，…，V_{k-1}，V_k是刚产生的路径，其中V_0是根。在插入前，令i是使得$hb(V_i) = hb(V_{i+1}) = \cdots = hb(V_{k-1}) = 0$最小的$i$。如果$i \geqslant 1$，那么结点$V_{i-1}$称为该路径的关键结点（critical node），V_i，…，V_{k-1}称为由V_k产生的关键路径（critical path）。

参见图10-7，如果新插入的结点连接到B，那么关键路径是E，C，B。将图10-7与图10-8比较，注意到结点E，C，B的平衡因子都从0改变为-1。假设增添一个新结点到结点R的右分支，那么Q和R的平衡因子从0改变为+1。

令T_0是一棵空AVL树，考虑向T_0进行m次插入的操作序列。令X_1表示在这m次插入过程中，平衡因子从0改变为+1或-1的总次数。现在的问题是找出X_1。下面的分摊分析将证明X_1小于或等于$2.618m$，其中m是在AVL树中数据元素的数量。

令L_i表示涉及第i次插入操作的关键路径长度，那么

$$X_1 = \sum_{i=1}^{m} L_i$$

下面证明，当一个新元素插入时，可能有三种情况：

情况1：不需要平衡调整。树的高度不增长，只“吸收”该结点而形成新树。

情况2：需要平衡调整。需要进行二次或一次旋转产生新的平衡树。

情况3：不需要调整。但树的高度增加了，只需在树的高度上记录该增加。

在说明这三种情况之前，令$Val(T)$表示树T中不平衡结点数，也就是$Val(T)$等于平衡因子不为0的结点数。

情况1：考虑图10-9，虚线表示插入新数据项。由于插入后仍然是AVL树，不需要做平衡调整。先前的树只简单地吸收该新结点，树的高度也没有改变。在这种情况下，在关键路径上所有结点的高度平衡因子的改变是从0到-1或$+1$。从而，关键结点的平衡因子的改变将从$+1$或-1到0。因此，

$$Val(T_i) = Val(T_{i-1}) + (L_i - 1)$$

情况2：参见图10-10，在这种情况下，需要调整平衡树。

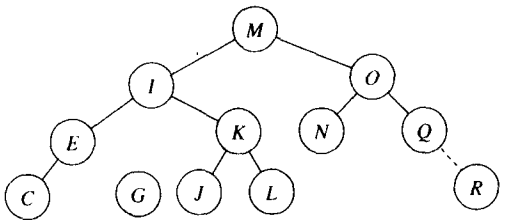


图10-9 插入后的情况1

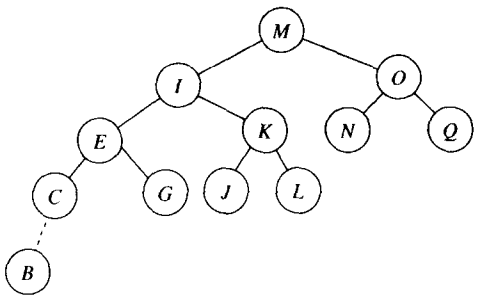


图10-10 插入后的情况2

需要某种两次和一次的旋转。对于如图10-10所示的情况，其平衡树如图10-11所示。很容易看到关键结点M的平衡因子从-1改变为0。但必须改变关键路径上所有结点的平衡因子从0到+1或-1，除了在关键路径中关键结点的孩子，在调整后它们也是平衡的。因此，在这种情况下，$Val(T_i) = Val(T_{i-1}) + (L_i - 2)$。

情况3：参见图10-12，在这种情况下，不需要调整平衡此树，但树的高度将会增加。

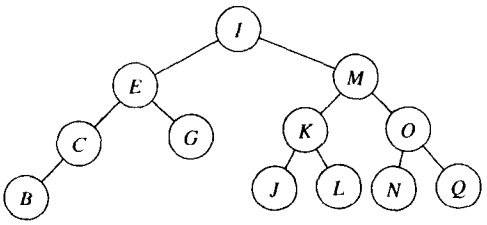


图10-11 图10-10的平衡树

容易看出

$$Val(T_i) = Val(T_{i-1}) + L_i$$

令X_2表示在情况1中需要吸收的结点数，X_3表示在情况2中需要一次旋转的次数，X_4表示在情况2中需要两次旋转的次数，X_5表示在情况3中增加的高度数，那么，利用上面三种情况描述的公式，得到

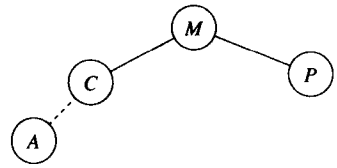


图10-12 插入后的情况3

$$Val(T_m) = Val(T_0) + \sum_{i=1}^{m} L_i - X_2 - 2(X_3 + X_4)$$

在上面的式子中，$-X_2$和$-2(X_3 + X_4)$分别对应情况1和情况2。$\sum_{i=1}^{m} L_i$ 是由于L_i项出现在所有三个式子中。

由于 $\sum_{i=1}^{m} L_i = X_1$，$Val(T_0) = 0$，又因为$X_2 + X_3 + X_4 + X_5 = m$，所以可得

$$\begin{aligned} X_1 &= Val(T_m) + X_2 + 2(X_3 + X_4) \\ &= Val(T_m) + (X_2 + X_3 + X_4 + X_5) + (X_3 + X_4 - X_5) \\ &= Val(T_m) + m + (X_3 + X_4 - X_5) \end{aligned}$$

现在，显然$X_3 + X_4 \leqslant m$，$X_5 \geqslant 0$，所以，

$$\begin{aligned} X_1 &\leqslant Val(T_m) + m + m \\ &= Val(T_m) + 2m \end{aligned}$$

这在Knuth的《计算机程序设计艺术》第3卷中已证明

$$Val(T_m) \leqslant 0.618m$$

因此，可得

$$\begin{aligned} X_1 &\leqslant Val(T_m) + 2m \\ &= 2.618m \end{aligned}$$

10.4 自组织顺序检索启发式方法的分摊分析

顺序检索（sequential search）是最简单的一种检索方法，有许多提高顺序检索性能的方法，其中之一称为自组织方法（self-organizing method）。

想象一栋学生宿舍，有几个相当受欢迎的学生，他们不停地接收电话。如果足够聪明的话，接线员会将这些学生的房间号列在表前，这样，会很快地回应电话。

自组织方法利用了这种思想，动态地向前移动频繁检索的项。也就是在任意检索完成后，将该项向前移动。有三种有名的自组织方法：

(1) **交换方法**（transpose method）：当查找到某项，将它与其前一项交换。因此，该方法向表的最前端移动一个位置。

(2) **移前方法**（move-to-the-front method）：当查找到某项，移动该项到表的最前端。

(3) **计数方法**（count method）：当查找到某项，增加该项的计数，向前移动保持存储表的递减次序。

习惯上称上述这些方法为启发式方法（heuristic method）。所以，这三种方法也分别称为交换启发式方法（transpose heuristics），移前启发式方法（move-to-the-front heuristics）和计数启发式方法（count heuristics）。

接下来，将通过几个例子说明这些启发式方法。我们总是假定以空表开始。

(1) 交换启发式方法（见表10-2）。

(2) 移前启发式方法（见表10-3）。

表10-2 交换启发式方法

查询	序列
B	*B*
D	*D B*
A	*D A B*
D	*D A B*
D	*D A B*
C	*D A C B*
A	*A D C B*

表10-3 移前启发式方法

查询	序列
B	*B*
D	*D B*
A	*A D B*
D	*D A B*
D	*D A B*
C	*C D A B*
A	*A C D B*

(3) 计数启发式方法（见表10-4）。

现在关注这些不同启发式方法的分摊分析。已知查询序列S，在启发式方法R下，查询序列S的代价定义为该序列在这个启发式方法R下，依次检索需要比较的总次数，将其表示为$C_R(S)$。假如有m次查询，如果能找到关于m的函数作为其代价这是最好的。正如以前所做的，变为找出此代价的上界。如果此序列数据项已优化为静态次序，那么此上界与该代价相关。当对于序列S数据项的次序已静态优化，那么其代价标记为$C_O(S)$。例如，假如有两个查询序列，令查询B的次数高于查询A的次数，那么对此查询序列的优化静态序列为$B\ A$。

表10-4 计数启发式方法

查询	序列
B	*B*
D	*B D*
A	*B D A*
D	*D B A*
D	*D B A*
A	*D A B*
C	*D A B C*
A	*D A B C*

在本节的剩余部分中，将对$C_R(S)$与$C_O(S)$进行比较。比较这两个代价将基于一些启发式方法具有的一个非常重要的属性，称为*两两独立属性*（pairwise independent property）。在介绍该属性前，定义字内比较（intraword comparison）为在不同的数据项间的比较，因此，字内比较是一种不成功的比较。字间比较（interword comparison）是在相似的数据项间做的比较，是一种成功的比较。现在介绍两两独立属性。

考虑移前启发式方法。有下面的查询序列：*C A C B C A*，该序列现在表10-5中。

接下来，将注意力集中在对A与B间的比较，上面的分析表示对于该序列，在A与B间的比较总次数是2。

假定只考虑由A和B组成的子序列，参见表10-6。在A和B间的比较总次数仍是2，这与表10-5一样。

换句话说，对于移前启发式方法，任何一对数据项P和Q之间的比较总次数只依赖

表10-5 查询序列C A C B C A 的移前启发式方法

查询	序列	(*A*, *B*) 比较
C	*C*	
A	*A C*	
C	*C A*	
B	*B C A*	✓
C	*C B A*	
A	*A C B*	✓

表10-6 查询序列A B A 的移前启发式方法

查询	序列	(*A*, *B*) 比较
A	*A*	
B	*B A*	✓
A	*A B*	✓

于P和Q在查询序列中的相对次序，这独立于其他项。也就是说，对任何序列S及所有的P与Q对，P与Q字内比较的总次数就是对只由P和Q构成的子序列S所作的字内比较次数，此属性称为两两独立属性。

例如，再次考虑查询序列$C\ A\ C\ B\ C\ A$。有三个不同的字内比较：(A, B)，(A, C)和(B, C)。可以单独考虑它们，并将它们加起来，如表10-7所示。

字内比较的总次数等于$0+1+1+2+1+2=7$，很容易检查这是对的。

表10-7 移前启发式方法的字内比较次数

查询	序列	C	A	C	B	C	A	
			0		1		1	(A, B)
		0	1	1		0	1	(A, C)
		0		0	1	1		(B, C)
		0	1	1	2	1	2	

令$C_M(S)$表示移前启发式方法的代价。接下来，将证明$C_M(S) \leqslant 2C_O(S)$。换句话说，对任何查询序列，移前启发式方法的代价不会超过优化静态查询序列代价的两倍。

首先由两个不同的数据项组成的S证明$C_M(S) \leqslant 2C_O(S)$。令S由a个A和b个B组成，其中$a<b$。所以，优化静态次序为$B\ \ A$。如果使用移前启发式方法，那么总的字内比较次数就是从$B\ \ A$到$A\ \ B$的改变数加上从$A\ \ B$到$B\ \ A$的改变数。因此，该数不会超过$2a$。令$Intra_M(S)$和$Intra_O(S)$分别表示在移前启发式方法下和优化静态次序下字内比较的总次数。对于任意由两个不同的数据项组成的S可得

$$Intra_M(S) \leqslant 2Intra_O(S)$$

考虑由多于两个数据项组成的任意序列S。由于移前启发式方法具有两两独立属性，易得

$$Intra_M(S) \leqslant 2Intra_O(S)$$

令$Intra_M(S)$和$Intra_O(S)$分别表示在移前启发式方法下和优化静态次序下字间比较的总次数。易得

$$Inter_M(S) = Inter_O(S)$$

所以，

$$Inter_M(S) + Intra_M(S) \leqslant Inter_O(S) + 2Intra_O(S)$$
$$C_M(S) \leqslant 2C_O(S)$$

人们也许想知道系数2是否可进一步紧致，下面将证明这是不可能进行的。

考虑查询序列$S=(A\ B\ C\ D)^m$，该序列的优化静态序列为$A\ B\ C\ D$，其总的比较次数$C_O(S)$为

$$\underbrace{1+2+3+4+1+2+3+4+\cdots}_{4m} = 10\cdot\frac{4m}{4} = 10m$$

在移前启发式方法下，总的比较次数$C_M(S)$确定如下：

$$\begin{array}{lllll}
C_M(S) = 1 & A & A & & \\
\quad +2 & B & B & A & \\
\quad +3 & C & C & B & A \\
\quad +4 & D & D & C & B & A \\
\quad +4 & A & A & D & C & B \\
\quad +4 & B & B & A & D & C \\
\quad \vdots & \vdots & & \vdots & & \\
\hline
10 + 4(4(m-1)) & & & & &
\end{array}$$

所以，$C_M(S) = 10 + 4(4(m-1)) = 10 + 16m-16 = 16m-6$。

如果从4到k增加不同的项数，可得

$$C_O(S) = \frac{k(k+1)}{2}m = \frac{m(k+1)k}{2}$$

和 $C_M(S) = \frac{k(k+1)}{2} + k(k(m-1)) = \frac{k(k+1)}{2} + k^2(m-1)$。

因此，

$$\frac{C_M(S)}{C_O(S)} = \frac{\frac{k(k+1)}{2} + k^2(m-1)}{\frac{k(k+1)}{2}\cdot m}$$

$$= \frac{1}{m} + 2\frac{k^2}{k(k+1)}\cdot\frac{m-1}{m} \to 2 \quad 由于k\to\infty及m\to\infty$$

这意味着系数2不能进一步紧致了。

上面的讨论表明移前启发式方法最坏情况的代价是2 $C_O(S)$。当认为$C_C(S)$是计数启发式方法的代价时，使用同样的推导可以证明$C_C(S) \leqslant 2C_O(S)$。遗憾的是，交换启发式方法不具有两两独立属性。所以，对于交换启发式方法的代价没有类似的上界。交换启发式方法不具有两两独立属性，可再次考虑相同的查询序列$C\ A\ C\ B\ C\ A$来加以说明。

对于交换启发式方法，如果独立考虑不同项的对，如表10-8所示的表示情形。

表10-8 交换启发式的字内比较次数

查询	序列	C	A	C	B	C	A	
			0		1		1	(A, B)
		0	1	1	[illegible]	0	1	(A, C)
		0		0	1	1		(B, C)
		0	1	1	2	1	2	

由于这样的计算，字内比较的总数为1 + 1 + 2 + 1 + 2 = 7。但这是不对的，正确的字内比较次数确定如下：

查询序列	C	A	C	B	C	A
数据次序	C	AC	CA	CBA	CBA	CAB
字内比较数	0	1	1	2	0	2

确切的字内比较次数应为1 + 1 + 2 + 2 = 6。这表明两两独立属性对于交换启发式方法不成立。

10.5 配对堆及其分摊分析

在本节中将介绍配对堆（pairing heap）及完成对该数据结构的分摊分析。很显然，配对堆可以简化成这样的一个扩展，它使得从配对堆中删除最小元素变得困难。尽管如此，在配对恶化到如此糟糕之后，分摊分析也能证明它能很快地恢复。

图10-13显示一个典型的配对堆。在介绍删除最小元素操作之后，将解释为什么称其为配对堆。

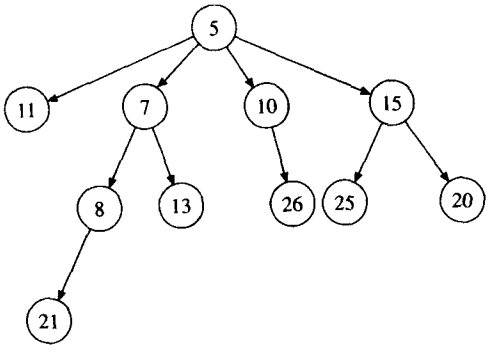


图10-13 一棵配对堆

注意，如同在任何堆中一样，一个结点的

父结点的关键字小于该结点本身。当然应使用一种良结构的数据结构来实现配对堆。对于配对堆，采用二叉树技术。图10-13中的配对堆的二叉树表示在图10-14中。如图10-14中所示，如果有的话，每个结点都关联到其最左后代和下一个右兄弟。如图10-13中，结点7将结点8作为其最左后代，结点10作为其下一个右兄弟。因此，在二叉树表示中，结点7与结点8和结点10关联，如图10-14所示。

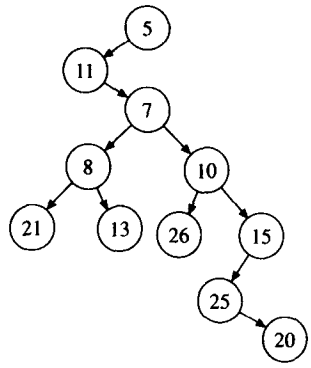


图10-14 图10-13中配对堆的二叉树表示

配对堆有七种基本的操作，它们是

(1) make heap(h)：建立一个名为h的新的空堆。

(2) find min(h)：找出配对堆h的最小者。

(3) insert(x，h)：向配对堆h中插入元素x。

(4) delete min(h)：从配对堆h中删除最小元素。

(5) meld(h_1，h_2)：合并两个配对堆h_1和h_2为新堆。

(6) decrease key(Δ，x，h)：将配对堆h中的元素x减少Δ大小。

(7) delete(x，h)：从配对堆h中删除元素x。

另一个基本的内部操作称为link(h_1，h_2)，将两个配对堆连接为一个新堆，可以证明上面提到的许多操作都基于该连接操作。图10-15说明了一个典型的连接操作。

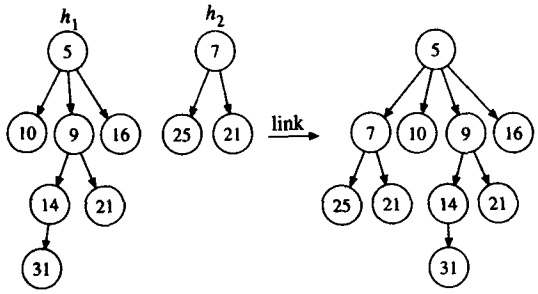


图10-15 操作link(h_1，h_2)的例子

现在举例说明在配对堆中这七个操作是如何实现的。

(1) make heap(h)：仅给新堆分配一个空内存位置。

(2) find min(h)：简单返回堆的根结点。

(3) insert(x，h)：该操作由两步组成，首先生成只有一个结点的树，再将其与配对堆h连接。图10-16和图10-17显示了这两步。

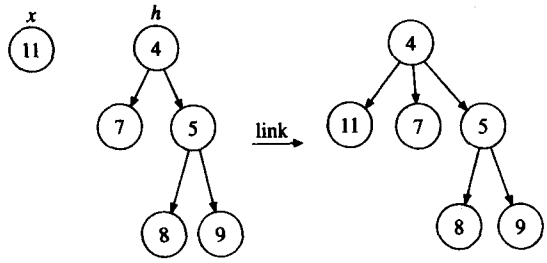


图10-16 插入示例

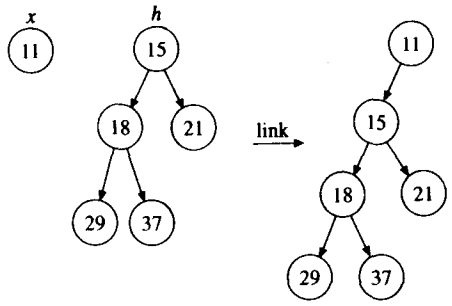


图10-17 另一插入示例

(4) meld(h_1, h_2)：简单地返回通过连接h_1和h_2形成的新堆。

(5) decrease(Δ, x, h)：该操作由三步组成。

步骤1：从元素x中减去Δ。

步骤2：如果x是根，那么返回。

步骤3：剪断x到父结点的连接，这将生成两棵树，连接这两棵树。

考虑操作decrease(3，9，h)，其中h如图10-18所示。图10-19说明该操作是如何实现的。

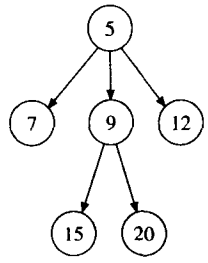


图10-18 说明decrease操作的配对堆

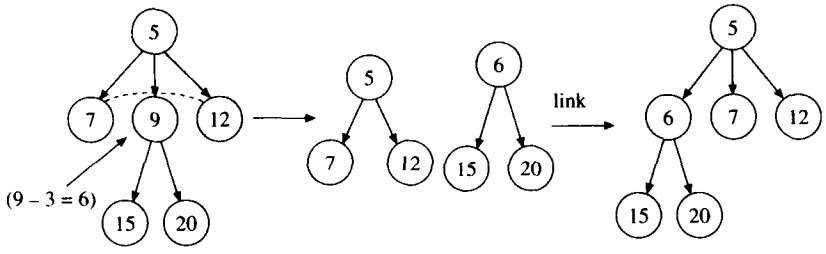


图10-19 对图10-18中的配对堆实施decrease(3，9，h)操作

(6) delete(x, h)：此操作有两步。

步骤1：如果x是根，那么返回（delete min(h)）。

步骤2：否则，

步骤2.1：剪断x到父结点的连接。

步骤2.2：在根为x的树上实现delete min(h)操作。

步骤2.3：连接生成的树。

注意，在delete(x，h)内部有还未描述的delete min(h)操作。因为该操作对于配对堆的分摊分析是非常重要的，所以我们有意推迟对该操作的讨论。实际上，从分摊分析意义上，delete min(h)操作使得配对堆成为一个优美的数据结构。

在描述delete min(h)操作前，考虑图10-20中的配对堆，图10-21是该配对堆的二叉树表示。如图10-21所示，该二叉树严重地向右倾斜。

在删除堆的最小元素，也就是二叉树的根之后，需要重构该堆。该重构必须执行一个函数：找出堆的第二层元素的最小者。相对于二叉树表示，第二层元素都在图10-21中一条路径中。如果堆的第二层有许多元素，那么对应的路径将相当长。在此情形下，找出最小元素的

操作将花费相当长的时间，这是不希望出现的。

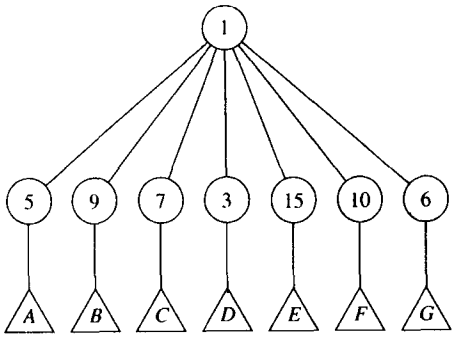


图10-20 说明delete min(h)操作的配对堆

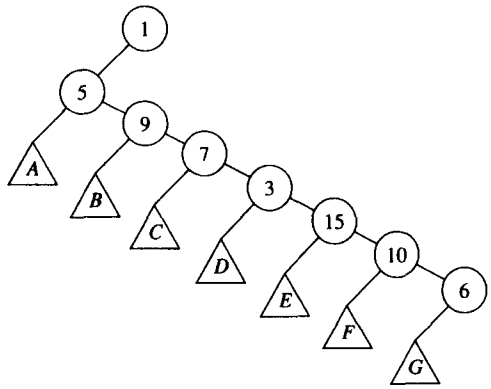


图10-21 图10-20中配对堆的二叉树表示

我们将以很高的概率论证堆中第二层的元素数目是很少的。换句话说，对应的二叉树将是非常平衡的。

再次考虑图10-20中的配对堆。delete min(h)操作首先剪断与堆的根的所有连接，这将生成七个堆，如图10-22所示。

在delete min(h)操作的这一步，成对地合并生成的堆，第一个与第二个，第三个与第四个等。如图10-23所示。

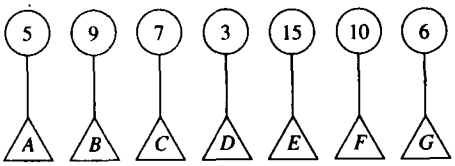


图10-22 delete min(h)操作的第一步

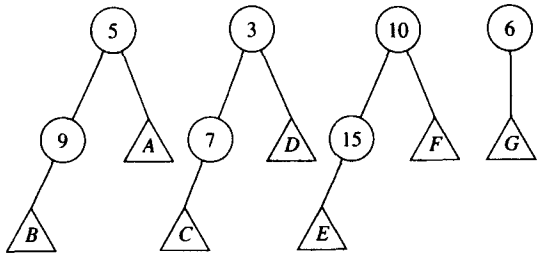
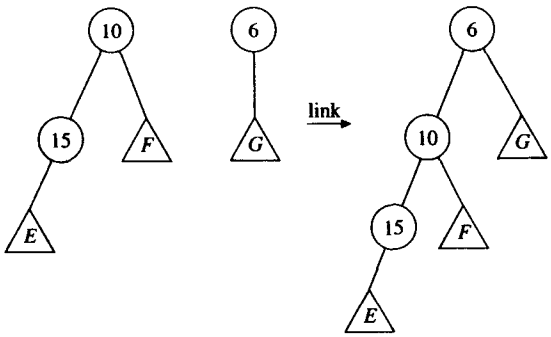


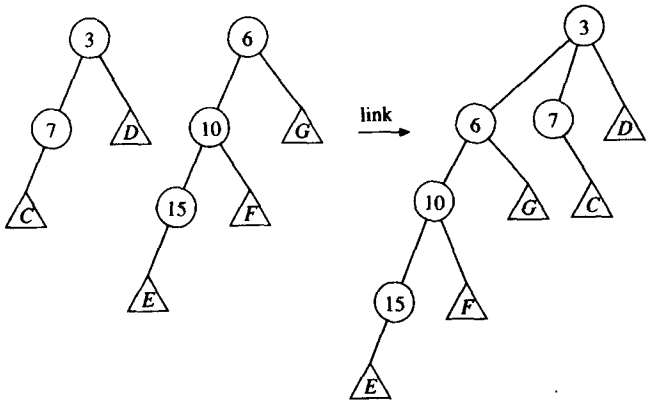
图10-23 delete min(h)操作的第二步

在成对合并操作之后，再一个接一个地连接这些配对堆到最后一个堆上，从倒数第二个堆开始，然后是倒数第三个堆，依此类推，如图10-24所示。

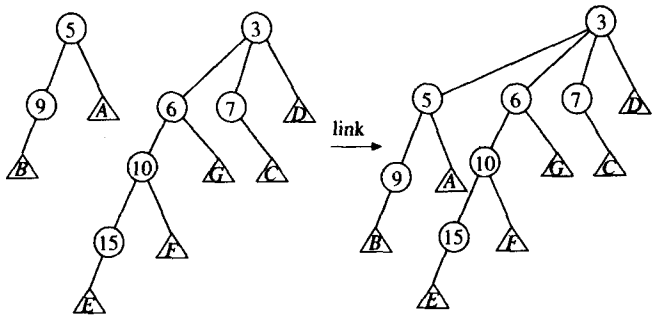


a)

图10-24 delete min(h)操作的第三步



b)



c)

图10-24 （续）

所有delete min(h)操作的三步都可以在堆的二叉树表示上直接实现。

图10-24中生成的堆的二叉树表示如图10-25所示。显然，该二叉树比图10-21所示的更加平衡。

在说明了delete min(h)操作之后，说明delete(x，h)操作。考虑图10-26中的配对堆。如果$x = 6$，那么删除它之后，得到图10-27所显示的堆。

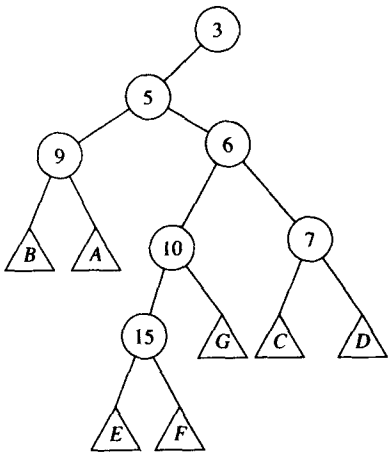


图10-25 图10-24中生成的堆的二叉树表示

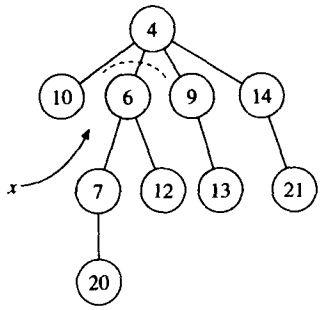


图10-26 说明delete(x，h)操作的配对堆

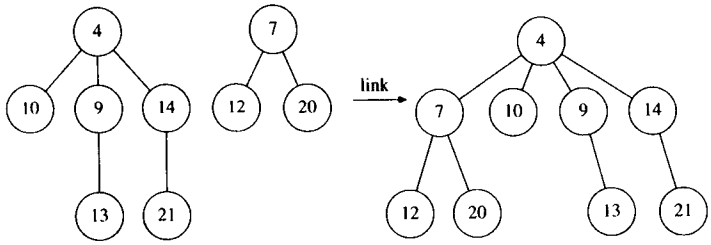


图10-27 从图10-26中的配对堆删除6后的结果

现在将对配对堆的操作给出分摊分析。如前所述，首先定义一个势能函数。

已知二叉树的一个结点x，令$s(x)$表示包含x的子树中的结点数。结点x的秩表示为$r(x)$，定义为$\log(s(x))$。一棵树的势能是所有结点秩的和。

例如，图10-28显示了两棵树的各自势能。

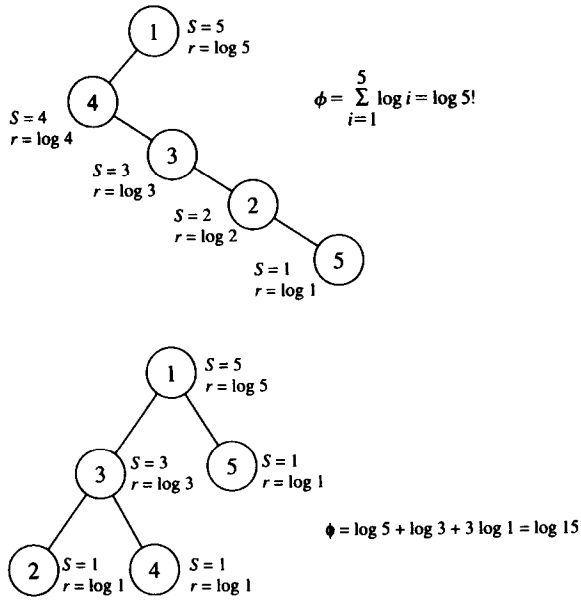


图10-28 两棵二叉树的势能

在七个配对堆的操作中，建堆（make heap）和找最小元素（find minimum）操作对势能不产生任何改变。插入（insert）、合并（meld）和减量（decrease）操作很容易明白势能的改变至多是$\log(n+1)$，这是由内部的连接操作产生的。对于删除（delete）和删除最小元素（delete minimum）操作，关键的操作是删除最小元素操作。使用一个例子来说明势能的改变。

考虑图10-29，在图10-29a中配对堆的二叉树如图10-29b所示。

令r_a、r_b、r_c和r_d分别表示结点A、B、C和D的秩，结点A、B、C和D的结点总数分别为S_a、S_b、S_c和S_d，那么，图10-29b中二叉树的势能是

$$\begin{aligned}\phi = {} & r_a + r_b + r_c + r_d + \log(S_a + S_b + S_c + S_d + 5) \\ & + \log(S_a + S_b + S_c + S_d + 4) + \log(S_b + S_c + S_d + 3) \\ & + \log(S_c + S_d + 2) + \log(S_d + 1)\end{aligned}$$

在删除最小元素1后，配对堆操作序列将如图10-30a~c所示。最后的配对堆表示在图10-30c中，其二叉树表示在图10-30d中。

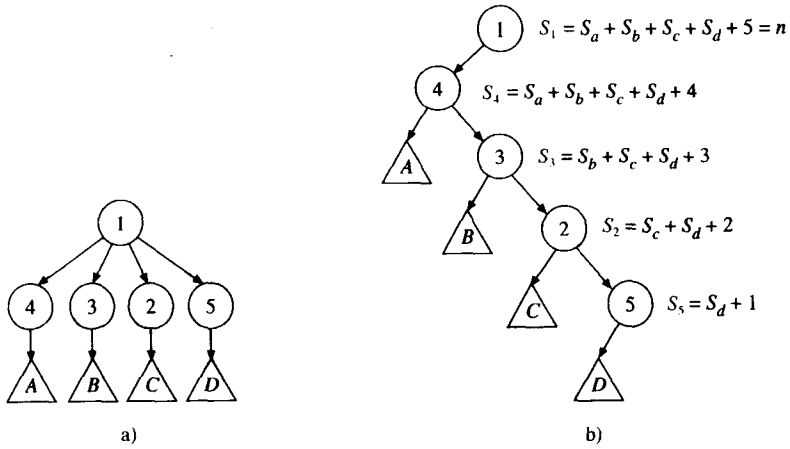


图10-29 一个堆和它的说明势能改变的二叉树

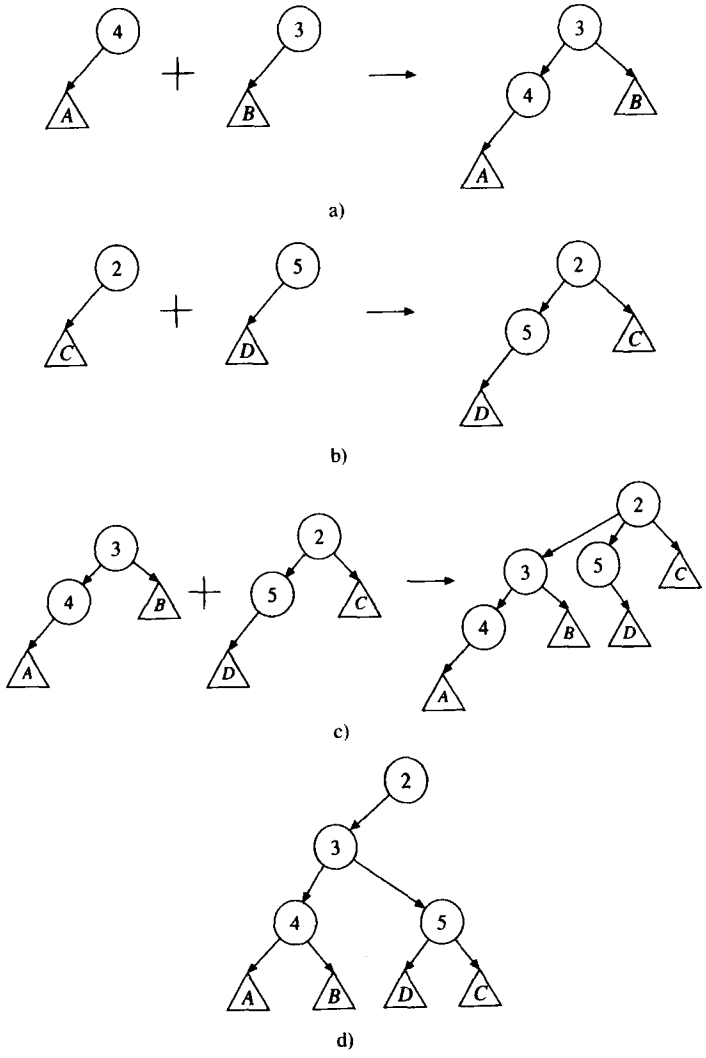


图10-30 删除图10-29中最小元素的配对堆序列及其最终堆的二叉树表示

新的势能变为

$$\begin{aligned}\phi' &= r_a + r_b + r_c + r_d + \log(S_a + S_b + 1) + \log(S_c + S_d + 1) \\ &\quad + \log(S_a + S_b + S_c + S_d + 3) + \log(n-1) \\ &= r_a + r_b + r_c + r_d + \log(S_a + S_b + 1) + \log(S_c + S_d + 1) \\ &\quad + \log(S_a + S_b + S_c + S_d + 3) + \log(S_a + S_b + S_c + S_d + 4)\end{aligned}$$

总之，设有操作序列OP_1，OP_2，…，OP_q，每种操作OP_i是配对堆的七个操作之一。令$a_i = t_i + \phi_i - \phi_{i-1}$，其中$\phi_i$和$\phi_{i-1}$分别表示操作$OP_i$之后和之前的势能，$t_i$是完成操作$OP_i$所需的时间。接下来，将集中注意力在删除最小元素（delete minimum）操作上，推导出该操作的分摊上界。显然，所有其他操作都具有同样的上界。

注意，删除最小元素操作由下面的操作组成：

(1) 删除根。

(2) 配对合并。

(3) 一个接一个地与最后堆合并。

对第一个操作，显然势能的最大变化是$\Delta\phi_i = -\log n$。

接着考虑配对合并。最初，二叉树有一串结点，如图10-31所示。

重画图10-31，如图10-32所示。

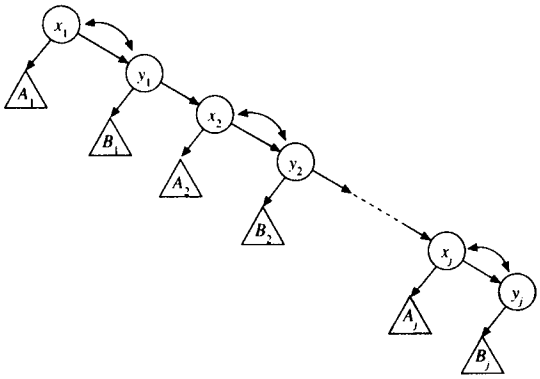


图10-31 配对操作

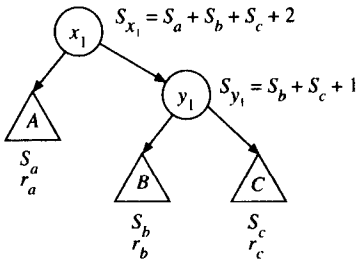


图10-32 图10-31的重画

令r_a、r_b和r_c分别表示结点A、B和C的秩，结点A、B和C的结点总数分别表示为S_a、S_b和S_c，那么第一次合并前的势能为

$$\phi_{before} = r_a + r_b + r_c + \log(S_a + S_b + S_c + 2) + \log(S_b + S_c + 1)$$

第一次合并操作合并两个堆，如图10-33所示。依赖于x_1和y_1之间的关系有两种合并的可能。这两种生成堆的二叉树表示如图10-34所示，对于两种可能的二叉树表示，再加入其他部分如图10-35所示。

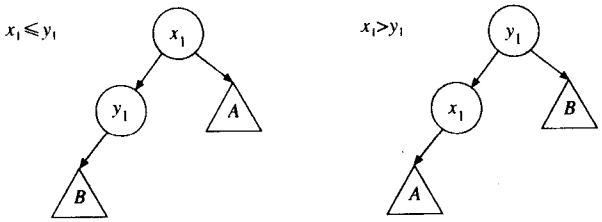


图10-33 第一次合并后两种可能的堆

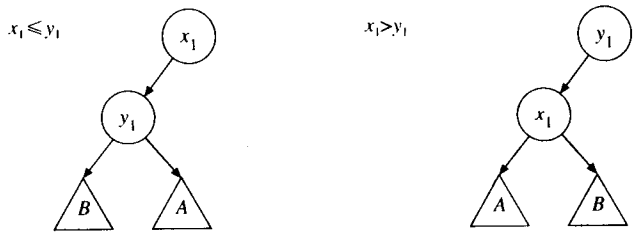


图10-34 图10-33中两个堆的二叉树表示

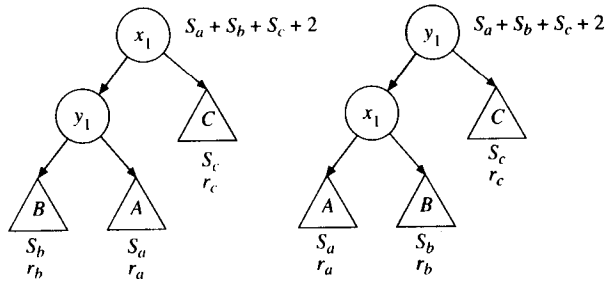


图10-35 第一次合并后生成的二叉树

对于图10-35中的两棵二叉树，其势能是

$$\phi_{after} = r_a + r_b + r_c + \log(S_a + S_b + S_c + 2) + \log(S_a + S_b + 1)$$

因此，势能的改变为

$$\begin{aligned}\Delta\phi_{pairing} &= \phi_{after} - \phi_{before} \\ &= \log(S_a + S_b + 1) - \log(S_b + S_c + 1)\end{aligned}$$

尽管此结果参考了第一次合并，但是，显然$\Delta\phi_{pairing}$也指一般的配对。

我们要证明$\Delta\phi_{pairing} \leqslant 2\log(S_x) \leqslant 2\log(S_c) - 2$，其中$S_x$表示在改变前根为$x$的子树总的结点数。为了证明上述结论，将利用下面的事实：

如果p，$q>0$，$p+q \leqslant 1$，那么$\log p + \log q \leqslant -2$。

上面的性质很容易证明。现在，令

$$p = \frac{S_a + S_b + 1}{S_a + S_b + S_c + 2}$$

和 $q = \dfrac{S_c}{S_a + S_b + S_c + 2}$

那么，$\log\left(\dfrac{S_a + S_b + 1}{S_a + S_b + S_c + 2}\right) + \log\left(\dfrac{S_c}{S_a + S_b + S_c + 2}\right) \leqslant -2$，也就是

$$\log(S_a + S_b + 1) - \log(S_c) \leqslant 2\log(S_a + S_b + S_c + 2) - 2\log(S_c) - 2$$

但是，$\log(S_c) < \log(S_b + S_c + 1)$。

所以，

$$\log(S_a + S_b + 1) - \log(S_b + S_c + 1) \leqslant 2\log(S_a + S_b + S_c + 2) - 2\log(S_c) - 2$$

因此，可以得到

$$\Delta\phi_{pairing} \leqslant 2\log(S_a + S_b + S_c + 2) - 2\log(S_c) - 2$$
$$= 2\log(S_x) - 2\log(S_c) - 2$$

对于最后一对，$S_c = 0$。在此情况下，

$$\Delta\phi_{pairing} \leqslant 2\log(S_x)$$

首先注意到，确实满足了有j对结点的假定。如果有$2j+1$个结点，那么最后一个结点不与其他结点合并，所以势能没有改变。配对合并总的变化是

$$\Delta\phi_{total\ pairing} = \sum_{i=1}^{j-1}(\text{第}i\Delta\phi_{pairing}) + (\text{第}j\text{对的}\Delta\phi_{pairing})$$
$$\leqslant \sum_{i=1}^{j-1}(2\log(S_{x_i}) - 2\log(S_{x_{i+1}}) - 2) + 2\log(S_{x_j})$$
$$= 2(\log(S_{x_1}) - \log(S_{x_2}) + \log(S_{x_2}) - \log(S_{x_3}) + \cdots$$
$$-\log(S_{x_j})) + 2\log(S_{x_j}) - 2(j-1)$$
$$= 2\log(S_{x_1}) - 2(j-1)$$
$$\leqslant 2\log n - 2j + 2$$

因此，删除最小元素操作的第二步，

$$\Delta\phi_{total\ pairing} \leqslant 2\log n - 2j + 2$$

最后，讨论最后一步。这一步是在上面配对操作之后发生的，它一个接一个与最后的堆合并。在图10-36中说明由配对合并形成的二叉树（对于所有的i，假定$x_i < y_i$）。

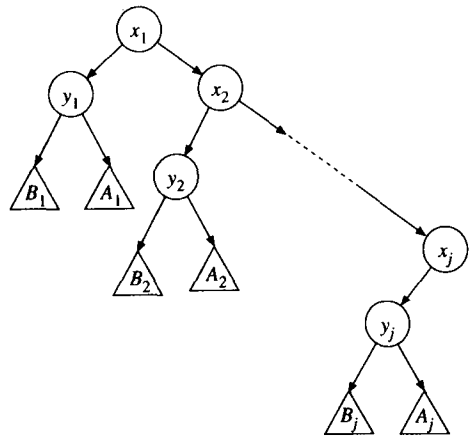


图10-36 配对操作之后的二叉树

最后一对配对合并如图10-37所示，势能为

$$\phi_{before} = r_{A_{j-1}} + r_{B_{j-1}} + r_{A_j} + r_{B_j} + \log(S_{A_{j-1}} + S_{B_{j-1}} + 1)$$
$$+ \log(S_{A_j} + S_{B_j} + 1) + \log(S_{A_j} + S_{B_j} + 2)$$
$$+ \log(S_{A_{j-1}} + S_{B_{j-1}} + S_{A_j} + S_{B_j} + 4)$$

合并之后，依赖于$x_{j-1} < x_j$或$x_{j-1} \geqslant x_j$，新生成的二叉树或者像图10-38a或者像图10-38b。

由于新势能与图10-38a和图10-38b中显示的二叉树相同，所以只考虑其中的一种。对于图10-38a中的二叉树，新的势能为

$$\phi_{after} = r_{A_{j-1}} + r_{B_{j-1}} + r_{A_j} + r_{B_j} + \log(S_{A_{j-1}} + S_{B_{j-1}} + 1) + \log(S_{A_j} + S_{B_j} + 1)$$
$$+ \log(S_{A_{j-1}} + S_{B_{j-1}} + S_{A_j} + S_{B_j} + 3)$$
$$+ \log(S_{A_{j-1}} + S_{B_{j-1}} + S_{A_j} + S_{B_j} + 4)$$
$$\Delta\phi = \phi_{after} - \phi_{before}$$
$$= \log(S_{A_{j-1}} + S_{B_{j-1}} + S_{A_j} + S_{B_j} + 3) + \log(S_{A_j} + S_{B_j} + 2)$$

$S_{A_{j-1}} + S_{B_{j-1}} + S_{A_j} + S_{B_j} + 4$是整棵树的总结点数，$S_{A_j} + S_{B_j} + 2$是最后二叉树的总结点数。令由$x_i$、$y_i$、$A_i$和$B_i$组成的树的结点数为$n_i$，那么子树中的结点数是$n_1$，$n_2$，…，$n_j$。总势能的改变是

$$\Delta\phi_{\text{third step}} = \log(n_1 + n_2 + \cdots + n_{j-1}) - \log(n_2 + n_3 + \cdots + n_j)$$

$$
\begin{aligned}
&+\log(n_2+n_3+\cdots n_j-1)-\log(n_3+n_4+\cdots+n_j)\\
&+\cdots+\log(n_{j-1}+n_j-1)-\log(n_j)\\
&<\log(n-2)-\log(n_j)\\
&<\log(n-1)
\end{aligned}
$$

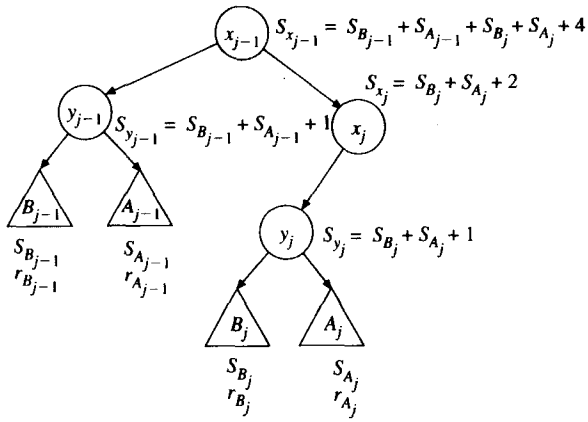


图10-37 删除最小元素操作的第三步中的一对子树

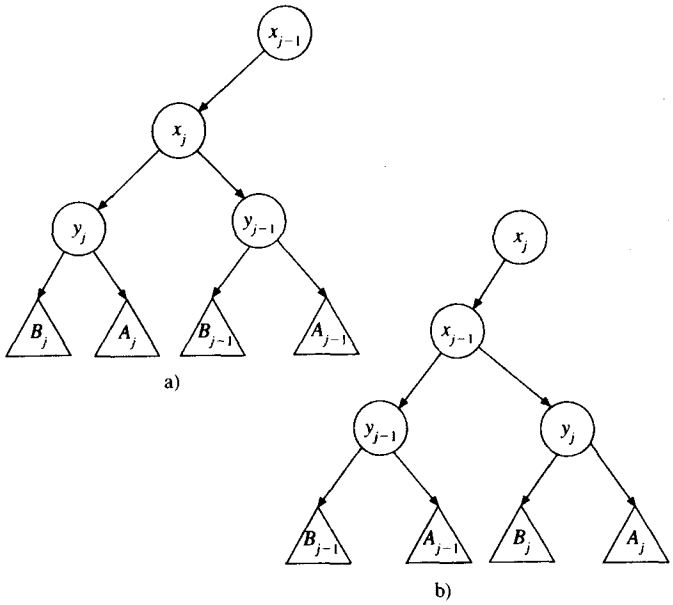


图10-38 在删除最小元素操作的第三步中一对子树的合并结果

现在可以明白整个删除最小元素的操作，

$$
\begin{aligned}
a_i &= t_i+\phi_i-\phi_{i-1}\\
&\leqslant 2j+1-\log n+(2\log n-2j+2)+\log(n-1)\\
&\leqslant 2\log n+3\\
&= O(\log n)
\end{aligned}
$$

尽管对删除最小元素的操作，$O(\log n)$是一个上界，很容易明白这也是所有其他操作的上

界。总之，可以断定配堆操作的分摊时间是$O(\log n)$。

10.6 不相交集合并算法的分摊分析

在本节中，将讨论一种很容易实现的算法。对它的分摊分析表明它有一个明显的几乎线性的运行时间。在合并操作下，该算法解决了维护不相交集合的合并问题。更精确地说，该问题在不相交集合中执行三种操作：生成新集合（makeset）；确定包含已知元素的集合（find）；合并两个集合成一个集合（link）。作为识别集合的方法，假定算法在每个集合中维护一个任意但唯一的代表元素，称为该集合的典型元素（canonical element）。现在阐明三个集合操作如下：

(1) makeset(x)：生成包含单一元素x且先前不存在的新集合。

(2) find(x)：返回包含元素x的集合的典型元素。

(3) link(x, y)：形成一个由典型元素分别是x和y的两集合并而形成的新集合。销毁原来的两个集合，为新集合选择并返回一个典型元素。此操作假定$x \neq y$。

注意没有删除操作。

为解决此问题，用根树（rooted tree）表示每个集合。树的结点是集合中的元素，而典型元素作为树的根。每个结点x有指向其父结点$p(x)$的指针，根结点的指针指向自身。为了执行操作makeset(x)，定义$p(x)$就是x。为了执行find(x)，沿着从x到含有x的树根的指针，并返回根。为了执行link(x, y)，定义$p(x)$是y，将y作为新集合的典型元素返回。参见图10-39，操作find(x_6)将返回x_1，操作link(x_1, x_8)将使x_1指向x_8。

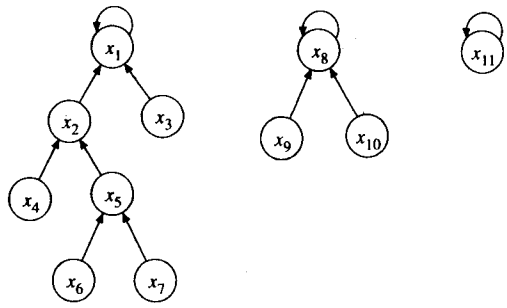


图10-39 集合$\{x_1, x_2, \cdots, x_7\}$、$\{x_8, x_9, x_{10}\}$和$\{x_{11}\}$的表示

该朴素算法并不非常有效，每个find操作在最坏情况下要求$O(n)$时间，其中n是元素的总数（makeset操作）。通过对该算法增添两个启发式方法，可极大地提高其性能。第一个方法称为路径压缩（path compression），在find操作期间，通过向树根移近结点来改变树结构：当执行find(x)时，在定位包含x的树根r后，将从x到r路径上的每个结点直接指向r（见图10-40）。路径压缩增加了单个find操作一个常数因子的时间，但为后续的find操作节省了足够的时间。

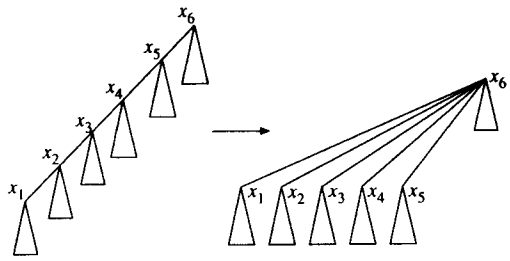


图10-40 路径$[x_1, x_2, x_3, x_4, x_5, x_6]$的压缩

第二种称为按秩合并（union by rank）的启发式方法保持树得低。结点x的秩表示为$rank(x)$，定义如下：

（1）当makeset(x)操作执行时，$rank(x)$赋值为0。

（2）当link操作执行时，令x和y是两个根结点，有两种情况：

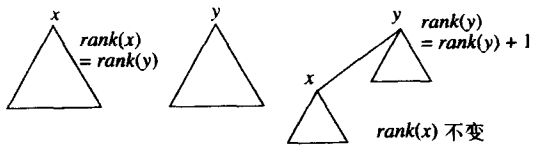
（a）情况1：$rank(x) = rank(y)$。

在此情况下，使x指向y，$rank(y)$增加1，返回y为典型元素。（见图10-41a），不改变x的秩。

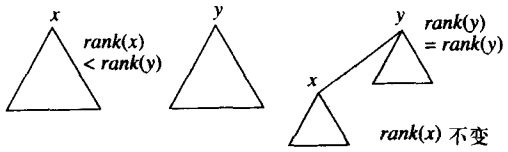
（b）情况2：$rank(x) < rank(y)$。

在此情况下，使x指向y，返回y为典型元素。x和y的秩保持一样。(见图10-41b)。

(3) 当路径压缩启发式方法执行时，没有秩的改变。



a) 相同秩的根



b) 不同秩的根

图10-41 按秩合并

现在证明一些如上定义秩的有趣性质，这些性质在分析算法性能时是很有用的。

性质1：*如果$x \neq p(x)$，那么$rank(x)<rank(p(x))$。*

根据秩启发式方法，由于合并该性质成立。当连接两棵树时，新树的根具有最高的秩。

性质2：*$rank(x)$的值初始是0，随时间而增长，直到它不再是根为止；随后$rank(x)$值不再改变。$rank(p(x))$的值是个非递减的时间函数。*

注意，由于x要初始化，它的秩从0开始。只要x保持是根，它的秩就永不减少。一旦它有父结点，它的秩就不再改变了，因为路径压缩启发式方法不能抵消秩。

考虑图10-42。在一个find操作期间，假如这是算法转换的一条路径。由于性质1，

$rank(x_1)<rank(x_2)<\cdots<rank(x_r)$

在路径压缩操作之后，这部分树如图10-43所示。现在，x_1有一个新父结点x_r，显然，此新的父结点x_r比x_1先前的父结点有更高的秩。随后，x_r可能连接到某个具有更高秩的结点。所以，x_1后来有另一个新的更高秩的父结点是可能的。这就是$rank(p(x))$的值不会减少的原因。

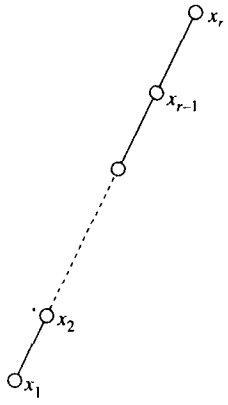


图10-42 find操作路径

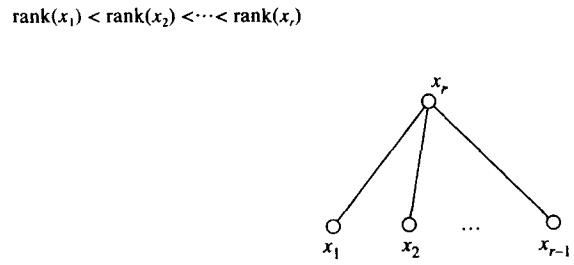


图10-43 在路径压缩操作后图10-42中find操作的路径

性质3：*根为x的树的结点数至少是$2^{rank(x)}$。*

显然树在初始化时性质3显然是成立的。现在假设在两棵树连接前性质3成立。令两棵树

的根是x和y，如果$rank(x)<rank(y)$，那么新树将y作为根。y的秩不再改变，新树比之前有更多的结点。因此，对该新树性质3成立。如果$rank(x) = rank(y)$，那么新树至少有$2^{rank(x)} + 2^{rank(y)} = 2 \times 2^{rank(y)} = 2^{rank(y)+1}$个结点。但是，新根的秩是$rank(y) + 1$。所以，性质3对此情形也成立。

性质4：*对任意$k \geqslant 0$，秩为k的结点数至多是$n/2^k$，其中n是元素的个数。*

可将秩为k的结点作为树的根。根据性质3，该树至少有2^k个结点。因此，有不超过$n/2^k$结点具有秩k；否则，可能超过n个结点，这是不可能的。

性质5：*最高秩是$\log_2 n$。*

当只有一棵树时，才可得到最高秩。在此情形下，如性质3表明$2^k \leqslant n$。因此，$k \leqslant \log_2 n$。

我们的目标是分析三种集合操作混合序列的运行时间。很容易看到操作makeset(x)和link(x，y)可在常数时间内完成。所以，下面只分析find操作所需的时间。使用m表示find操作的次数，n表示元素数。这样，makeset操作次数是n，link操作的次数至多是$n-1$，并且$m \geqslant n$。因为路径压缩以一种复杂的方式改变了树的结构，使得分析变得困难。然而，使用分摊分析，可以得到实际最坏情况的界为$O(m\alpha(m, n))$，其中$\alpha(m, n)$是阿克曼函数（Ackermann's function）的逆函数。对i，$j \geqslant 1$，阿克曼函数$A(i, j)$定义如下：

$A(1, j) = 2^j$　　对于$j \geqslant 1$

$A(i, 1) = A(i-1, 2)$　　对于$i \geqslant 2$

$A(i, j) = A(i-1, A(i, j-1))$　　对于$i, j \geqslant 2$

对m，$n \geqslant 1$，阿克曼函数的逆$\alpha(m, n)$定义为

$\alpha(m, n) = \min\{i \geqslant 1 \mid A(i, \lfloor m/n \rfloor) > \log_2 n\}$

$A(i, j)$最重要的性质是它爆炸性地增长。可以验证其中一些：

$A(1, 1) = 2^1 = 2$

$A(1, 2) = 2^2 = 4$

$A(1, 4) = 2^4 = 16$

$A(2, 1) = A(1, 2) = 4$

$A(2, 2) = A(1, A(2, 1)) = A(1, 4) = 16$

$A(2, 3) = A(1, A(2, 2)) = A(1, 16) = 2^{16}$

$A(3, 1) = A(2, 2) = 16$。

这样，$\alpha(m, n) \leqslant 3$是对m/n并且$n<2^{16} = 65\ 536$合理的值。事实上，对所有现实的目标，$\alpha(m, n)$是不超过4的常数。最后，要推导m次find操作总的运行时间$O(m\alpha(m, n))$的上界。

在正式分析算法之前，先简要地描述分析的基本思想。分析由下面的步骤组成：

(1) 对于树的每个结点，分配一个层次i，该层次与秩有关。将证明有1，2，…，$\alpha(m, n)+1$层。如果一个结点有较高的层，那么表明该结点有大量的兄弟，或者树的这部分相当扁平。对i层，整数分成许多块(i, j)，后面将加以解释。

(2) 通过一些机制，将结点分成两类：贷方结点（credit node）和借方结点（debit node）。对于任意路径，贷方结点至多是一个常数。因此，如果路径很长，那么总是由于有大量的借方结点。反之，如果路径很短，几乎都是贷方结点。

(3) find操作总的时间花费是被find操作转换的贷方结点数与借方结点数的和。由于被find操作转换的贷方结点数有常数界，所以，找出被find操作转换的借方结点数的上界是非常重要的。

(4) 对属于块(i, j)的借方结点，如果它被find操作算法转换过$b_{ij} - 1$次，其中b_{ij}将在后面解释，结点的层将增加到$i + 1$。

(5) 令n_{ij}表示在块(i, j)中的借方结点数。值$n_{ij}(b_{ij}-1)$是被find操作转换借方结点所花费的时间单位，这将提高在块(i, j)中的所有结点层次到$i+1$。

(6) 由于最高层是$\alpha(m, n)+1$，$\sum_{i=1}^{\alpha(m,n)+1}\sum_{j\geq 0} n_{ij}(b_{ij}-1)$是被这$m$次find操作转换的借方结点总数的上界。

从定义层开始，再先回顾一下阿克曼函数。本质上阿克曼函数将整数分为不同层次的块，再重写阿克曼函数：

$A(1, j) = 2^j$ 对于$j \geq 1$

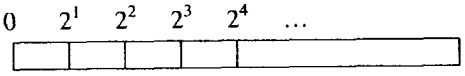
$A(i, 1) = A(i-1, 2)$ 对于$i \geq 2$

$A(i, j) = A(i-1, A(i, j-1))$ 对于$i, j \geq 2$

我们将使用上面的函数来划分整数。对层i，整数划分到$A(i, j)$定义的块中。对于$j \geq 1$，块$(i, 0)$含从0到$A(i, 1)-1$的整数，块(i, j)含从$A(i, j)$到$A(i, j+1)-1$的整数。例如，对层1，

$A(1, j) = 2^j$

这样，这些块可说明如下：



对层2，

$A(2, 1) = 2^2$

$A(2, 2) = 2^4 = 16$

$A(2, 3) = 2^{16} = 65\,536$

对层3，

$A(3, 1) = 2^4 = 16$

$A(3, 2) = A(2, A(3, 1))$

$= A(2, 16)$

$= A(1, A(2, 15))$

$= 2^{A(2,15)}$

由于$A(2, 3)$已经是一个非常大的数，$A(2, 15)$对于实际目标是个非常大的数，$2^{A(2,15)}$可看作是无限。

组合这三层得到如图10-44所示的图表。

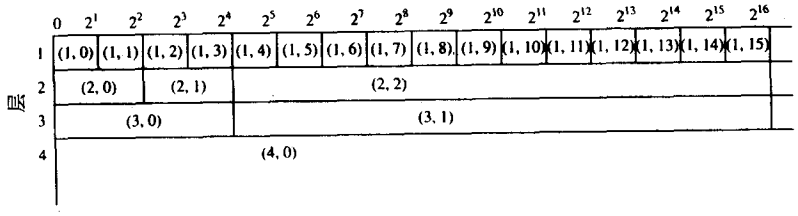


图10-44 对应于阿克曼函数的不同层

假定有一个整数$2^3 \leq i \leq 2^4$，那么在层1，它在块(1，3)中；在层2，它在块(2，1)中；在层3，它在块(3，0)中。

如之前所定义的，$p(x)$表示结点x的父结点。现在x的层次定义成使$rank(x)$和$rank(p(x))$在层i划分的同一个块内的最小层i。如果$x = p(x)$，那么x的层是0。假定$rank(x)$在块(1，1)中，$rank(p(x))$在块(1，3)中，那么x在层3。另一方面，如果$rank(x)$在块(1，3)中，$rank(p(x))$在块(1，6)中，那么x在层4。

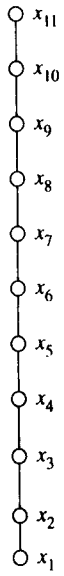
层有多高呢？当然，如果层很高使得该层的第一块包含$\log_2 n$，那么该层因为一个简单原因足够高：根据性质5，$\log_2 n$是最大可能的秩。现在验证一个例子，假如$n = 2^{16}$，这已经足够大。在此情形下，$\log_2 n = 16$。如图10-44所示，这包含在层4的第一块内。回忆

$$\alpha(m, n) = \min\{i \geqslant 1 \mid A(i, \lfloor m/n \rfloor) > \log_2 n\}$$

可以很容易明白$A(\alpha(m, n), m/n) > \log_2 n$。换句话说，$\alpha(m, n)+1$是结点能被关联的最高层。

现在考虑涉及路径x_1，…，x_r的一次find操作。一个典型的例子如图10-45所示。在图10-45中，每个结点关联一个秩，利用$rank(x_i)$和$rank(x_{i+1})$，可确定每个x_i的层。

对于每个i，$0 \leqslant i \leqslant \alpha(m, n) + 1$，分配在该路径上层$i$的最后一个结点为贷方结点，其他结点为借方结点。所有的贷方结点和借方结点显示在图10-45中。根据贷方结点的定义，任何find路径的贷方结点的总数是以$\alpha(m, n) + 2$为界。因此，被算法转换的贷方结点总数是以$m(\alpha(m, n) + 2)$为界。



秩	层	贷方/借方
150	0	贷方
18	2	贷方
17	1	贷方
13	4	贷方
12	1	借方
10	1	借方
7	2	借方
5	1	借方
3	3	贷方
2	1	借方
1	2	借方

图10-45 贷方结点和借方结点

现在讨论结点的层如何那么高。根据定义，如果结点x的$rank(x)$和$rank(p(x))$的差非常大，那么x的层就高。所以，可能有下面的问题：x的父结点的秩比x的秩怎么高出那么多？（只有两种操作：合并和压缩。当执行一次合并操作，x的秩与$p(x)$的秩间的差别仅是1——译者注）

当执行压缩操作时，结点x的秩与其父结点的秩间的差别将非常显著。设想在图10-45的路径上执行find操作，在find操作之后，路径将被压缩，如图10-46所示。尽管x_1的秩仍然是1，但其父结点的秩突变为150。这样，现在x_1的层从1增加到4。

总之，find操作路径中的结点在find操作之后，会有一个新的父结点，并且新的父结点必

定有一个较高的秩。我们将证明，如果x在层i是借方结点，并且$rank(x)$在块$(i-1, j)$中，那么新的父结点实际上是树的根，将在块$(i-1, j')$中，其中$j' > j$。

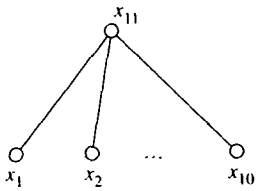


图10-46 find操作之后图10-45的路径

假定结点x的层是i，$rank(x)$在块$(i-1, j)$中。令x的父结点$p(x)$的秩在块$(i-1, j')$中，那么，因为$rank(p(x)) \geqslant rank(x)$，当然$j' \geqslant j$。进而，$j' \neq j$；否则，$x$将在层$i-1$。因此推断，如果结点$x$的层是$i$，$rank(x)$在块$(i-1, j)$中，那么$p(x)$的秩在块$(i-1, j')$中，其中$j' > j$。

考虑图10-47，其中x_{k+1}是x_k的父结点，x_k是在层i的一个借方结点。由于x_k是一个借方结点，根据定义，一定有贷方结点，比如说x_a，必定在x_k与该路径的根x_r之间。令x_{a+1}是x_a的父结点，现在，令x_k、x_{k+1}、x_a、x_{a+1}和x_r的秩分别在块$(i-1, j_1)$、$(i-1, j_2)$、$(i-1, j_3)$、$(i-1, j_4)$和$(i-1, j_5)$中，如下所示。

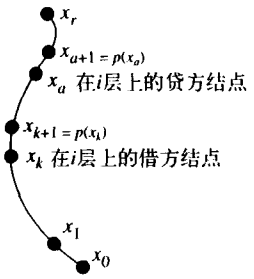


图10-47 在路径中结点的秩

	$rank$
x_k	$(i-1, j_1)$
x_{k+1}	$(i-1, j_2)$
x_a	$(i-1, j_3)$
x_{a+1}	$(i-1, j_4)$
x_r	$(i-1, j_5)$

现在，由于x_k的层是i，可得$j_2 > j_1$。

由于x_a在x_{k+1}与x_r之间，可得$j_3 \geqslant j_2$。

由于x_a的层也是i，可得$j_4 > j_3$。

最后，可得$j_5 \geqslant j_4$。

简而言之，可以得到$j_1 < j_3 < j_5$。

上面的讨论解答了一般的情形。$x_a = x_{k+1}$和$x_r = x_{a+1}$也是可能的。在此情形下，相关的讨论只涉及三个结点，即x_k、x_{k+1}和x_r。很容易证明对一般及特殊情形，下面的陈述是正确的：令x_k是在层i的借方结点，x_{k+1}是其父结点，x_r是根结点。令$rank(x_k)$、$rank(x_{k+1})$和$rank(x_r)$分别在块$(i-1, j)$、$(i-1, j')$和$(i-1, j'')$中，那么$j < j' < j''$，如图10-48所示。

$rank(x_k)$		$rank(x_{k+1})$		$rank(x_r)$	
	$i-1, j$	…	$i-1, j'$	…	$i-1, j''$

图10-48 在层$i-1$中x_k、x_{k+1}和x_r的秩

在一次find操作之后，由于路径压缩方法，路径中除了根结点外的每个结点都变成叶结点。根据定义，叶结点是贷方结点。这样，在find操作之后，借方结点将变为贷方结点。只要没有合并操作，它将保持是贷方结点，而在link操作后，它将再次变为借方结点。

每次find操作使一个借方结点有新的父结点，新的父结点的秩将高于先前父结点的秩。此外，如上面所证明的，新的父结点的秩将在较高层$(i-1)$的块中。令b_{ij}表示层$(i-1)$的块数，它们与块(i, j)的交集非空。例如，$b_{22} = 12$，$b_{30} = 2$，那么可说当x是一个借方结点时，在

$b_{ij}-1$次find操作转换x一次后，$rank(x)$和$rank(p(x))$将在不同的层i块中。或者，更详细地，在$b_{ij}-1$次借方结点改变后，x的层增加到$i+1$。也可以说成最大$b_{ij}-1$次借方结点改变后，x的层数增1。

令n_{ij}表示秩在块(i, j)中的借方结点的总数。find操作能够转换借方结点的总次数至多是

$$Q=\sum_{i=1}^{\alpha(m,n)+1}\sum_{j\geq 0}n_{ij}(b_{ij}-1)$$

在下面的段落中，将证明如何推导Q的上界，此推导是相对复杂的。实际上，如果读者只对分摊分析的基本规律感兴趣，那么可以简单地忽略此部分的讨论。

再次考虑图10-44所示的划分图表，可以划分该图表为三部分，如图10-49所示。

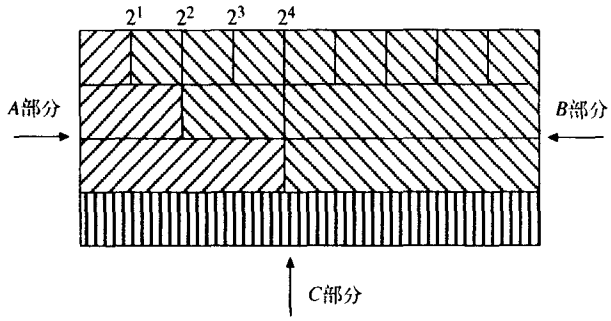


图10-49 对图10-44中图表的划分

这样，可以得到

$$\begin{aligned}Q&=\sum_{i=1}^{\alpha(m,n)+1}\sum_{j\geq 0}n_{ij}(b_{ij}-1)\\&=\sum_{i=1}^{\alpha(m,n)}n_{i0}(b_{i0}-1) && (A\text{部分})\\&+\sum_{i=1}^{\alpha(m,n)}\sum_{j\geq 1}n_{ij}(b_{ij}-1) && (B\text{部分})\\&+n_{\alpha(m,n)+1,0}(b_{\alpha(m,n)+1,0}-1) && (C\text{部分})\end{aligned}$$

先计算A部分。根据定义，

$$\begin{aligned}block(i,0)&=[0,\cdots,A(i,1)-1]\\&=[0,\cdots,A(i-1,2)-1]\\&=[0,\cdots,A(i-1,1),\cdots,A(i-1,2)-1]\end{aligned}$$

这意味着$block(i, 0)$覆盖两个$(i-1)$层的块，即块$(i-1, 0)$和$(i-1, 1)$。所以，可以得到

$b_{i0}=2$

另一方面，$n_{i0}\leq n$。这样，对A部分

$$\sum_{i=1}^{\alpha(m,n)}n_{i0}(b_{i0}-1)\leq\sum_{i=1}^{\alpha(m,n)}n(2-1)=n\alpha(m,n)$$

下面计算B部分，就是

$$\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1}n_{ij}(b_{ij}-1)$$

首先推导n_{ij}的上界。根据上面性质4的讨论，

$$\begin{aligned} n_{ij} &\leqslant \sum_{k=A(i,j)}^{A(i,j+1)-1} n/2^k \\ &\leqslant \sum_{k\geqslant A(i,j)} n/2^k \\ &\leqslant 2n/2^{A(i,j)} \\ &= n/2^{A(i,j)-1} \end{aligned} \tag{10-1}$$

对于b_{ij}，$1\leqslant i\leqslant\alpha(m,n)$及$j\geqslant 1$，一般也可以得到一个上界。

$$\begin{aligned} block(i,j) &= [A(i,j),\cdots,A(i,j+1)-1] \\ &= [A(i-1,A(i,j-1)),\cdots,A(i-1,A(i,j))-1] \\ &= [A(i-1,A(i,j-1)),A(i-1,A(i,j-1)+1),\cdots,A(i-1,A(i,j))-1] \end{aligned}$$

这意味着对于$1\leqslant i\leqslant\alpha(m,n)$和$j\geqslant 1$，$block(i,j)$覆盖$A(i,j)-A(i,j-1)$ $(i-1)$层的块。在此情形下，

$$b_{ij}\leqslant A(i,j) \tag{10-2}$$

将公式(10-1)和式(10-2)代入B部分，可以得到

$$\begin{aligned} &\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1}n_{ij}(b_{ij}-1) \\ \leqslant\ &\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1}n_{ij}b_{ij} \\ \leqslant\ &\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1}(n/2^{A(i,j)-1})A(i,j) \\ =\ &n\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1}A(i,j)/2^{A(i,j)-1} \end{aligned}$$

令$t=A(i,j)$，可以得到

$$\begin{aligned} &\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1}n_{ij}(b_{ij}-1) \\ \leqslant\ &n\sum_{i=1}^{\alpha(m,n)}\sum_{\substack{t=A(i,j)\\ j\geqslant 1}}t/2^{t-1} \\ =\ &n\sum_{i=1}^{\alpha(m,n)}((A(i,1)/2^{A(i,1)-1})+(A(i,2)/2^{A(i,2)-1})+\cdots) \\ \leqslant\ &n\sum_{i=1}^{\alpha(m,n)}((A(i,1)/2^{A(i,1)-1})+((A(i,1)+1)/2^{A(i,1)}) \\ &+((A(i,1)+2)/2^{A(i,1)+1})+\cdots) \end{aligned}$$

令$a=A(i,j)$。这样，必须找出下列值：

$$S_1=\sum_{t=a}t/2^{t-1}=(a)/2^{a-1}+(a+1)/2^a+(a+2)/2^{a+1}+\cdots \tag{10-3}$$

$$2S_1 = a/2^{a-2} + (a+1)/2^{a-1} + (a+2)/2^{a} + \cdots \tag{10-4}$$

将式（10-4）减式（10-3），可以得到

$$\begin{aligned} S_1 &= a/2^{a-2} + (1/2^{a-1} + 1/2^{a} + 1/2^{a+1} + \cdots) \\ &= a/2^{a-2} + 1/2^{a-2} = (a+1)/2^{a-2} \end{aligned}$$

这样，

$$\begin{aligned} & n\sum_{i=1}^{\alpha(m,n)}((A(i,1)/2^{A(i,1)-1}) + ((A(i,1)+1)/2^{A(i,1)}) \\ & \quad + ((A(i,1)+2)/2^{A(i,1)+1}) + \cdots) \\ &= n\sum_{i=1}^{\alpha(m,n)}(A(i,1)+1)/2^{A(i,1)-2} \\ &\leqslant n((A(1,1)+1)/2^{A(1,1)-2} + (A(2,1)+1)/2^{A(2,1)-2} \\ & \quad + (A(3,1)+1)/2^{A(3,1)-2} + \cdots) \\ &\leqslant n((2+1)/2^{2-2} + (3+1)/2^{3-2} + (4+1)/2^{4-2} + \cdots) \\ &= n\sum_{t\geqslant 2}(t+1)/2^{t-2} \end{aligned}$$

容易证明

$$\begin{aligned} \sum_{t\geqslant 2}(t+1)/2^{t-2} &= 3/2^{-1} + (1/2^{0} + 1/2^{1} + 1/2^{2} + \cdots) \\ &= 6+2 \\ &= 8 \end{aligned}$$

这样，对于B部分，可以得到

$$\sum_{i=1}^{\alpha(m,n)}\sum_{j\geqslant 1} n_{ij}(b_{ij}-1) \leqslant 8n$$

最后，对于C部分，

$$n_{\alpha(m,\ n)+1,0}(b_{\alpha(m,\ n)+1,0}-1) \leqslant n_{\alpha(m,\ n)+1,0} b_{\alpha(m,\ n)+1,0} \tag{10-5}$$

$$n_{\alpha(m,\ n)}+1,0 \leqslant n \tag{10-6}$$

$block(\alpha(m,\ n)+1,\ 0)$

$= [0,\ \cdots,\ A(\alpha(m,\ n),\ m/n)-1]$

$= [0,\ \cdots,\ A(\alpha(m,\ n),\ 1),\ \cdots,\ A(\alpha(m,\ n),\ 2),\ \cdots,\ A(\alpha(m,\ n),\ m/n)-1]$

这意味着$block(\alpha(m,\ n)+1,\ 0)$覆盖$m/n(i-1)$层的块。这样，

$$b_{\alpha(m,\ n)+1,0} = \lfloor m/n \rfloor \tag{10-7}$$

对于C部分，将公式（10-6）和式（10-7）代入公式（10-5），得到

$$n_{\alpha(m,\ n)+1,0}(b_{\alpha(m,\ n)+1,0}-1) \leqslant n\lfloor m/n \rfloor \leqslant m$$

概括地说，对于Q，

$$Q \leqslant n\alpha(m,\ n) + 8n + m = (\alpha(m,\ n) + 8)n + m$$

注意，m次find操作所花费的总时间等于find操作转换的贷方结点数与借方结点数之和。

仅有两种结点：贷方结点和借方结点。转换的贷方结点数以$(\alpha(m, n)+2)m$为界，转换的借方结点数以$(\alpha(m, n)+8)n+m$为界。

这样，推导出由find操作花费的平均时间是$O(\alpha(m, n))$。

由于$\alpha(m, n)$几乎是一个常数，所以分摊分析表明find操作花费的平均时间是一个常数。

10.7 一些磁盘调度算法的分摊分析

在计算机科学中，磁盘调度问题是很有趣的，实际上也是非常重要的问题。考虑单一磁盘，数据存储在不同的柱面上。在任何时间，都有检索数据的请求集，该请求集称为等待队列（waiting queue），这些请求称为等待请求（waiting requests）。磁盘调度问题是选择一个请求来服务。

例如，假定有一个检索数据的请求序列，数据分别存放在柱面16，2，14，5和21上。磁头初始时在柱面0上，同时假定从柱面i移动到柱面j需要的时间为$|i-j|$。现在证明磁盘调度算法怎样产生不同的结果。

先考虑先来先服务（first-come-first-serve，FCFS）算法。磁头先移动到柱面16，然后移动到柱面2，之后到14，等。用于请求的服务总时间是$|0-16|+|16-2|+|2-14|+|14-5|+|5-21|=16+14+12+9+16=67$。

设想使用另一种算法，称为最短寻道时间优先（shortest-seek-time-first，SSTF）算法。在该算法中，磁头总是移动到最近的柱面上。这样，磁头首先移动到柱面2，然后移动到5，之后到14，等。这样，用于请求的服务总时间是$|0-2|+|2-5|+|5-14|+|14-16|+|16-21|=2+3+9+2+5=21$，这比67小很多。

在本节中，假定在请求服务时，新的请求不断进来。同时假定等待队列能容纳m个请求，任何这m个请求以外的请求将被丢弃。换句话说，考虑请求的最大数是m。另一方面，也假定至少有W个请求，其中$W\geqslant 2$。磁盘总的柱面数是$Q+1$，令t_i表示第i个服务所花费的时间。主要关注$\sum_{i=1}^{m} t_i$的计算。正如所期望的，要找出$\sum_{i=1}^{m} t_i$的上界。如之前所做的，令$a_i(x)=t_i(x)+\phi_i(x)-\phi_{i-1}(x)$，其中$x$表示一个特定的算法，$\phi_i$表示请求的第$i$次服务之后的势能，$a_i(x)$表示第$i$次服务的分摊时间。

可以得到，

$$\sum_{i=1}^{m} a_i(x)=\sum_{i=1}^{m}(t_i(x)+\phi_i(x)-\phi_{i-1}(x))$$

$$=\sum_{i=1}^{m} t_i(x)+\phi_m(x)-\phi_0(x)$$

$$\sum_{i=1}^{m} t_i(x)=\sum_{i=1}^{m} a_i(x)+\phi_0(x)-\phi_m(x)$$

找出$\sum_{i=1}^{m} t_i(x)$上界的策略现在变为找出$a_i(x)$的上界，令此上界表示为$A(x)$，那么

$$\sum_{i=1}^{m} t_i(x)\leqslant mA(x)+\phi_0(x)-\phi_m(x)$$

在本节的剩余部分，将讨论使用分摊分析如何分析两磁盘调度算法：SSFT算法和SCAN算法。

最短寻道时间优先（SSTF）算法的分析

在SSTF算法中，等待队列将服务最近的请求。考虑第i次服务之后的情况，$1 \leqslant i \leqslant m$。令$N_i(SSTF)$表示在当前服务方向上等待的请求数，$L_i(SSTF)$表示磁头与最近请求之间的距离，$D_i(SSTF)$表示在服务方向上的柱面数。$L_i(SSTF)$和$D_i(SSTF)$的定义如图10-50所示。

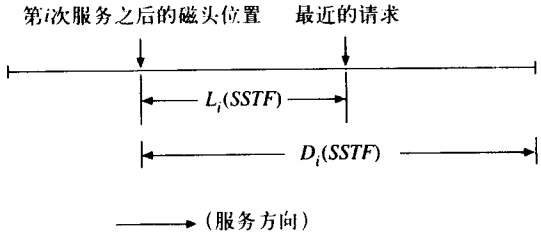


图10-50 $L_i(SSTF)$与$D_i(SSTF)$的定义

SSTF算法的势能函数可以定义如下：

$$\phi_i(SSTF) = \begin{cases} L_i(SSTF) & \text{如果} N_i(SSTF) = 1 \\ \min\{L_i(SSTF), D_i(SSTF)/2\} & \text{如果} N_i(SSTF) > 1 \end{cases} \tag{10-8}$$

从公式（10-8）中，很容易证明

$$\phi_i(SSTF) \geqslant 0 \tag{10-9}$$

下面将证明

$$\phi_i(SSTF) \leqslant Q/2 \tag{10-10}$$

考虑两种情况：

情况1. $N_i(SSTF) = 1$。

由于假设$W \geqslant 2$，在相反的方向上至少有一个请求，如图10-51所示。令在磁头与反方向上最近邻请求的距离为b，那么$b + L_i(SSTF) \leqslant Q$。而$L_i(SSTF) \leqslant b$；否则，将颠倒服务方向。

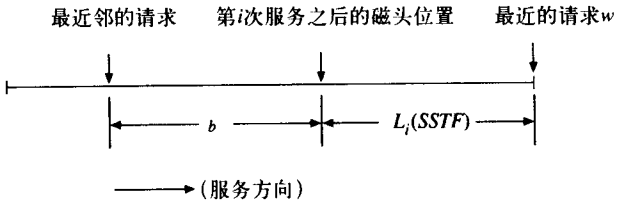


图10-51 $N_i(SSTF) = 1$的情况

所以，

$2L_i(SSTF) \leqslant L_i(SSTF) + b \leqslant Q$

或者，$L_i(SSTF) \leqslant Q/2$。

这意味着$\phi_i(SSTF) = L_i(SSTF) \leqslant Q/2$。

这是由于$\phi_i(SSTF)$的定义是公式(10-8)所描述的。

情况2. $N_i(SSTF) \geqslant 2$。

由于$D_i(SSTF) \leqslant Q$，$D_i(SSTF)/2 \leqslant Q/2$。所以，根据公式（10-8），

$$\begin{aligned}\phi_i(SSTF) &= \min\{L_i(SSTF), D_i(SSTF)/2\} \\ &\leqslant D_i(SSTF)/2\end{aligned}$$

$\leqslant Q/2$

很容易得到$\phi_i(SSTF)$的另一个上界，即

$$\phi_i(SSTF) \leqslant L_i(SSTF) \tag{10-11}$$

上面的等式用于求解下式的上界

$$a_i(SSTF) = t_i(SSTF) + \phi_i(SSTF) - \phi_{i-1}(SSTF)$$

为了求解，需要考虑下面两种情况：

情况1. $N_{i-1}(SSTF) = 1$。

在此情况下，根据公式(10-8)，可得

$$\phi_{i-1}(SSTF) = L_{i-1}(SSTF) = t_i(SSTF) \tag{10-12}$$

这样，

$$\begin{aligned} a_i(SSTF) &= t_i(SSTF) + \phi_i(SSTF) - \phi_{i-1}(SSTF) \\ &= t_i(SSTF) + \phi_i(SSTF) - L_{i-1}(SSTF) \\ &\leqslant t_i(SSTF) + Q/2 - t_i(SSTF) \quad \text{（根据公式（10-10）和公式（10-12））} \\ &= Q/2 \end{aligned}$$

情况2. $N_{i-1}(SSTF) \geqslant 2$。

这又有两种子情况。

情况2.1 $N_{i-1}(SSTF) \geqslant 2$，且$L_{i-1}(SSTF) \leqslant D_{i-1}(SSTF)/2$。

对于此种情况，根据公式（10-8），

$$\phi_{i-1}(SSTF) = \min\{L_{i-1}(SSTF), D_{i-1}(SSTF)/2\} \leqslant L_{i-1}(SSTF)$$

由此，正如情况1一样，可以证明

$$a_i(SSTF) \leqslant Q/2$$

情况2.2 $N_{i-1}(SSTF) > 1$，且$L_i(SSTF) > D_{i-1}(SSTF)/2$。

对于此种情况，根据公式（10-9），

$$\phi_{i-1}(SSTF) = D_{i-1}(SSTF)/2$$

由于$N_{i-1}(SSTF) > 1$，除了最近请求外，在服务方向上一定还有一些请求。令c是在服务方向上最近的请求与第二近的请求间的距离。图10-52表示此种情况。

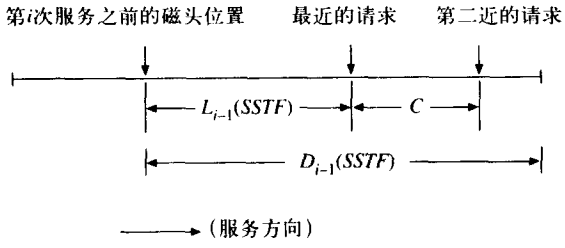


图10-52 $N_{i-1}(SSTF) > 1$，且$L_{i-1}(SSTF) > D_{i-1}(SSTF)/2$的情况

由于使用SSTF算法，$L_i(SSTF) \leqslant c$。这样，可得

$$L_i(SSTF) \leqslant c \leqslant D_{i-1}(SSTF) - L_{i-1}(SSTF) \tag{10-13}$$

这样，

$$
\begin{aligned}
a_i(SSTF) &= t_i(SSTF) + \phi_i(SSTF) - \phi_{i-1}(SSTF) \\
&\leqslant t_i(SSTF) + L_i(SSTF) - \phi_{i-1}(SSTF) \text{（根据公式(10-11)）} \\
&\leqslant t_i(SSTF) + (D_{i-1}(SSTF) - L_{i-1}(SSTF)) - \phi_{i-1}(SSTF) \text{（根据公式(10-13)）} \\
&= L_{i-1}(SSTF) + D_{i-1}(SSTF) - L_{i-1}(SSTF) - \phi_{i-1}(SSTF) \text{（因为}t_i(SSTF) = L_{i-1}(SSTF)\text{）} \\
&= D_{i-1}(SSTF) - \phi_{i-1}(SSTF) \\
&= D_{i-1}(SSTF)/2) \text{（因为 } \phi_{i-1}(SSTF) = D_{i-1}(SSTF)/2\text{）} \\
&\leqslant Q/2 \text{（因为}D_{i-1}(SSTF) \leqslant Q\text{）}
\end{aligned}
$$

将情况1和2合并，得到

$$a_i(SSTF) \leqslant Q/2 \tag{10-14}$$

将公式（10-14）、（10-9）和（10-10）合并，可以找出 $\sum_{i=1}^{m} t_i(SSTF)$的上界为：

$$\sum_{i=1}^{m} t_i(SSTF) \leqslant \sum_{i=1}^{m} a_i(SSTF) + \phi_0(SSTF) - \phi_m(SSTF)$$

因为$\phi_0(SSTF)$的最大值是$Q/2$，$\phi_m(SSTF)$的最小值是0，所以有

$$
\begin{aligned}
\sum_{i=1}^{m} t_i(SSTF) &\leqslant \sum_{i=1}^{m} a_i(SSTF) + \phi_0(SSTF) - \phi_m(SSTF) \\
&\leqslant mQ/2 + Q/2 - 0 \\
&= (m+1)Q/2
\end{aligned} \tag{10-15}
$$

对于SSTF算法，等式（10-15）说明m个请求的总时间花费是$(m + 1)Q/2$。或者等价地说，该算法需要的时间花费是$Q/2$。

为了证明上界$(m + 1)Q/2$不能再紧致。考虑m个请求的序列，分别位于如下柱面：

$$(Q/2,\ Q,\ (0,\ Q/2)^{(m-3)/2},\ 0)$$

其中x^y表示$\overbrace{(x,x,\cdots,x)}^{y}$。假如在任何时间等待的请求数是2，$(m-3)$能被2整除。在此情况下，磁头将在柱面$Q/2$与0之间摆动，表示如下：

$(Q/2,\ 0)^{(m-1)/2},\ Q)$

这样，该请求总的服务时间是

$(Q/2 + Q/2)\ (m-1)\ /2 + Q = (m + 1)Q/2$

这意味着公式（10-15）表示的上界不能进一步紧致。

SCAN算法的分摊分析

如前面所讨论的，SSTF算法可能迫使磁头来回摆动。通过在当前搜索方向上选取最近请求，SCAN算法避免了该问题。这样，如果磁头正移向一个方向，那么它可以继续进行到在该方向上没有请求为止。此时，磁头再倒转方向。

考虑第i次服务之后的状态。现在定义两个术语$N_i(SCAN)$和$D_i(SCAN)$如下：如果在第$(i + 1)$次服务时，没有改变搜索方向，那么$N_i(SCAN)$和$D_i(SCAN)$分别定义为已经服务的请求数及磁头在当前搜索方向上移动的距离；否则，$N_i(SCAN)$和$D_i(SCAN)$都置为0。$N_0(SCAN)$和$D_0(SCAN)$都是0，SCAN的势能函数可定义为

$$\phi_i(SCAN) = N_i(SCAN)Q/W - D_i(SCAN) \tag{10-16}$$

对于SSTF，将证明$a_i(SSTF) \leqslant Q/2$。对于SCAN，将证明$a_i(SCAN) \leqslant Q/W$。

考虑第i次服务，令$t_i(SCAN)$表示第i次服务的时间，又有两种情况需要考虑。

情况1. 不改变第$(i+1)$次服务的扫描方向。在这种情况下，显然可以得到

$N_i(SCAN) = N_{i-1}(SCAN) + 1$以及

$$\begin{aligned}
D_i(SCAN) &= D_{i-1}(SCAN) + t_i(SCAN) \\
a_i(SCAN) &= t_i(SCAN) + \phi_i(SCAN) - \phi_{i-1}(SCAN) \\
&= t_i(SCAN) + (N_i(SCAN)Q/W - D_i(SCAN)) \\
&\quad - (N_{i-1}(SCAN)Q/W - D_{i-1}(SCAN)) \\
&= t_i(SCAN) + ((N_{i-1}(SCAN) + 1)Q/W - (D_{i-1}(SCAN) \\
&\quad + t_i(SCAN)) - (N_{i-1}(SCAN)Q/W - D_{i-1}(SCAN) \\
&= Q/W
\end{aligned}$$

情况2. 改变第$(i+1)$次服务的扫描方向。在这种情况下，$N_i(SCAN) = D_i(SCAN)$，$\phi_i(SCAN) = 0$。

由于磁头初始定位在柱面0，等待队列的最小值假设为W，这样在一次扫描中对请求服务的最小值也是W。也就是，$N_{i-1}(SCAN) > (W-1)$。这样，

$$\begin{aligned}
a_i(SCAN) &\leqslant t_i(SCAN) - ((W-1)Q/W - D_{i-1}(SCAN)) \\
&\leqslant (t_i(SCAN) + D_{i-1}(SCAN) - Q) + Q/W
\end{aligned}$$

由于在一次扫描中磁头移动的最大距离是Q，

$$t_i(SCAN) + D_{i-1}(SCAN) \leqslant Q$$

所以，

$$a_i(SCAN) \leqslant Q/W \tag{10-17}$$

上面的讨论证明了$a_i(SCAN) \leqslant A(SCAN) = Q/W$。

其中$A(SCAN)$是$a_i(SCAN)$的上界。

现在，可以得到

$$\begin{aligned}
\sum_{i=1}^{m} t_i(SCAN) &\leqslant \sum_{i=1}^{m} a_i(SCAN) + \phi_0(SCAN) - \phi_m(SCAN) \\
&\leqslant mQ/W + \phi_0(SCAN) - \phi_m(SCAN)
\end{aligned}$$

但是，$\phi_0(SCAN) = 0$。这样，

$$\sum_{i=1}^{m} t_i(SCAN) \leqslant mQ/W - \phi_m(SCAN)$$

为估计$\phi_m(SCAN)$，再次考虑两种情况：

情况1. $N_m(SCAN) = D_m(SCAN) = 0$ (10-18)

在这种情况下，$\phi_m(SCAN) = 0$。

情况2. $N_m(SCAN)$之一与$D_m(SCAN)$不为0。

在这种情况下，$N_m(SCAN) \geqslant 1$，且$D \leqslant D_m(SCAN) \leqslant Q$。

这样，

$$\phi_m(SCAN) = N_m(SCAN)Q/W - D_m(SCAN)$$
$$\geqslant Q/W - Q \tag{10-19}$$

注意到$Q/W-Q\leqslant 0$，我们合并式（10-18）和式（10-19），可以得出结论

$$\phi_m(SCAN)\geqslant Q/W - Q \tag{10-20}$$

最后，可以得到

$$\sum_{i=1}^{m} t_i(SCAN)\leqslant mQ/W - \phi_m(SCAN)$$
$$= mQ/W - (Q/W-Q)$$
$$\leqslant (m-1)Q/W + Q \tag{10-21}$$

式（10-21）表明由这m次请求所需的时间耗费不大于$(m-1)Q/W + Q$。对于SCAN算法，一次请求的平均时间耗费不大于$((m-1)Q/W + Q)/m$。

由于之前所证明的，现在我们可以证明上界$(m-1)Q/W + Q$不可能再进一步紧致了。我们可以证明该结论，通过考虑分别定位在柱面

$$((Q^4,\ 0^4)^{(m-1)/8},\ Q)$$

上的m次请求序列，并假设在任何时间，等待请求的数量是4。令$W = 4$，假设$(m-1)$能被8整除。因此，该请求序列也可被序列$((Q^4,\ 0^4)^{(m-1)/8},\ Q)$调度和处理。

该序列总的服务时间是$((m-1)/8)2Q + Q = ((m-1)/(2W))2Q + Q = (m-1)Q/W + Q$。

这说明在式（10-21）中表示的总的时间上界不能被紧致。

现在，通过比较式（10-15）和式（10-21）来总结本节内容。从式（10-15），可以得到

$$t_{ave}(SSTF) = \sum_{i=1}^{m} t_i(SSTF)/m\leqslant (m+1)Q/(2m) \tag{10-22}$$

随着$m\to\infty$，可以得到

$$t_{ave}(SSTF)\leqslant Q/2 \tag{10-23}$$

从式（10-21），可以证明

$$t_{ave}(SCAN)\leqslant Q/W \tag{10-24}$$

由于$W\geqslant 2$，通过对这两个算法的分摊分析可以得出结论$t_{ave}(SCAN)\leqslant t_{ave}(SSTF)$。

10.8 实验结果

分摊分析通常涉及很多数学背景，对于许多研究者来说，结果应该是怎样的并不那么直观清晰。例如，不相交集合并算法的平均性能几乎是一个常数，这对任何人来说都不是那么明显。所以，我们实现了不相交集合并算法，使用的数据有10 000个元素，由随机数生成器来决定下一个操作是查找或是连接。程序用Turbo Pascal编写，并在IBM PC上执行。表10-9总结了结论。

表10-9 不相交集合并算法的分摊分析实验结果

操作次数	总时间（毫秒）	平均时间（微秒）
2000	100	50
3000	160	53
4000	210	53
5000	270	54
7000	380	54
9000	490	54
11 000	600	55
13 000	710	55
14 000	760	54
15 000	820	55

从实验结果可知，分摊分析准确地预测了

不相交集合并算法的行为。对于一个操作序列，有关的总时间在递增。然而，每个操作的平均时间是个常数，正如分摊分析所预测的。

10.9 注释与参考

术语分摊分析首先在文献Tarjan(1985)中提出。然而，此概念在1985前已由许多研究者在使用。例如，在文献Brown and Tarjan(1980)中对2-3树的分析，在文献Huddleston and Mehlhorn(1982)中对弱B树的分析都使用了此概念，尽管在当时并没有使用“分摊”。

斜堆的分摊分析可在文献Sleator and Tarjan(1986)中找到，文献Mehlhorn and Tsakalidis(1986)对AVL树给出了分摊分析，自组织顺序搜索启发式的分摊分析出现在文献Bentley and McGeoch(1985)。配对堆的分摊分析可参阅文献Fredman，Sedgewick，Sleator and Tarjan(1986)，不相交集合并算法的分摊分析可在文献Tarjan(1983)和Tarjan and Van Leeuwen(1984)中找到。

分摊分析经常用在涉及相关数据结构及反复应用在该数据结构上的操作序列的算法中。例如，在文献Fu and Lee(1991)中，需要一个数据结构使树可以动态高效地得到处理。也就是许多树操作，像检查边、插入边等都可以实现。在这种情况下，可以使用动态树（Sleator and Tarjan，1983）并进行分摊分析。

分摊分析也应用于分析一系列操作的一些实际策略中。文献Chen，Yang and Lee(1989)对一些磁盘调度方法进行了分摊分析。

非常少的教科书讨论分摊分析概念，但文献Purdom and Brown(1985a)和Tarjan(1983)对此做了讨论。

10.10 进一步的阅读资料

对于分摊分析的回顾和介绍性的讨论可参阅文献Tarjan(1985)。对于分摊分析，下面的论文都是最近出版的，对于进一步的阅读有极高的推荐价值：Bent，Sleator and Tarjan(1985)；Eppstein，Galil，Italiano and Spencer(1996)；Ferragina(1997)；Henzinger(1995)；Italiano(1986)；Karlin，Manasse，Rudolph and Sleator(1988)；Kingston(1986)；Makinen(1987)；Sleator and Tarjan(1983)；Sleator and Tarjan(1985a)；Sleator and Tarjan(1985b)；Tarjan and Van Wyk(1988)；以及Westbrook and Tarjan(1989)。

习题

10.1 对于在10.1节中讨论的栈问题，证明：不使用势能函数上界是2，并给出例子说明该上界也是紧致的。

10.2 设想一个人的唯一收入是月薪，假如是每月k个单位。只要他的银行账户有足够的钱，他就可以花费任意数量的钱。每月他将k个单位的工资存入银行账户中。你能对该人的行为进行分摊分析吗？（定义你的问题，注意，他不能在任何时间取出大量的钱。）

10.3 分摊分析隐含着相关的数据结构有某种自组织机制。换句话说，当它变得非常差时，有机会变向好的方向。在这种情况下，通过分摊分析能分析这种杂乱的情况吗？对该主题做些研究，也许你能发表一些论文呢。

10.4 选择并实现在本章所介绍的任意算法。完成一些实验，检验对分摊分析的理解。

10.5 阅读关于动态树数据结构的文献Sleator and Tarjan(1983)。

第11章 随机算法

随机算法（randomized algorithm）的概念相对较新。本书到目前为止所介绍的每一种算法中的每一步都是确定的。也就是说，在执行算法的过程中，从未做出任何随心所欲的选择。在本章将要介绍的随机算法可以做出任意的选择，这意味着可随机地执行某些动作。

由于某些动作的执行是随机的，后面将要介绍的随机算法具有下面两个属性：

(1) 在最优化问题的情况中，随机算法给出一个最优解。但是由于在算法中随机地采取动作，因此随机最优算法（randomized optimization algorithm）的时间复杂度也是随机的。这样，随机最优算法的平均情况时间复杂度远比它的最坏情况时间复杂度重要。

(2) 在判定问题的情况中，随机算法有时会出错。然而，产生错误的概率是非常小的；另外，随机算法并非总是有用。

11.1 解决最近点对问题的随机算法

我们曾在第4章介绍过最近点对问题。在第4章中，我们已经证明这个问题可通过分治法在$O(n \log n)$时间内得到解决，这是最坏情况下的时间复杂度。在本节中，将介绍一种随机算法，使用该算法解决这个问题的平均情况时间复杂度是$O(n)$。

令x_1，x_2，…，x_n为二维平面上的n个点。最近点对问题是找到最近的一对点x_i和x_j，使得x_i与x_j之间的距离是所有可能的点对距离中最小。解决该问题的直接方法是计算出所有的$n(n-1)/2$个点对之间的距离，并在这些距离中找出最小值。

该随机算法的主要思想基于下面的观察：*如果两个点x_j和x_k彼此远离，那么它们之间的距离不可能最小，因此可以被忽略。*由于这种思想，随机算法首先将点分成几组，使得各组内的点相互之间相隔较近，然后，只需计算同一组内的点之间的最小距离。

研究图11-1中的六个点。如果将这六个点分成三组：$S_1 = \{x_1, x_2\}$，$S_2 = \{x_5, x_6\}$和$S_3 = \{x_3, x_4\}$，那么，只需要计算三个距离，即$d(x_1, x_2)$，$d(x_5, x_6)$和$d(x_3, x_4)$。随后，从这三个距离中找出最小者即可。如果没有将这些点分成组，那么必须计算$(6 \cdot 5)/2 = 15$个距离。

当然，这种讨论会相当令人误解，因为不能保证该策略会起作用。事实上，这种策略可看作是分而不治策略（divide-without-conquer strategy）。存在划分过程，但是没有合并过程。下面参见图11-2，可以看出最近的点对是（x_1，x_3）。但是，已将这两个点分在不同的组中。

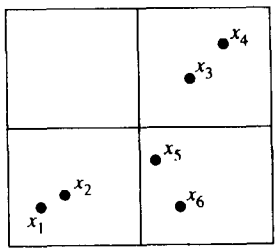


图11-1 点的划分

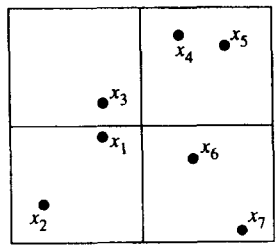


图11-2 表明组内部距离重要性的一种情况

如果将整个空间划分成边长等于δ的正方形，其中δ不小于最短距离。那么，当所有的组

内距离计算后，可以使正方形的边长加倍，构成一个更大的正方形，这样最短距离一定在某个扩展后的方形中。图11-3表示对应于某个小方形扩展后的四个大方形。每个扩展后的方形属于一种特定的类型，如图11-3所示。

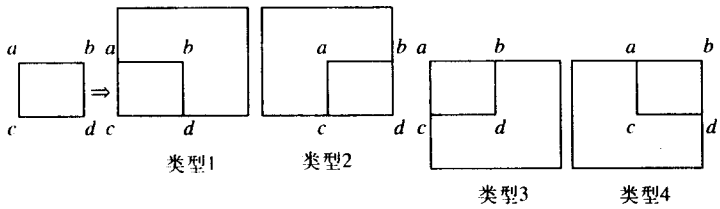


图11-3 四个扩展的正方形的构造

设想整个空间已划分成若干个宽度为δ的正方形，这些方形的扩展产生四种增大的方形集合，对应于类型1、类型2、类型3和类型4分别表示为T_1、T_2、T_3和T_4。典型情况如图11-4所示。

当然，核心的问题是找到适当的网格大小δ。如果δ的值过大，那么最初的方形区域太大，要计算大量的距离。事实上，当δ的值太大时，问题等于没有划分而蜕变成原始问题。另一方面，δ值也不能太小，它不能比最短距离还小。在随机算法中，随机选择一个点集，找到这些点集的最短距离，令这个最短距离为δ。

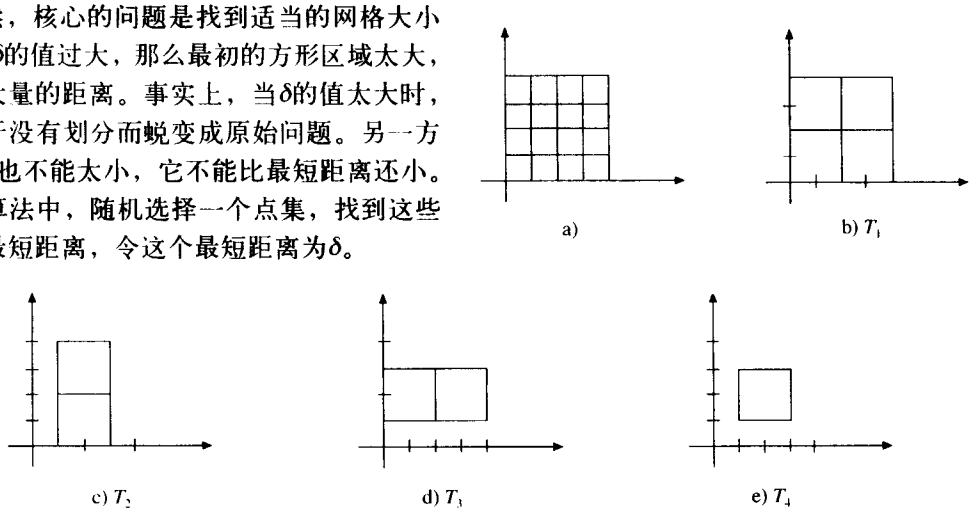


图11-4 四种增大的方形

算法11-1 寻找最近点对的随机算法

输入：包含n个元素x_1，x_2，…，x_n的集合S，其中$S \subseteq R^2$。

输出：S中的最近点对。

步骤1. 随机选择一个集合$S_1 = \{x_{i1}, x_{i2}, \cdots, x_{im}\}$，其中 $m = n^{\frac{2}{3}}$。
找出S_1中的最近点对，令这对点间的距离为δ。

步骤2. 构造一个边长为δ的方形集合T。

步骤3. 通过加倍边长到2δ，由T构造出T_1，T_2，T_3和T_4四类方形集合。

步骤4. 对每个T_i，分解S，使得$S = S_1^{(i)} \cup S_2^{(i)} \cup \cdots \cup S_j^{(i)}$，$1 \leqslant i \leqslant 4$，其中$S_j^{(i)}$为$S$与方形区域$T_i$的非空交集。

步骤5. 对任意x_p，$x_q \in S_j^{(i)}$，计算$d(x_p, x_q)$。令x_a和x_b是在这些点对中距离最短的点对。返回最近点对x_a和x_b。

例11-1

给出有27个点的集合S，如图11-5所示。在步骤1中，随机选择 $27^{\frac{2}{3}} = 9$ 个元素x_1，x_2，…，x_9。其中最近点对为(x_1, x_2)。在步骤2中，运用x_1和x_2之间的距离δ作为边长构造36个符合要求的方形。将会有如下四个方形集合T_1，T_2，T_3和T_4：

$T_1=\{[0:2\delta,\ 0:2\delta],\ [2\delta:4\delta,\ 0:2\delta],\ [4\delta:6\delta,\ 0:2\delta],\ \cdots,\ [4\delta:6\delta,\ 4\delta:6\delta]\}$

$T_2=\{[\delta:3\delta,\ 0:2\delta],\ [3\delta:5\delta,\ 0:2\delta],\ \cdots,\ [3\delta:5\delta,\ 4\delta:6\delta]\}$

$T_3=\{[0:2\delta,\ \delta:3\delta],\ [2\delta:4\delta,\ \delta:3\delta],\ [4\delta:6\delta,\ \delta:3\delta],\ \cdots,\ [4\delta:6\delta,\ 3\delta:5\delta]\}$

$T_4=\{[\delta:3\delta,\ \delta:3\delta],\ [3\delta:5\delta,\ \delta:3\delta],\ \cdots,\ [3\delta:5\delta,\ 3\delta:5\delta]\}$

相互距离计算的总数量为

$N(T_1): C_2^4+C_2^3+C_2^2+C_2^3+C_2^3+C_2^3+C_2^3+C_2^3+C_2^3=28$

$N(T_2): C_2^3+C_2^3+C_2^2+C_2^5+C_2^3+C_2^4=26$

$N(T_3): C_2^4+C_2^3+C_2^3+C_2^3+C_2^4+C_2^3=24$

$N(T_4): C_2^3+C_2^4+C_2^3+C_2^5=22$

在这28 + 24 + 22 + 26 = 100点对中，最近的一对在$[3\delta:5\delta,\ 3\delta:5\delta]$中。

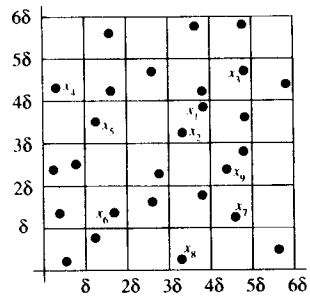


图11-5 说明随机最近点对算法的例子

11.2 随机最近点对问题的平均性能

在上节所介绍的随机最近点对问题中，第一步是在$n^{\frac{2}{3}}$个点中找到最近点对。该最近点对可通过递归地应用此随机算法求解。也就是，可以从原始点集中随机选出$(n^{\frac{2}{3}})^{\frac{2}{3}}=n^{\frac{4}{9}}$个点，在$n^{\frac{4}{9}}$个点集中找到最近点对。可以使用直接方法进行$(n^{\frac{4}{9}})^2=n^{\frac{8}{9}}<n$次距离的计算。但是，这并不意味着步骤1可以在$O(n)$时间内执行完毕。现在暂时不考虑步骤1的时间复杂度，随后证明步骤1的确只花费$O(n)$的时间。

显然，步骤2和步骤3能够在$O(n)$的时间内执行完毕。步骤4可以利用散列技术。利用散列技术容易地确定哪个点位于特定的方形中，这意味着步骤4在时间$O(n)$内计算完毕。

步骤5中期望的距离计算次数绝不是能容易地确定下来的。事实上，并没有计算距离期望次数的公式。确实，我们能证明在步骤5中，距离计算总数的期望值的概率为$1-2e^{-cn^{\frac{1}{6}}}$，随着n增大，该概率能快速地接近1。或者，从另一角度来说，在$O(n)$时间内计算出距离总数期望值的概率是非常高的。

为什么能得出上述的结论？注意在随机最近点对算法中，方形边长为δ，用T表示该划分，且对该划分所需计算的距离总数为$N(T)$。接下来证明存在一个特殊的划分称为T_0，边长为δ_0，有以下两个性质：

(1) $n\leqslant N(T_0)\leqslant C_0 n$。

(2) $\delta\leqslant\sqrt{2}\,\delta_0$的概率为$1-2e^{-cn^{\frac{1}{6}}}$，此概率非常高。

我们之后会解释该划分的存在原因。首先假定它是正确的，并从这里开始推导。

假定将方形边长从δ_0扩大四倍到$4\delta_0$，扩大四倍导致16个方形集合，将它们表示为$T_i(i=1,2,\cdots,16)$。由于$\delta\leqslant\sqrt{2}\delta_0$的概率为$1-2e^{-cn^{\frac{1}{6}}}$，那么$T$中任意方形至少属于$T_i(i=1,2,\cdots,16)$中一个的概率为$1-2e^{-cn^{\frac{1}{6}}}$。对于划分$T_i$中距离计算的总数为$N(T_i)$，那么

$$N(T)\leqslant\sum_{i=1}^{16}N(T_i)$$

为真的概率为$1-2e^{-cn^{\frac{1}{6}}}$。

现在计算$N(T)$。从$N(T_i)$开始，在T_i中每个方形是T_0中方形的16倍，令T_0中内部元素最多的方形有k个元素，S_{ij}表示T_0中16个方形集合属于T_i中的第j个方形区域，令在S_{ij}中拥有最多元素的区域最多有k_{ij}个元素，那么T_0中距离计算的总数比$\sum_j k_{ij}(k_{ij}-1)/2$大，也就是$\sum_j k_{ij}(k_{ij}-1)/2 \leqslant N(T_0) \leqslant C_0 n$。并且，$T_i$中第$j$个方形距离计算的总数小于$16k_{ij}(16k_{ij}-1)/2$。因此，$N(T_i) \leqslant \sum_j 16k_{ij}(16k_{ij}-1)/2 \leqslant C_i n$，其中$C_i$为常数。所以，$N(T) \leqslant \sum_{i=1}^{16} N(T_i) = O(n)$的概率为$1-2e^{-cn^{\frac{1}{6}}}$。

由于n很大，$e^{-cn^{\frac{1}{6}}}$快速缩减为0，所以$N(T) \leqslant O(n)$的概率为1。

接下来，将解释为什么能得出存在具有以上两种性质的划分结论。原因是很复杂的，在给出主要论证前，先定义一些术语。

令D为点集的一个划分，也就是，$S = S_1 \cup S_2 \cup \cdots \cup S_r$且当$i \neq j$时，$S_i \cap S_j = \phi$。如果$T \subseteq S$是$m$个元素的一种选择，那么当$T$中至少有两个元素是从某个$i$的划分的同一部分$S_i$中选出的，那么称$T$是$D$的一个后继（success）。如果$D'$是$S$的另一种划分，如果对于任意的$m$，$D$中拥有$m$个元素的后继的概率大于或等于在$D'$上有$m$个元素的后继，那么称$D$支配$D'$。

根据上面的定义，易见下面的陈述是正确的：

(1) 划分(2，2，2)支配(3，1，1，1)。这意味着有六个元素的集合划分成3部分的所有划分法均优于将同样的集合分成一个3和3个单数1的划分。为什么能支配？理由很简单。对于划分(2，2，2)，从同一方形中选出2个点的概率远比划分(3，1，1，1)大得多，后者仅有一个方形多于1个元素。

(2) 划分(3，3) 支配(4，1，1)。

(3) 划分(4，4) 支配(5，1，1，1)。

(4) 划分(p, q) $(p \geqslant 5, q \geqslant 5)$，支配$(l, 1, 1, \cdots, 1)$，其中1的个数为$p+q-l$，$l(l-1) \leqslant p(p-1)+q(q-1)l \leqslant l+1$。

由此可见，如果划分D支配D'，那么D'所需计算的距离总数小于等于D计算的距离数。

令$N(D)$为D中所需计算的距离总数，由前面的定义和讨论可知，显然对有限点集合S的每一个划分D，存在一个同样集合的划分D'，使得$\lambda N(D) \leqslant N(D')$，其中$D$支配$D'$，并且$D'$的所有子集中除去1个外全为单元素集合，$\lambda$是一个小于或等于1的正数。

假定D是集合S，$|S| = n$且$n \leqslant N(D)$的一个划分，那么令D'为一个划分使得$S = H_1 \cup H_2 \cup \cdots \cup H_{k'}$，由$D$所支配，其中$|H_1| = b$，$|H_i| = 1$，$i = 2, 3, \cdots, k'$，$\lambda n \leqslant N(D')$。这表明$\lambda n \leqslant b(b-1)/2$，那么$b \geqslant \sqrt{2\lambda n}$，令$c = \sqrt{2\lambda}$，有$b \geqslant c\sqrt{n}$。

对于点的每一次选择，点不是从H_1中选出的概率为$1-b/n \leqslant 1-c/\sqrt{n}$。假设$n^{\frac{2}{3}}$个点是从$S$中随机选出的，那么它们都不是从$H_1$中选出的概率小于

$$\left(1-\frac{c}{\sqrt{n}}\right)^{n^{\frac{2}{3}}} = \left(\left(1-\frac{c}{\sqrt{n}}\right)^{\sqrt{n}}\right)^{n^{\frac{1}{6}}} = e^{-cn^{\frac{1}{6}}}$$

当从S中随机选出$n^{\frac{2}{3}}$个点时，至少有两个点来自于H_1的概率大于$1-2e^{-cn^{\frac{1}{6}}}$。由于D支配D'，可以推出，如果点是从S中随机选出的，那么有两个点来自于D的同一个集合的概率至少为$\mu(n) = 1-2e^{-cn^{\frac{1}{6}}}$，其中$c$为常数。

仍有一个问题尚待解决，划分T_0是否满足$n \leqslant N(T_0) \leqslant C_0 n$，其中$C_0$为常数？这个划分可通过下面的算法建立。

算法11-2 划分算法

输入：有n个点的集合S。

输出：划分T_0满足$n \leqslant N(T_0) \leqslant c_0 n$。

步骤1. 找出划分T，其网格恰好使T中的每个方形区域至多包含S中的一个点，且S中没有点在网格线上，因此，$N(T)=0$。

步骤2. 当$N(T)<n$时，双倍扩大T网格的规模。

现在解释算法11-2的含义。算法的第一步将整个点集划分成正方形区域，使得每个方形区域中至多有一个点。显然，在这种情况下，$N(T)$为0，对我们毫无用处。所以，需要逐渐地加倍方形的大小，直到遇到算法11-1的第一次划分使得$N(T) \geqslant n$为止。

考虑如图11-6所示的例子，其中$n = 10$。图11-6a显示最初的划分，即步骤1的结果；现在使网格扩大1倍，得到如图11-6b所示的划分。现在，总共所需计算的距离总数为$N(T) = 4 \times 3/2 + 2 \times 1/2 + 4 \times 3/2 = 6 + 1 + 6 = 13 > n = 10$。因此，这个特殊的划分是满足要求的$T_0$。注意在此例中，$C_0 = 1.3$。

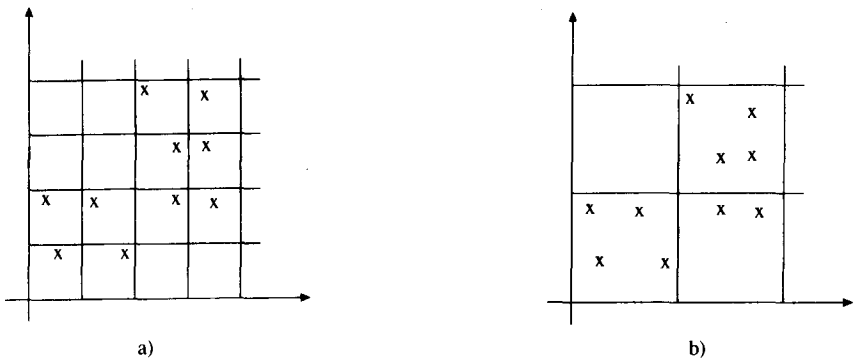


图11-6 说明算法11-1的例子

令δ_0表示T_0的网格边长，那么对任意选出$n^{\frac{2}{3}}$点的集合S_a，如果S_a中存在两个点在T_0中的同一方形区域中，那么S_a中的最小距离小于或等于$\sqrt{2}\,\delta_0$；不等式成立的概率大于$\mu(n)$。至此，已经证明，存在特殊的T_0，其大小为δ_0，且有如下两个性质：

(1) $n \leqslant N(T_0) \leqslant C_0 n$。

(2) $\delta \leqslant \sqrt{2}\,\delta_0$的概率为$1-2e^{-cn^{\frac{1}{6}}}$，概率非常高，其中$\delta_0$为$n^{\frac{2}{3}}$个随机选出点集的最短距离。

现在解释算法11-1中步骤5只需$O(n)$步。注意步骤1是递归的。但是，因为步骤5是最重要的一步，可以得出整个随机查找最临近点对算法的时间复杂度为$O(n)$。

11.3 素数测试的随机算法

素数问题（prime number problem）是确定一个给定的正数是否素数。这是一个很难的问题，直到2004年还没有出现多项式时间复杂度的算法来解决该问题。在本节中，将介绍一种随机算法，该随机算法执行一系列m次的测试。如果这些测试中任意一次成功，那么可推断其为合数，且能保证该结论是绝对正确的。另一方面，如果所有m次测试全都失败，那么可推断这个数为素数。但这次不能确保一定是正确的。结论正确的概率为$1-2^{-m}$。因此，如果m足够

大，可以有足够的信心得出该推断。

算法11-3 随机素数测试算法

输入：正数N，参数m，其中$m \geqslant \lceil \log_2 \varepsilon \rceil$。

输出：N是否素数，且正确的概率为$1-\varepsilon = 1-2^{-m}$。

步骤1. 随机地选出m个数b_1，b_2，…，b_m，$1<b_1$，b_2，…，$b_m<N$。

步骤2. 对每个b_i，测试$W(b_i)$是否成立，其中$W(b_i)$定义如下：

(1) $b_i^{N-1} \neq 1 \bmod N$。或者

(2) $\exists j$，使得$\frac{N-1}{2^j}=k$为整数，b_i^k-1和N的最大公约数小于N大于1。

如果$W(b_i)$成立，那么返回N为合数；否则，返回N为素数。

现在通过一些例子说明该随机算法，考虑$N = 12$，假定选择2，3和7。$2^{12-1} = 2048 \neq 1 \bmod 12$。由此，我们推断出12为合数。

考虑$N = 11$，假定选择2、5和7。

$b_1 = 2 : 2^{11-1} = 1024 \equiv 1 \bmod 11$。

令$j = 1$。$(N-1)/2^j = (11-1)/2^1 = 10/2 = 5$。

此$j = 1$是使$(N-1)/2^j$为整数仅有的j。

但是，$2^5-1 = 31$和11的最大公约数为1。

$W(2)$不成立。

$b_2 = 5 : 5^{11-1} = 5^{10} = 9\ 765\ 625 \equiv 1 \bmod 11$。

由上面的讨论，该$j = 1$是使$(N-1)/2^j = k = 5$为整数的j。

$b_2^k = 5^5 = 3\ 125$

$5^5-1 = 3\ 124$和11的最大公约数为11。

$W(5)$也不成立。

$b_3 = 7 : 7^{11-1} = 282\ 475\ 249 \equiv 1 \bmod 11$。

同理，令$j = 1$。

$b_3^k = 7^5 = 16\ 807$

$7^5-1 = 16\ 806$和11的最大公约数为1。

$W(7)$也不成立。

可以推论11为素数，且正确的概率为$1-2^{-3} = 1-1/8 = 7/8$。

如果还选择3，那么同样可证$W(3)$也不成立。现在$m = 4$，那么正确的概率增至$1-2^{-4} = 15/16$。

对于任意的N，如果m为10，那么正确的概率为$1-2^{-10}$，几乎为1。

这个随机素数测试算法的正确性基于下面的定理。

定理11-1

(1) 如果对任意的$1<b<N$，使得$W(b)$成立，那么N为合数。

(2) 如果N为合数，那么$\frac{N-1}{2} \leqslant |\{b|1 \leqslant b<N,\ W(b)\text{成立}\}|$。

由上面的定理，如果N为合数，那么至少一半的$b_i(b_i<N)$使$W(b_i)$成立。如果存在一个b_i使得$W(b_i)$成立，那么N一定为合数。如果对于m个b_i，$W(b_i)$都不成立，那么根据定理11-1，N为合数的概率为$\left(\frac{1}{2}\right)^m$。因此，$N$为素数的概率大于$1-2^{-m}$。

11.4 模式匹配的随机算法

在本节中，将介绍随机模式匹配算法（randomized pattern matching algorithm）。该算法用来解决下述问题：给定长度为n的模式串X和长度为$m(m \geqslant n)$的文本串Y，找出作为Y的连续子串X的第一次出现的位置。不失一般性，假定X和Y都是二进制串。

例如，令$X = 01001$，$Y = 101\underline{0100}111$。可见$X$确实出现在$Y$中，如下划线所示。

令模式X和Y分别为

$X = x_1x_2\cdots x_n$，$x_i \in \{0, 1\}$

和 $Y = y_1y_2\cdots y_m$，$y_i \in \{0, 1\}$。

令$Y(1) = y_1y_2\cdots y_n$，$Y(2) = y_2y_3\cdots y_{n+1}$等等。一般地，令$Y(i) = y_iy_{i+1}\cdots y_{i+n-1}$，那么对于某个$i$，如果$X = Y(i)$，那么匹配出现。

存在另一种检验X和$Y(i)$的方法。令串s的二进制值为$B(s)$，那么

$B(X) = x_12^{n-1} + x_22^{n-2} + \cdots + x_n$，及

$B(Y_i) = y_i2^{n-1} + y_{i+1}2^{n-2} + \cdots + x_{i+n-1}$，$1 \leqslant i \leqslant m-n+1$，因此，只需检查$B(X)$是否等于$B(Y_i)$。

麻烦在于当n很大时，$B(X)$与$B(Y_i)$的计算将变得很困难。因此，将提供一种随机算法来解决这个问题。因为是随机算法，所以有可能造成错误。

该方法是计算$B(X)$除以素数P的余数。令整数u和v的余数为$(u)_v$。显然，当$(B(X))_p \neq (B(Y_i))_p$时，$X \neq Y_i$；但反之不然。例如，考虑$X = 10110$，$Y = 10011$，那么

$B(X) = 2^4 + 2^2 + 2^1 = 16 + 4 + 2 = 22$

$B(Y) = 2^4 + 2^1 + 2^0 = 16 + 2 + 1 = 19$

令$P = 3$，那么，$(B(X))_p = (22)_3 = 1$，$(B(Y))_p = (19)_3 = 1$。尽管$(B(X))_p = (B(Y))_p$，但是仍不能断定$X = Y(i)$。

该随机算法由以下思想组成：

(1) 选择k个素数p_1，p_2，$\cdots$，p_k。

(2) 如果对任意的j，有$(B(X))p_j \neq (B(Y_i))p_j$，那么$X \neq Y(i)$。

对于$j = 1, 2, \cdots, k$，如果$(B(X))p_j = (B(Y_i))\,p_j$，那么推断$X = Y(i)$。

我们将说明，如果推论是$X \neq Y(i)$，那么是绝对正确的；另一方面，如果推论为$X = Y(i)$，那么有可能做出错误的结论。

这种方法的优点在于$(B(X))_p$和$(B(Y_i))_p$计算简便，实际上无需计算$B(X)$与$B(Y_i)$。回忆

$B(X) = x_12^{n-1} + x_22^{n-2} + \cdots + x_n$

易知

$(B(X_i))_p = ((((x_1 \cdot 2)_p + x_2)_p \cdot 2)_p + x_3)_p \cdot 2 + \cdots$

类似地，

$(B(Y(i)))_p = ((((y_i \cdot 2)_p + y_{i+1})_p \cdots 2)_p + y_{i+2})_p \cdot 2 + \cdots$

采用这种机制，无需担心$B(X)$或$B(Y(i))$是否大数。现举例说明该点。

令$X = 10110$

$x_1 = 1$

$x_2 = 0$

$x_3 = 1$

$x_4 = 1$

$x_5 = 0$

令p为3，

$(x_1 \cdot 2)_p = (1 \cdot 2)_3 = (2)_3 = 2$

$(2 + x_2)_p = (2 + 0)_3 = (2)_3 = 2$

$(2 \cdot 2)_p = (4)_3 = 1$

$(1 + x_3)_p = (1 + 1)_3 = (2)_3 = 2$

$(2 \cdot 2)_p = (4)_3 = 1$

$(1 + x_4)_p = (1 + 1)_3 = (2)_3 = 2$

$(2 \cdot 2)_p = (4)_3 = 1$

$(1 + x_5)_p = (1 + 0)_3 = (1)_3 = 1$

即$(B(X))_p = (B(X))_3 = 1$

可见通过这种计算，所有涉及的数都相对较小。

接下来，给出模式匹配的随机算法。

算法11-4 模式匹配的随机算法

输入：模式串$X = x_1x_2\cdots x_n$，文本串$Y = y_1y_2\cdots y_m$及参数k。

输出：(1) 否，Y中无连续子串匹配X。

(2) 是，$Y(i) = y_iy_{i+1}\cdots y_{i+n-1}$匹配$X$。

如果答案为“否”，那么无错。

如果答案为“是”，那么可能以某个概率出错。

步骤1. 从$\{1, 2, \cdots, nt^2\}$，$t = m + n - 1$中随机选出k个素数$p_1, p_2, \cdots, p_k$。

步骤2. $i = 1$。

步骤3. $j = 1$。

步骤4. 如果$(B(X))p_j \neq (B(Y_i))p_j$，那么转至步骤5。

如果$j = k$，返回$Y(i)$作为答案

$j = j + 1$

转到步骤3。

步骤5. 如果$i = t$，返回“否，Y中无连续子串匹配X”

$i = i + 1$。

转到步骤3。

现在，对该随机算法作理论分析。本质上，可以证明当k足够大时，得出错误结论的概率是非常小的。

也许会问：当产生一个错误结果时，发生了什么？当得出错误的结果时，有以下条件：

(1) $B(X) \neq B(Y_i)$

(2) 对$j = 1, 2, \cdots, k$，均有$(B(X))p_j = (B(Y_i))p_j$

由于上述条件，可以得出

$$B(X) = a_jp_j + c_j \tag{11-1}$$

$$B(Y_i) = b_jp_j + c_j \tag{11-2}$$

因此，

$$B(X) - B(Y_i) = (a_j - b_j)p_j \tag{11-3}$$

等式（11-3）表明对所有的p_j，$B(X) - B(Y_i) \neq 0$且p_j能整除$B(X) - B(Y_i)$时，会产生错误结论。

现在提出一个实质性的问题：有多少个素数能够整除$|B(X) - B(Y(i))|$？

为了回答这个问题，给出一个n比特的模式X和m比特的文本，令Q表示下面的积

$$\prod_{i}^{k=m-n+1} |B(X) - B(Y(i))|$$

其中p_j能整除$|B(X) - B(Y(i))|$。

注意p_j也能整除Q。

Q的值小于$2^{m(m-n+1)}$。有了这个值，就能确定出错的概率，需要下面的声明：

如果$u \geqslant 29$且$a \geqslant 2^u$，那么与a相异的素数约数个数小于$\pi(u)$，其中$\pi(u)$表示小于u的素数个数。

我们并不详细证明上述声明是如何得出的。为了利用上述声明，注意Q小于$2^{n(m-n)}-2^{nt}$，因此，如果$nt \geqslant 29$，那么Q的相异素数约数个数少于$\pi(nt)$。

在算法中，p_j是从$\{1, 2, \cdots, nt^2\}$中选出的一个素数。因此，p_j能整除Q的概率小于$\pi(nt)/\pi(nt^2)$。这意味着p_j给出错误结论的概率小于$\pi(nt)/\pi(nt^2)$。因此，当选出k个素数时，那么做出错误结论的概率取决于这些数据，将少于$(\pi(nt)/\pi(nt^2))^k$。

讨论的总结如下：

对于该随机模式匹配算法，如果选出k个不同的素数，当$nt \geqslant 29$时，那么做出错误结论的概率小于$(\pi(nt)/\pi(nt^2))^k$。

下一个问题是：如何估算$\pi(x)$？有如下估算公式：

对所有的$u \geqslant 17$，

$$\frac{u}{\ln u} \leqslant \pi(u) \leqslant 1.255\,06\,\frac{u}{\ln u}$$

一般地，即使k不是很大，做出错误结论的概论也相当得小。

假定$nt \geqslant 29$，则有

$$\begin{aligned}\frac{\pi(nt)}{\pi(nt^2)} &\leqslant 1.255\,06\,\frac{nt}{\ln nt}\cdot\frac{\ln(nt^2)}{nt^2}\\ &= \frac{1.255\,06}{t}\left(\frac{\ln(nt^2)}{\ln(nt)}\right)\\ &= \frac{1.255\,06}{t}\left(\frac{\ln(nt)+\ln(t)}{\ln(nt)}\right)\\ &= \frac{1.255\,06}{t}\left(1+\frac{\ln(t)}{\ln(nt)}\right)\end{aligned}$$

例如，令$n = 10$和$m = 100$，那么$t = 91$。

$$\begin{aligned}\frac{\pi(nt)}{\pi(nt^2)} &\leqslant \frac{1.255\,06}{t}\left(1+\frac{\ln t}{\ln(nt)}\right)\\ &= \frac{1.255\,06}{91}\left(1+\frac{\ln(91)}{\ln(910)}\right)\\ &= 0.013\,792\cdot\left(1+\frac{4.510\,9}{6.813\,4}\right)\\ &= 0.013\,792\cdot(1+0.662\,1)\\ &= 0.013\,792\cdot 1.662\,1\\ &= 0.022\,9\end{aligned}$$

假定$k = 4$，那么得出错误结论的概率为

$(0.022\,9)^4 \approx 2.75*10^{-7}$

这是非常非常小的。

11.5 交互证明的随机算法

考虑以下问题。在冷战时期，一名英国的MI5（Military Intelligence Unit 5，即第五军事情报组织，参考P.Wright的《Spy Catcher》（情报捕手））特工想与一名已被她的最高政府在KGB培训数年的间谍联系。

令该MI5特工为B，而在KGB中培训过的间谍为A。现在的问题是：B怎样知道A就是真正的A，而不是其他的KGB的冒充者。一种简单的方法是问A的娘家姓。但麻烦的是：如果A回答正确，那么一些KGB工作人员在下一次能很容易地冒充A。因此，B必须让A做些相当困难的事，这些事的难度是普通人难以胜任的。例如，他可以让A判断一个布尔表达式的可满足性。假定A足够聪明，知道怎么漂亮地解决该NP完全问题。因此，当他每次从B处得到一个布尔表达式，他解答出该问题。如果表达式是可满足的，他给B发送一个赋值；而表达式不可满足时，仅发送“NO”。B不需要很聪明，他只需知道可满足性的定义，至少可以检查赋值是否满足表达式，如果A每次都正确地解决了可满足性问题，那么B会很高兴，并确信A就是真正的A。

但是，一名窃听者可能逐步发现B总是发送布尔表达式给A，并且，如果该公式是满足的，那么A发送可满足的答案给B。这个窃听者开始研究这种机械的定理证明方法，他迟早能假扮A。

可以说A和B都太容易造成泄密。下面的方法可能更好些：B发送给A一些信息后，A先对数据作一些计算再给B发送回结果数据。这样，B需作相应的计算后验证A的结果。如果符合，那么他可确信A是正确的人。通过这种方法，窃听者仍能截走一些数据，但对他而言，很难明白接下来的事。

在本节中，将说明B要求A解决二次非剩余问题（quadratic non-residue problem），并且有趣地看到数据在来回传送过程中，没有泄露太多的信息。当然，由于是随机算法，它包含一定程度的允许误差。

令x与y为两个正整数，$0<y<x$，使得$gcd(x, y) = 1$。定义$Z_x = \{z|0<z<x, gcd(z, x) = 1\}$。如果对某些$z \in Z_x$有$y = z^2 \bmod x$，称$y$为模$x$的二次剩余；否则称为模$x$的二次非剩余。进一步定义两个集合：

$QR = \{(x, y) \mid y$是模x的二次剩余$\}$

$QNR < \{(x, y) \mid y$是模x的二次非剩余$\}$

例如，设$y = 4$且$x = 13$。显然，存在z，也就是2，满足$z \in Z_x$和$y = z^2 \bmod x$（容易验证：$4 = 2^2 \bmod 13$）。因此，4为模13的二次剩余，同理能证明8为模x的二次非剩余。

现在，B与A怎样进行通信使得A能为B解决二次非剩余问题而不泄露太多信息？

他们执行过程如下：

(1) A与B都知道x的值，且保守该秘密，B知道y的值。

(2) B的行为：

(a) 抛硬币，获得m比特：$b_1, b_2, \cdots, b_m$，其中m为x的二进制表示的长度。

(b) 对所有的i，找出$z_1, z_2, \cdots, z_m$，$0<z_i<x$，使得$gcd(z_i, x) = 1$。

(c) 按如下方法由$b_1, b_2, \cdots, b_m, z_1, z_2, \cdots, z_m$计算出$w_1, w_2, \cdots, w_m$的值：

如果$b_i = 0$，那么$w_i = z_i^2 \bmod x$。

如果$b_i = 1$，那么$w_i = (z_i^2 \cdot y) \bmod x$。

(d) 将$w_1, w_2, \cdots, w_m$发送给A。

(3) A的行为：

(a) 从B处接收$w_1, w_2, \cdots, w_m$。

(b) 按如下方法，置c_1，c_2，…，c_m如下：

如果$(x, w_i) \in QR$，那么$c_i = 0$。

如果$(x, w_i) \in QNR$，那么$c_i = 1$。

(c) 将c_1，c_2，…，c_m发送给B。

(4) B的行为：

(a) 从A处接收c_1，c_2，…，c_m。

(b) 对于所有的i，如果$b_i = c_i$，那么返回“是，y为模x的二次非剩余”；否则返回“否，y为模x的二次剩余”。

可以证明，如果y为模x的二次非剩余，那么B以概率$1-2^{-|x|}$接收它；如果y为模x的二次剩余，那么B以概率$2^{-|x|}$接收它。

B能够通过重复该过程减少出错。注意A必须是足够聪明的人，能够解决二次非剩余问题。因此，A是一个“证明者”，而B是一个“验证者”。

现在给出一些例子。

假定$(x, y) = (13, 8)$，$|x| = 4$。

B的行为：

(a) 假定b_1，b_2，b_3，$b_4 = 1, 0, 1, 0$。

(b) 假定z_1，z_2，z_3，$z_4 = 9, 4, 7, 10$，它们相对于$x = 13$都与x互质。

(c) w_1，w_2，…，w_m计算如下：

$b_1 = 1$，$w_1 = (z_1^2 \cdot y) \bmod x = (9^2 \cdot 8) \bmod 13 = 648 \bmod 13 = 11$

$b_2 = 0$，$w_2 = (z_2^2) \bmod x = 4^2 \bmod 13 = 16 \bmod 13 = 3$

$b_3 = 1$，$w_3 = (z_3^2 \cdot y) \bmod x = (7^2 \cdot 8) \bmod 13 = 392 \bmod 13 = 2$

$b_4 = 0$，$w_4 = (z_4^2) \bmod x = 10^2 \bmod 13 = 100 \bmod 13 = 9$

因此，$(w_1, w_2, w_3, w_4) = (11, 3, 2, 9)$。

(d) 将$(11, 3, 2, 9)$发送给A。

A的行为：

(a) 从B处接收$(11, 3, 2, 9)$。

(b) $(13, 11) \in QNR$，$c_1 = 1$。

$(13, 3) \in QR$，$c_2 = 0$。

$(13, 2) \in QNR$，$c_3 = 1$。

$(13, 9) \in QR$，$c_4 = 0$。

(c) 将$(c_1, c_2, c_3, c_4) = (1, 0, 1, 0)$发送给$B$。

B的行为：

对所有的i，均有$b_i = c_i$，B接收到的是8为模13的二次非剩余，此为真。

现在看另一个例子。

$(x, y) = (13, 4)$，$|x| = 4$。

B的行为：

(a) 假定b_1，b_2，b_3，$b_4 = 1, 0, 1, 0$。

(b) 假定z_1，z_2，z_3，$z_4 = 9, 4, 7, 10$。

(c) 易得$(w_1, w_2, w_3, w_4) = (12, 3, 1, 9)$。

(d) 将$(12, 3, 1, 9)$发送给A。

A的行为：

(a) 从B处接收$(12, 3, 1, 9)$。

(b) (13，12)∈QR，$c_1=0$。

(13，3)∈QR，$c_2=0$。

(13，1)∈QR，$c_3=0$。

(13，9)∈QR，$c_4=0$。

(c) 将(0，0，0，0)发送给B。

B的行为：

由于对所有的i，并非$b_i=c_i$，B接收到的事实是4为模13的二次剩余。

该随机算法的理论基础是数论，超出了本书的讨论范围。

11.6 最小生成树的随机线性时间算法

在第3章中，介绍了基于贪心法的两种最小生成树算法。这些树算法其中之一是Kruskal算法，其时间复杂度为$O(n^2 \log n)$，另一种算法为Prim算法，该算法复杂版本的时间复杂度为$O(n+m\alpha(m, n))$，其中$n(m)$为该图中的顶点（边）的个数，而$\alpha(m, n)$是阿克曼函数的逆函数。在本节中，将介绍一种随机的最小生成树算法，其期望的时间复杂度是$O(n+m)$。

该算法建立在1926年由Boruvka所提出的所谓Boruvka步骤之上。下面的引理反映了Boruvka步骤背后的思想。

引理1：令V_1和V_2为非空顶点集，其中$V_1 \cup V_2 = V$且$V_1 \cap V_2 = \phi$，边(v, u)为最小权重的边，该边的一个顶点在V_1中，另一个顶点在V_2中，这样，(v, u)必包含于图G的最小生成树中。

引理1可以用另一种方式描述如下：在图G中，对任一结点u，在所有与点u邻接的边中，如果边(v, u)为最小权重的边，那么，(v, u)一定是图G中的最小生成树中的一条边。很容易证明引理1，参见图11-7，对结点c，在所有邻接到结点c的边中，边(c, e)权重最小，所以，边(c, e)必包含于图G的最小生成树中。同样地，很容易证明边(f, g)也包含于其中。

现在，运用引理1选择出图11-7中所有包含在最小生成树中的边，最终不同的连通分支如图11-8所示，在图11-8中，所有虚线为连通分支边。

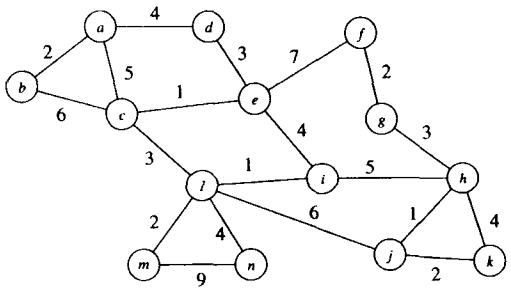


图11-7 一个图

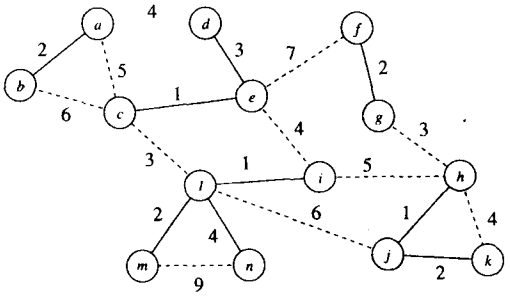


图11-8 在Boruvka步骤中边的选择

我们可将每个连通分支中的结点压缩成一个结点，现有五个结点，如图11-9所示。消除多余重复边和回路后，结果如图11-10所示。

由于得到的图由多于一个的结点组成，所以，再次应用引理1，结果如图11-11所示，被选择的边为(a, d)，(c, l)和(g, h)。

将每个连通分支缩成一个结点后，得到两个结点，如图11-12所示。

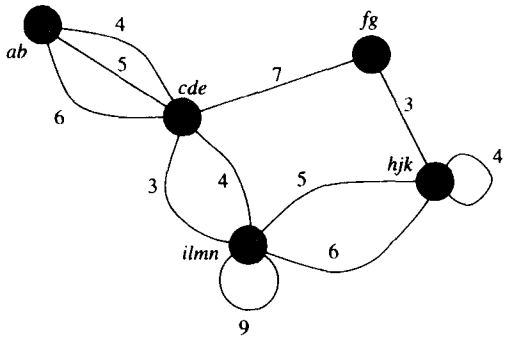


图11-9 结点的构造

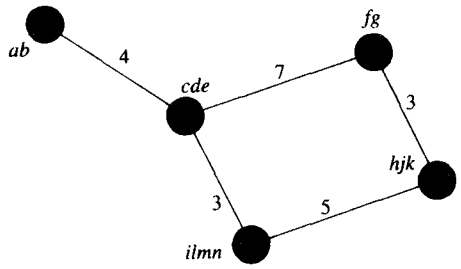


图11-10 第一次运用Boruvka步骤的结果

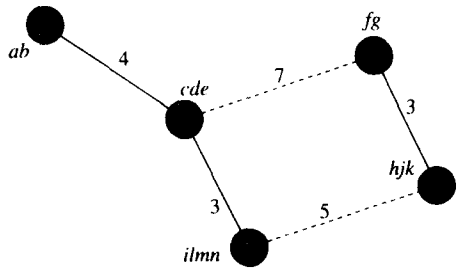


图11-11 边的第二次选择

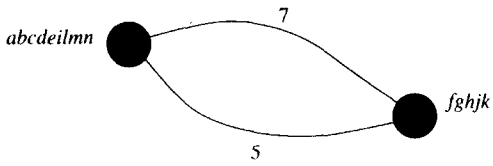


图11-12 边的第二次构造

同样，消除多余重复边，并选择边(i，h)后，能将所有结点构造成一结点，过程结束。所有被选择构成最小生成树的边如图11-13所示。

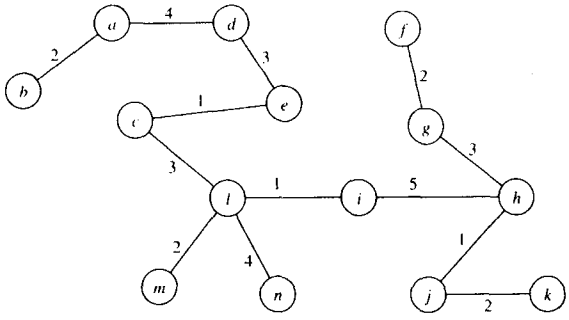


图11-13 运用Boruvka步骤获得的最小生成树

寻找最小生成树的Boruvka算法是递归运用Boruvka步骤直到最终生成的图成为单独的一点。令Boruvka步骤的输入为图$G(V, E)$，输出为图$G'(V', E')$，Boruvka步骤描述如下。

Boruvka步骤

1. 对每个结点u，找出与它关联的最小权重边(u，v)，找出由此标记的边确定的所有连通分支。
2. 将由标记边确定下来的每个连通分支压缩成一个顶点，令产生的图为$G'(V', E')$，并消除重复边和回路。

一次Boruvka步骤的时间复杂度为$O(n + m)$，其中$|V| = n$，$|E| = m$。由于G是连通的，$m>n$，所以$O(n + m) = O(m)$。因为由标记边所确定下来的每个连通分支至少包含两个点，在每次Boruvka步骤执行后，剩余的边比原始的边至少要减少一半。因此，Boruvka步骤执行的总次

数为$O(\log n)$，这样，Boruvka算法的总时间复杂度为$O(m \log n)$。

为了更有效地运用Boruvka步骤，必须使用一个新概念，参见图11-14。在图11-14b中，图G_s是图11-14a中图G的子图。G_s的最小生成森林F如图11-14c所示。在图11-14d中，最小生成森林F隐藏在原始图G中。所有不在F中的边用虚线标记。考虑边(e, f)，边(e, f)的权重为7。因此。在森林F中，在e至f之间有一条路径，即$(e, d)\rightarrow(d, g)\rightarrow(g, f)$。边$(e, f)$的权重大于该路径中的边的最大权重。由下面的一条引理可知，边(e, f)不是G的最小生成树中的边。在介绍引理之前，先定义一个称为F重的（F-heavy）术语。

令$w(x, y)$表示图G中边(x, y)的权重，G_s为图G的一个子图，F为图G_s的最小生成森林，$w_F(x, y)$为在F中连接x与y的路径中边的最大权重。如果在F中x与y不相连，那么令$w_F(x, y) = \infty$。相对于F，如果$w(x, y)>w_F(x, y)(w(x, y)\leqslant w_F(x, y))$，那么称边$(x, y)$是$F$重的（$F$轻的）。

参见图11-14d，可知边(e, f)、(a, d)和(c, g)都是关于F的F重的。在定义了这个新概念后，有如下引理，对使用Boruvka步骤是非常重要的。

引理2：令G_s为图$G(V, E)$的子图，F为图G_s的最小生成森林，图G中相对于F的F重边不可能是图G的最小生成树的边。

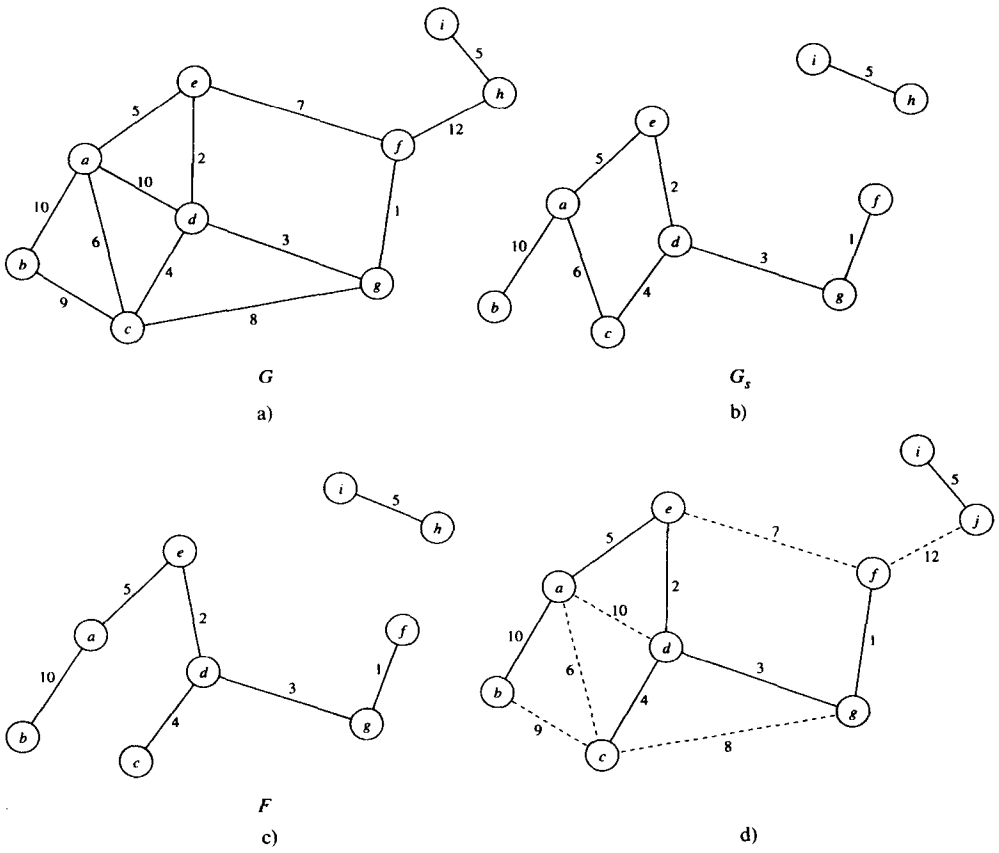


图11-14 F重边

我们不证明该引理。运用该引理，可知边(e, f)、(a, d)和(c, g)不是最小生成树的边。

为了完全运用Boruvka步骤，需要引入另一个引理，即引理3。

引理3：令H为从G得到的子图，它是通过以概率p独立地包含每条边，F为H的最小生成森林。在G中，F轻（F-light）边的期望数最多为n/p，其中n为G中顶点的个数。

随机的最小生成树算法描述如下。

算法11-5 随机的最小生成树算法

输入：带权的连通图G。

输出：G的最小生成树。

步骤1. 连续使用Boruvka步骤三次，令最终的图为$G_1(V_1, E_1)$。如果图G_1只含一个结点，那么返回步骤1中标记的边集合并退出。

步骤2. 以概率1/2独立地从G_1中选出每条边构造子图H。递归地对H运用该算法得出H的最小生成森林F。通过删除G_1中相对于F的所有F重边后得到图$G_2(V_2, E_2)$。

步骤3. 对G_2递归地应用该算法。

接下来分析算法11-5的时间复杂度。

令$T(|V|, |E|)$表示该算法对图$G(V, E)$的期望运行时间。

步骤1的每次执行花费$O(|V| + |E|)$时间。步骤1执行后，得到$|V_1| \leqslant |V|/8$及$|E_1| < |E|$。对于步骤2，计算H需要的时间为$O(|V_1| + |E_1|) = O(|V| + |E|)$，计算$F$需要的时间为$T(|V_1|, |E_1|/2) = T(|V|/8, |E|/2)$，删除所有$F$重边所需的时间为$O(|V_1| + |E_1|) = O(|V| + |E|)$。运用引理3可知，$|E_2|$的期望值最多为$2|V_2| \leqslant |V|/4$。因此，步骤3的执行时间为$T(|V_2|, |E_2|) = T(|V|/8, |V|/4)$。令$|V| = n$，$|E| = m$，有如下的递归关系式：

$$T(n, m) \leqslant T(n/8, m/2) + T(n/8, n/4) + c(n + m)$$

对某个常数c成立。可以证明

$$T(n, m) \leqslant 2c \cdot (n + m)$$

我们鼓励读者将(2)代入(1)来检验该解，因此，算法的期望运行时间为$O(n + m)$。

11.7 注释与参考

随机算法的概述可在文献Maffioli(1986)中找到。文献Gill(1987)和Kurtz(1987)也分别纵览了随机算法。值得提出的是，对于不同的人随机算法意味着不同的东西。它有时表明一个算法在平均情况下是好的。也就是说，算法因不同的数据集而不同。在本书中，我们强调随机算法是一种在运行过程中采用抛掷硬币的随机的算法。换言之，对于相同的数据输入，因为过程的随机性，其结果可能非常不同。

通过随机算法解决最近点对问题是由Rabin(1976)提出的。最新的成果可参阅文献Clarkson(1988)。运用随机算法检测素数，参阅文献Solovay and Strassen(1977)和Rabin(1980)。素数判定问题由文献Agrawal, Kayal and Saxena(2004)证明是多项式问题。模式匹配的随机算法出现在文献Karp and Rabin(1987)中。交互式验证的随机算法由文献Goldwasser, Micali and Rackoff(1988)发现的。文献Galil, Haber and Yung(1989)提出了对该方法的进一步改善。随机的最小生成树算法在文献Karger, Klein and Tarjan(1995)中找出。Boruvka步骤则是在文献Boruvka (1926) 所发现，而删除F重边方法可在文献Dixon, Rauch and Tarjan(1992)中找到。

随机算法在文献Brassard and Bratley(1988)中广泛地讨论。

11.8 进一步的阅读资料

随机算法可分为两类：顺序的和并行的。虽然本书中只讲解了随机顺序算法，但我们也推荐一些随机并行算法。

对于随机顺序算法，我们推荐：Agarwal and Sharir(1996)；Anderson and Woll(1997)；

Chazelle, Edelsbrunner, Guibas, Sharir and Snoeyink (1993); Cheriyan and Harerup(1995); Clarkson(1987); Clarkson(1988); d'Amore and Liberatore(1994); Dyer and Frieze(1989); Goldwasser and Micali(1984); Kannan, Mount and Tayur(1995); Karger and Stein(1996); Karp(1986); Karp, Motwani and Raghavan(1988); Matousek(1991); Matousek(1995); Megiddo and Zemel(1986); Mulmuley, Vazirani and Vazirani(1987); Raghavan and Thompson(1987); Teia(1993); Ting and Yao(1994); Wenger(1997); Wu and Tang(1992); Yao(1991)以及Zemel(1987)。

对于随机并行算法，我们推荐Alon, Babai and Itai(1986); Anderson(1987); Luby(1986) and Spirakis(1988)。

大量最新出版的论文包括：Aiello, Rajagopalan and Venkatesan(1998); Albers(2002); Alberts and Henzinger(1995); Arora and Brinkman(2002); Bartal and Grove(2000); Chen and Hwang(2003); Deng and Mahajan(1991); Epstein; Noga, Seiden, Sgall and Woeginger(1999); Froda(2000); Har-Peled(2000); Kleffe and Borodovsky (1992); Klein and Subramanian(1997); Leonardi, Spaccamela, Presciutti and Ros(2001); Meacham(1981) and Sgall(1996)。

习题

11.1 编程实现解决最近点对问题的随机算法，并测试你的算法。

11.2 利用随机的素数测试算法确定下列数是否素数：13, 15, 17。

11.3 对于下列两字符串使用随机的模式匹配算法。

$X = 0101$

$Y = 0010111$

11.4 利用11.5节中介绍的算法判断5是否13的二次剩余。举出一个得出错误结论的例子。

11.5 阅读文献Brassard and Bratley(1988)中的8.5节和8.6节。

第12章 在线算法

在前面所介绍的各章，算法设计基于这样的假设，即在算法执行之前，整个数据的情况都是可知的。也就是，问题是与完整的数据信息一起解决的。然而，事实上并不完全是这样的。考虑磁盘调度问题（disk scheduling problem），对算法来说，磁盘的服务请求是完全不知的，请求一个接一个地到来。出现在操作系统设计中的分页问题（paging problem）也是一个在线问题（on-line problem）。在执行程序之前无法确定哪个页面将被访问。如果每个数据都是在线到来，在线算法就必须采取行动处理到达的每个数据。由于没有完整的信息可用，在此时看似正确的处理，但在之后可能就会产生错误。因此，从这个意义上在线算法（on-line algorithm）都是近似算法（approximation algorithm），不能保证产生最优解。以在线最小生成树问题（on-line minimum spanning tree problem）为例，在这种情况下，首先必须放弃“最小”这个词，因为生成树不会是最小的。因此，把这个问题称为在线小生成树问题（on-line small spanning tree problem）。在线算法处理该问题的过程如下：每次当一个数据项到达时，将其与离它最近的邻点相连接。假设有如图12-1所示的6个点，数据按指定的顺序到达，那么在线算法产生一个如图12-2所示的生成树。显然，这棵生成树不是最优的。依据全部数据信息的最优生成树如图12-3所示。

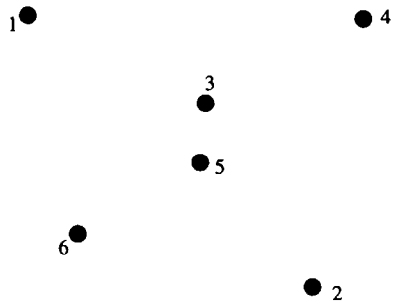


图12-1 说明在线生成树算法的数据集

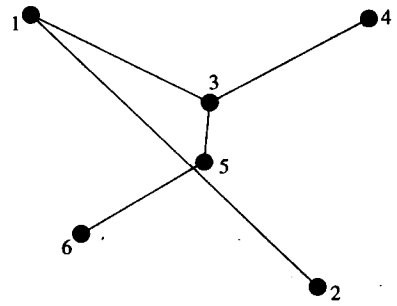


图12-2 在线小生成树算法产生的生成树

由于任何在线算法一定是近似算法，所以对它性能的度量自然是通过将它与最优离线算法（optimal off-line algorithm）所得结果做对比来得到的。对同一个数据集，令C_{onl}（C_{off}）表示运行在线（最优离线）算法的代价。如果$C_{onl} \leqslant c \cdot C_{off} + b$，其中$b$是一个常量，那么称该在线算法的性能比（performance ratio）是c，算法是c可竞争的（c-competitive）。如果c不能再小，就说这个在线算法是最优的。由于在线算法的设计必须与分析联系在一起，所以在线算法的设计决不是一件容易的事情。在本章中，将介绍几种在线算法及对它们的分析。

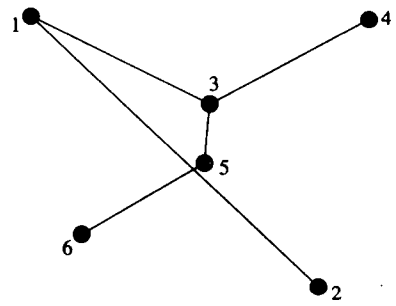


图12-3 依据图12-1中数据产生的最小生成树

12.1 用贪心法解决在线欧几里得生成树问题

在欧几里得最小生成树问题中，已知平面上的一个点集，要构造出这些点的一棵最小生成树。对于该问题的在线方案，各个点逐个出现，无论在任何时候出现某一点，都必须采取行动将这个点与已经构造好的生成树相连接。此外，所采取的行为是不可逆的。显然，由在线算法所构造的树一定是近似树。解决这种欧几里得生成树问题的贪心法可描述如下：令v_1，v_2，…，v_{k-1}是将出现的点，T是目前构造的生成树，将v_k与v_1，v_2，…，v_{k-1}间的最短边添加到T中。参见图12-4所示的点集，通过贪心法构造的生成树如图12-5所示。

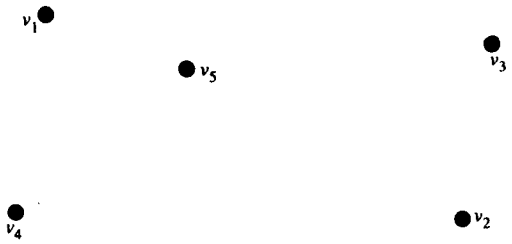


图12-4 五个点的集合

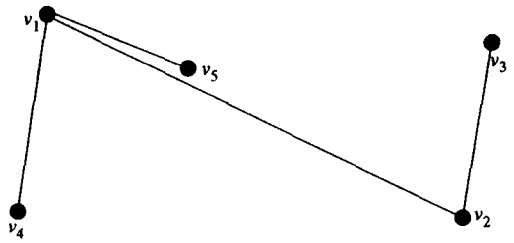


图12-5 用贪心法构造的生成树

接下来分析基于贪心法的在线算法性能。令S表示n个点的集合，l表示对集合S所构造的最小生成树的长度，T_{onl}表示由在线算法所构造的生成树。我们将证明该算法是$O(\log n)$可竞争的。

实质上，要证明在T_{onl}中，第k长边的长度最大是$2l/k$（$1 \leqslant k \leqslant n-1$）。另一种说法是在$T_{onl}$中最多有$k-1$条边，其长度大于$2l/k$。已知产生$T_{onl}$的顶点序列，令$S_k$表示加入$T_{onl}$中使边的长度大于$2l/k$的点集。这样，原来的陈述变成：$S_k$的基数小于$k$。为证明此结论，注意根据该贪心在线算法的定义，在$S_k$中每对顶点间的距离一定大于$2l/k$。因此，在$S_k$上最优旅行商问题回路的长度必须大于

$$|S_k|\frac{2l}{k}$$

因为由相同点集所构造的最优旅行商问题的回路长度最多是同一点集最小生成树长度的两倍，所以，在S_k上的最小生成树的长度大于

$$|S_k|\frac{l}{k}$$

由于S_k上的最小生成树的长度小于由S上的最小生成树的长度，可得

$$|S_k|\frac{l}{k} < l$$

或者等价地，

$$|S_k| < k$$

这意味着S_k的基数小于k，从而证明原来的陈述：*在T_{onl}中，第k长边的长度最大是$2l/k$*。这样，T_{onl}的总长度最大为

$$\sum_{k=1}^{n-1}\frac{2l}{k} = 2l\sum_{k=1}^{n-1}\frac{l}{k} = l \times (\log n)$$

这说明贪心在线生成树算法是$O(\log n)$可竞争的。这个算法是最优的吗？当然不是。可以证明竞争比的下界是$\Omega(\log n/\log\log n)$。这将在下面讨论。对于在线生成树问题将找出一个输

入σ，并证明不存在在线算法A，使得由算法A和输入σ构造的生成树长度与以同样输入σ构造的最小生成树长度之比是$\Omega(\log n/\log\log n)$。接下来，要描述该输入$\sigma$。输入$\sigma$的所有点都在一个网格内，假设$x$是整数且$x\geqslant 2$，令$x^{2x}=n$，那么$x\geqslant 1/2(\log n/\log\log n)$且$n\geqslant 16$。该输入由$x+1$层的点所组成，其中每一层又是由长度为$n=x^{2x}$的水平线均匀隔开的点集。在第$i$层（$0\leqslant i\leqslant x$）点的坐标记为$(a_i, b_i)$，其中$a_i=x^{2x-2i}$。当$i=0$时，$b_i=0$；当$i\neq 0$时，且$0\leqslant j\leqslant n/a_i$，$b_i=\sum_{k=1}^{i}a_k$。从而，$a_0=x^{2x}(=n)$，$a_x=1$。对于所有的$i$，第$i$层和第$i+1$层之间顶点间的距离是$c_i=b_{i+1}-b_i=a_{i+1}$。在第$i$层上的顶点数为$\frac{n}{a_i}+1$。输入顶点的总数为：

$$
\begin{aligned}
\sum_{i=0}^{x}\left(\frac{n}{a_i}+1\right) &= \sum_{i=0}^{x}\left(\frac{x^{2x}}{x^{2x-2i}}+1\right)\\
&= \sum_{i=0}^{x}(x^{2i}+1)\\
&= \frac{x^{2x+2}-1}{x^2-1}+x+1\\
&= \frac{nx^2-1}{x^2-1}+x+1\\
&= n+\frac{n-1}{x^2-1}+x+1\\
&= O(n)
\end{aligned}
$$

对于$x=2$和$n=16$的一个输入例子σ如图12-6所示。

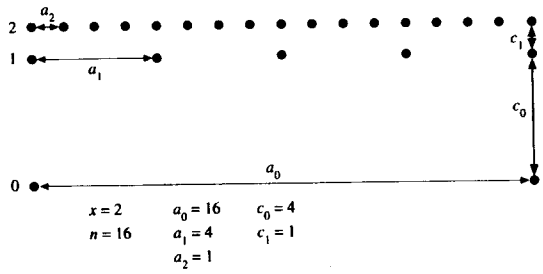


图12-6 一个输入σ例子

该输入的最小生成树的构造如下：

(1) 第x层的每个点与它的邻点水平相连接。

(2) 其他每个点与邻点垂直相连接。（图12-7所示是图12-6的顶点形成的最小生成树。）

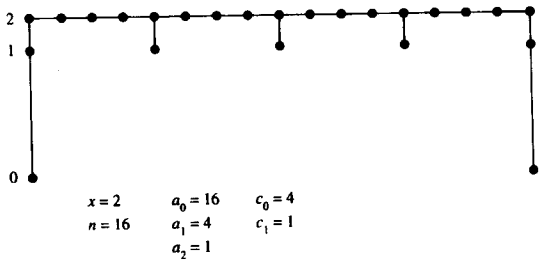


图12-7 在图12-6中的顶点形成的最小生成树

该最小生成树的总长度为

$$
\begin{aligned}
n+\sum_{i=0}^{x-1} c_i\left(\frac{n}{a_i}+1\right) &= n+\sum_{i=0}^{x-1} a_{i+1}\left(\frac{n}{a_i}+1\right) \\
&= n+\sum_{i=0}^{x-1} a_{i+1}+n\sum_{i=0}^{x-1}\frac{a_{i+1}}{a_i} \\
&= n+\sum_{i=0}^{x-1}\frac{n}{x^{2i+2}}+n\sum_{i=0}^{x-1}\frac{1}{x^2} \\
&\leqslant n+n\sum_{i=0}^{x-1}\frac{1}{x^2}+n\sum_{i=0}^{x-1}\frac{1}{x^2} \\
&= n+2n\frac{x}{x^2} \\
&\leqslant 3n
\end{aligned}
$$

假设σ中的点是在线地从第0层到第x层逐层地给出。令T_{i-1}表示在对获得第i层点之前，在线算法的图。在第i层上已安排有$\left(\frac{n}{a_i}+1\right) \geqslant \frac{n}{a_i}$个点。注意，在第$i$层两个相邻点之间的距离是$a_i$。对于每个点，任意边$T_i - T_{i-1}$的长度至少是$a_i$，即对于第$i$层，由在线算法增加的生成树的总长度至少是

$$\frac{n}{a_i}\cdot a_i = n$$

所以，由在线算法构造的生成树的总长度至少是$n \cdot x$。

由$C_{off} \leqslant 3n$及$C_{onl} \geqslant nx$，可得

$$\frac{C_{onl}}{C_{off}} \geqslant \frac{1}{3n}\cdot nx = \frac{1}{3}x > \frac{\log n}{6\log\log n}$$

这意味着该问题的竞争比下界是$\Omega(\log n/\log\log n)$，由于我们介绍的算法是$O(\log n)$可竞争的，所以它不是最优的。

12.2 在线k 服务员问题及解决定义在平面树上该问题的贪心算法

现在考虑k服务员问题（k-sever problem）。已知有n个顶点的图，每条边分配一个正的边长。令在k（$k<n$）个顶点处分配k名服务员。已知对服务员的请求序列，必须判定如何调动服务员才能满足请求。对一个请求的服务代价是满足该请求在服务员移动的总距离。参见如图12-8，有三名服务员s_1、s_2和s_3，分别位于a、e和g，假如在顶点i有一个请求，由于e靠近i，那么可能的一种移动方法是把位于顶点e处的s_2移到顶点i处。

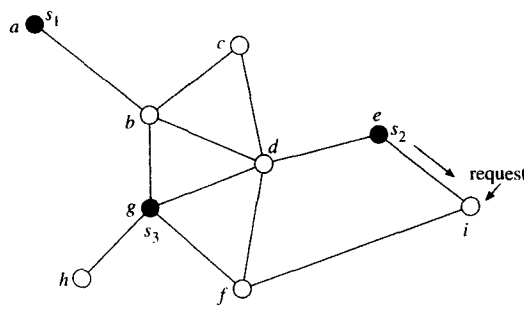


图12-8 k服务员问题实例

基于贪心策略的解决该k服务员问题的在线算法是移动离请求所在位置最近的服务员。不过这种头脑简单的贪心在线算法有一个瑕疵，如图12-9所示。

假设有一名服务员位于顶点d，第一个请求r_1来自顶点c，贪心算法会把位于顶点d的服务员移到c，此时第二个请求到达，而遗憾的是它来自d，那么，再把服务员从c移到d。如图12-9所示，如果请求不断在c和d之间转换，那么服务员就不停地在c和d之间移动。实际上，正如下面所示，如果把位于f的服务员逐渐地移到d，服务员最终位于d一段时间，那么不停地在c和d之间移动服务员的现象就会避免。解决该问题改进的贪心算法移动许多所谓的活动服务员（active severs）到当前请求所在顶点位置。我们将k服务员问题限制在一棵平面树T中。由于T是一棵平面树，所以T中各边的长度满足三角不等式。令x和y是T中的两个点，它们之间的距离指T中从x到y的简单路径长度，记为$|x, y|$。令s_i和d_i分别表示服务员i及他目前所处的位置。区间$(x, y]$表示T中从x到y的路径，不包括x。如果在区间$(d_i, x]$中再没有其他服务员，那么相对于x处的请求，称服务员s_i是活动的。改进的贪心在线k服务员算法执行如下：当x点有请求时，将当前相对于x的所有活动服务员以相同速度不断移向x，直到某一名服务员到达x。如果在移动期间，某一名活动服务员变为不活动的话，就中止。图12-10所示是相对于一个请求，上述改进的贪心法。

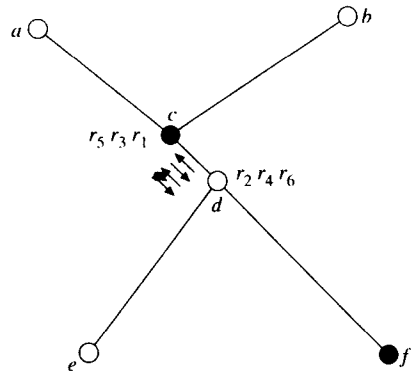


图12-9 一种贪心在线k服务员算法的最坏情况

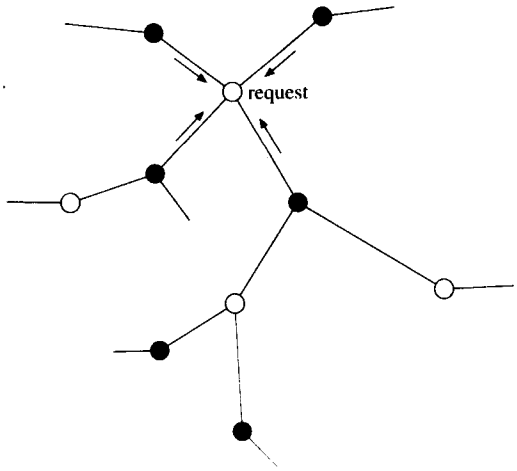


图12-10 改进的贪心在线k服务员算法

现在，对该算法进行性能分析。首先定义全信息的k服务员在线算法。该算法是绝对在线的。也就是在每次请求之后，算法将移动一名服务员到最近的请求发起顶点。然而，由于算法是全信息的，它拥有完全的请求序列的信息。所以，它的行为将与普通的不具有全信息的在线算法大不相同。事实上，它是最优算法，因为它能产生最优结果。

参见图12-11，在a和d两点分别有两名服务员s_1和s_2，假定请求序列为：

(1) 请求r_1在b点。

(2) 请求r_2在e点。

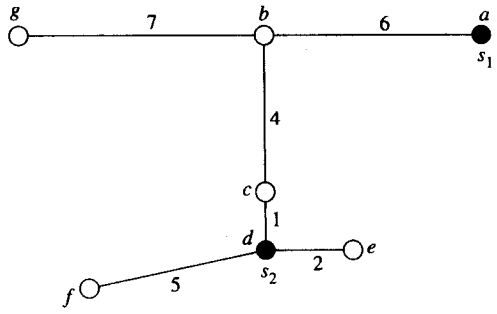


图12-11 完全信息的k服务员在线算法的实例

对于不包含全“信息”的贪心在线算法，服务员的移动如下：

(1) 把s_2从d移动到b，代价为5。

(2) 把s_2从b移动到e，代价为7。

总代价为12。

对于全信息的k服务员在线算法，服务员的移动如下：

(1) 把s_1从a移动到b，代价为6。

(2) 把s_2从d移动到e，代价为2。

总代价仅为8。由于全信息在线算法知道请求r_2位于e，所以在开始它移动s_1而不是s_2。

我们的分析是对比改进的贪心在线k服务员算法与全信息在线k服务员算法的移动代价。想象一个游戏，在每次请求到达时，知道全信息的对手做一次移动。然后，我们也做一次移动。需要注意在我们移动所有相对于该请求的所有活动服务员时，我们的对手只移动一名服务员。正如在分摊分析中所做的，定义一个势能函数，将我们及对手所有位置上的k名服务员映射为一个非负实数。

令$\tilde{\psi}_i$表示对手（全信息算法）在响应第i个请求后及我们（贪心法）在响应第i个请求之前势能函数的值，ψ_i表示贪心法在响应第i个请求后且响应第$(i+1)$个请求之前的势能函数的值，ψ_0表示势能函数的初值。这些术语最好在图12-12中加以说明。

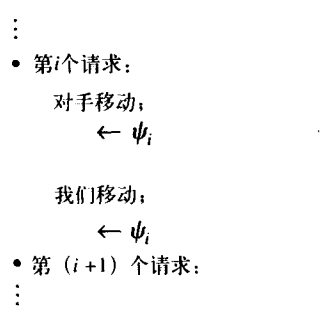


图12-12 响应请求的势能函数值

令我们的贪心法与对手的全信息算法响应第i个请求的代价分别用O_i和A_i表示，O和A分别表示所有的请求后贪心法与全信息法的总代价，证明下列的不等式。

(1) 对于某一α，$\tilde{\psi}_i - \psi_{i-1} \leqslant \alpha A_i$，$1 \leqslant i \leqslant n$。

(2) 对于某一β，$\tilde{\psi}_i - \psi_i \geqslant \beta O_i$，$1 \leqslant i \leqslant n$。

上述等式可扩展为：

$$\tilde{\psi}_1 - \psi_0 \leqslant \alpha A_1$$

$$\psi_1 - \tilde{\psi}_1 \leqslant -\beta O_1$$

$$\tilde{\psi}_2 - \psi_1 \leqslant \alpha A_2$$

$$\psi_2 - \tilde{\psi}_2 \leqslant -\beta O_2$$

$$\vdots$$

$$\tilde{\psi}_n - \psi_{n-1} \leqslant \alpha A_n$$

$$\psi_n - \tilde{\psi}_n \leqslant -\beta O_n$$

将各式相加，可以得到

$$\psi_n - \psi_0 \leqslant \alpha(A_1 + A_2 + \cdots + A_n) - \beta(O_1 + O_2 + \cdots + O_n)$$

$$\beta O \leqslant \alpha A + \psi_0 - \psi_n$$

由于$\psi_n \geqslant 0$，

$$\beta O \leqslant \alpha A + \psi_0$$

$$O \leqslant \frac{\alpha}{\beta} A + \frac{1}{\beta}\psi_0$$

现在定义势能函数。在任意时间的实例中，令贪心在线k服务员算法的k名服务员位于b_1，b_2，…，b_k，全信息算法的k名服务员位于a_1，a_2，…，a_k。由v_1，v_2，…，v_k及v_1'，v_2'，…，v_k'定义一个二分图，其中$v_i(v_i')$分别表示$b_i(a_i)$，边(v_i, v_j')的权值为$|b_i, a_j|$。在该二分图中，可以进行最小带权匹配（minimum weighted matching），用$M_{\min}$表示，那么势能函数定义为

$$\psi = k|M_{\min}| + \sum_{i<j} |b_i, b_j|$$

考虑 $\tilde{\psi}_i - \psi_{i-1}$。由于只有在全信息算法做一次移动，才有一个$a_i$改变。因此，在$\psi$中，仅有可能的增加与$|M_{\min}|$的改变有关，它等于$A_i$，故得到

$$\tilde{\psi}_i - \psi_{i-1} \leqslant kA_i \tag{12-1}$$

现在考虑$\psi_i - \tilde{\psi}_i$。对于第i个请求，贪心算法有q名服务员移动，$q \leqslant k$，假设每名服务员移动的距离都为d。对$M_{\min}$来说，至少有一条匹配的边将减少到0，因为两个算法都必须满足第i个请求，故必有一对v_a和v_b间的距离为0，这意味着$M_{\min}$至少增加$-d+(q-1)d=(q-2)d$。

现在计算$\sum_{i<j}|b_i, b_j|$的增加。令d_p表示贪心算法一名活动服务员的位置，b_r表示使得d_p位于第i个请求与b_r之间一名服务员的位置，l_p表示b_r的数目。由于位于d_p的服务员每次移动d距离，故在d_p与l_p个b_r之间移动距离的和将增加$(l_p-1)d$，但对于另外的$(k-l_p)$名服务员，移动距离的和将会减少$(k-l_p)d$。因此，对于第i个请求，随着q名服务员的移动，增加的总距离至多为$\sum_{p=1}^{q}(l_p-1+l_p-k)d$。所以，势能函数的改变至多是

$$k(q-2)d + \sum_{p=1}^{q}(l_p-1+l_p-k)d = -qd \quad \left(\text{注意} \sum_{p=1}^{q} l_p = k\right)$$

即，得到

$$\psi_i - \tilde{\psi}_i \leqslant -qd$$

或者等价地，

$$\tilde{\psi}_i - \psi_i \geqslant O_i \tag{12-2}$$

合并式（12-1）和式（12-2），可以得到

$$O \leqslant kA + \psi_0$$

这正是想要得到的。因此，k服务员问题的在线算法是k可竞争的。

已证明在线k服务员算法是k可竞争的，那么我们自然要问：在线算法是最优算法吗？是否存在其他在线k服务员算法有更好的性能呢？例如，能否有$k/2$可竞争的在线k服务员算法？我们将证明这是不可能的。换句话说，在此介绍的在线k服务员算法确实是最优的。

首先注意到在线算法是定义在平面树上，边长满足三角不等式。此外，算法同时移动几名服务员，而不是一名。为了证明在线k服务员算法的最优性，定义一个称为“懒惰的在线k服务员算法”（lazy on-line k-server algorithm）的算法。如果处理每个请求仅移动一名服务员，那么该在线k服务员算法称为是懒惰的。

容易看到，如果一个非懒惰在线k服务员算法的执行基于一个满足三角不等式的图，那么在不增加执行代价的条件下，该在线算法可以转化为懒惰的在线k服务员算法。令D表示非懒惰的在线k服务员算法。为了满足某一请求，D需把服务员s从v移动到w。但是，由于D是非懒惰的，在本步骤之前，即使在v处没有请求，D也必须把s从u移动到v，那么把s从u移动到w的代价为$d(u, v)+d(v, w)$，由三角不等式可知，该代价大于等于$d(u, w)$。因此，可以去掉移动s从u到v这一步而不增加代价。

在下面的讨论中，假设在线k服务员算法是一个懒惰算法。

令D表示任何在线的k服务员算法，$G(V, E)$表示在G的k个不同的顶点有k个不同的服务员的图。v_1是V的一个子集，包括k名服务员初始时所在的k个顶点及G的另一个任意顶点x。定义请求序列为：

(1) $\sigma(1)$位于x处。

(2) 对于$i \geqslant 1$，$\sigma(i)$是在i时刻没有被D所覆盖的v_1中唯一的顶点。

参见图12-13。

令$k=3$，三名服务员的初始位置分别在a，c和f。令另一个选择的点是d，那么$v_1=\{a, c, f, d\}$。现在，第一个请求$\sigma(1)$一定在d处。如果D把s_2从c移动到d，将c空出，那么$\sigma(2)=c$。如果D接着把s_1从a移动到c，将a空出，那么$\sigma(3)=a$。对于这个请求序列，D在$\sigma(1)$，$\sigma(2)$，…，$\sigma(t)$的服务代价为

$$C_D(\sigma, t)=\sum_{i=1}^{t} d(\sigma(i+1), \sigma(i))$$

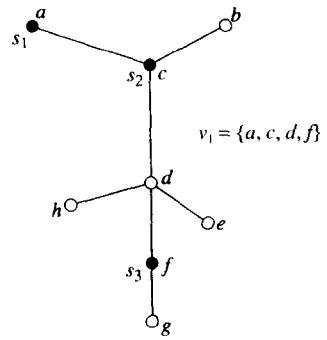


图12-13 3服务员问题的例子

现在定义一个比算法D有更多信息的在线k服务员算法，称为算法E。因为算法E设法知道了$\sigma(1)$，所以算法E具有更多的信息。令v_2是$\sigma(1)$，且是$v_1-\sigma(1)$的具有$k-1$个顶点的任一子集。例如，在上例中，$v_1=\{a, c, f, d\}$。由于$\sigma(1)=d$，v_2可以是$\{a, c, d\}$、$\{c, f, d\}$或$\{a, f, d\}$，考虑到该顶点子集v_2，可定义一个在线k服务员算法$E(v_2)$，它比算法D有更多的信息如下：

如果v_2包含$\sigma(1)$，那么什么也不做；否则，把在$\sigma(i-1)$处的服务员移动到$\sigma(i)$，更新v_2以反映该变化。初始时，服务员都占据v_2中的点。

注意在这个例子中，初始条件改变了，因为初始时服务员位于$\sigma(1)$所在位置。在这种情况下，$E(v_2)$在满足第一个请求时不必移动任何服务员，这就是为什么说$E(v_2)$比D具有更多信息。$E(v_2)$花费的代价绝不比D更多，所以可作为所有在线k服务员算法的下界。因此易知，对所有的$i>1$，当第i步开始时，v_2总包含着$\sigma(i-1)$。

再次通过举例说明此概念。考虑如图12-13所示的情况。假如$v_2=\{a, c, d\}$及

$\sigma(1)=d$

$\sigma(2)=e$

$\sigma(3)=f$

$\sigma(4)=a$

三名服务员s_1、s_2和s_3初始分别占据a、c和d，那么$E(v_2)$运行如下：

(1) 由于v_2中包含$\sigma(1)=d$，那么什么也不做。

(2) 由于v_2中不包含$\sigma(2)=e$，那么把位于$\sigma(1)=d$的服务员s_3移动到e，并令$v_2=\{a, c, e\}$。

(3) 由于v_2中不包含$\sigma(3)=f$，那么把位于$\sigma(2)=e$的服务员s_3移动到f，并令$v_2=\{a, c, f\}$。

(4) 由于在v_2中包含$\sigma(3)=a$，那么什么也不做。

由于v_1中包含$k+1$个顶点，所以共有$\binom{k}{k-1}=k$个不同的v_2，即有k个不同的$E(v_2)$算法。人们自然要问：哪个$E(v_2)$性能最好？不是找出最好的$E(v_2)$算法的代价，而是找出这k个$E(v_2)$算法的期望代价。最好的$E(v_2)$算法的代价将会比期望的代价小。

根据$E(v_2)$算法的定义，第1步的代价为0，在第$i+1(i \geqslant 1)$步，这些算法或者什么也不做而没有代价，或者从$\sigma(i)$到$\sigma(i-1)$移动一名服务员，代价是$d(\sigma(i), \sigma(i+1))$。在这$k$个算法中，它们中的$\binom{k-1}{k-1}=1$个，其包含$\sigma(i)$，但不包含$\sigma(i+1)$，将承担此代价，所有其他算法的代价

都为0。所以对于第$i+1$步，由所有这些算法所增加的代价是$d(\sigma(i),\ \sigma(i+1))$，那么包含$\sigma(t)$的这$k$个算法的总代价等于

$$\sum_{i=1}^{t} d(\sigma(i),\sigma(i-1))$$

因此，$E(v_2)$的期望代价为

$$\begin{aligned}E(v_2) &= \frac{1}{k}\sum_{i=2}^{t} d(\sigma(i),\sigma(i-1)) \\ &= \frac{1}{k}(d(\sigma(2),\sigma(1)) + d(\sigma(3),\sigma(2)) + \cdots + d(\sigma(t),\sigma(t-1))) \\ &= \frac{1}{k}\left(\sum_{i=1}^{t} d(\sigma(i+1),\sigma(i)) - d(\sigma(t+1),\sigma(t))\right) \\ &= \frac{1}{k}(C_D(\sigma,t) - d(\sigma(t+1),\sigma(t)))\end{aligned}$$

因为 $C_D(\sigma,t) = \sum_{i=1}^{t} d(\sigma(i+1),\sigma(i))$ 。

最好的$E(v_2)$算法代价一定小于期望的代价。令它为$E(v_2^1)$，其代价表示为 $C_{E(v_2^1)}$，这样，

$$C_{E(v_2)} \geqslant C_{E(v_2^1)}$$

这意味着

$$\frac{1}{k}C_D(\sigma,t) - \frac{1}{k}d(\sigma(t+1),\sigma(t)) \geqslant C_{E(v_2^1)}(\sigma,t)$$

即 $C_D(\sigma,t) \geqslant k \cdot C_{E(v_2^1)}(\sigma,t)$

或者等价地，

$$\frac{C_D(\sigma,t)}{C_{E(v_2^1)}(\sigma,t)} \geqslant k$$

上式说明对于任意非全信息的在线k服务员算法，必存在一个有更多信息的在线k服务员算法，其代价比非完全算法小k倍。这证明了定义在平面树上的在线贪心k服务员算法不是太好的。

12.3 基于平衡策略的在线穿越障碍算法

本节讨论穿越障碍问题（obstacle traversal problem）。设有一个方形障碍集，集合中所有方块的边平行于坐标轴，且每条边的长度小于或等于1。有一个开始点，表示为s和一个目标点，表示为t。穿越障碍问题是要避开障碍找到一条从s出发到达t的最短路径。图12-14说明一个典型的例子。

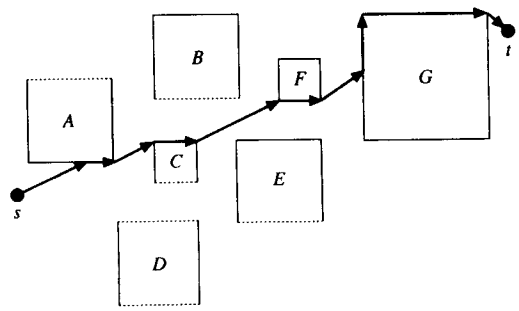


图12-14 穿越障碍问题的一个例子

对于该问题，这里所介绍的算法是在线算法。也就是，没有整个障碍的全景信息。这样，

获得最优解是不可能的。搜寻者从s开始利用许多启发式方法来指导，这些启发式方法形成了在线算法。在本节的剩余部分中，将描述这些规则。

首先注意到连接s与t的直线是s与t之间可能的最短几何距离，因此，将该距离表示为d，作为所得解的下界。后面将所得的解与d做比较。将证明当d很大时，使用在线算法搜寻者所遍历的距离不超过$3d/2$。

假设从s到t的直线与水平轴夹角$\psi \leqslant \frac{\pi}{4}$，否则它与竖轴夹角$\psi' \leqslant \frac{\pi}{4}$。显然，我们介绍的算法稍加改动后仍将应用。

令α表示从s到t的方向。接下来，将介绍针对不同情况的所有规则。首先，了解有多少种情况必须处理，参见图12-15所示。

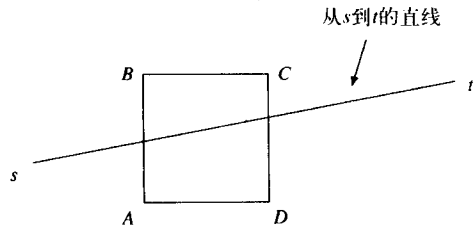


图12-15 一个障碍ABCD

至少有三种情况：

情况1：搜寻者从障碍物中间穿过。

情况2：搜寻者沿障碍物水平边，比如沿AD。

情况3：搜寻者沿障碍物竖直边，比如沿AB。

对情况1，得到规则1。

规则1：当搜寻者在障碍物中间穿过时，它沿着方向α前进，也就是可以看成没有障碍物地前进。

一旦有了规则1，假设有两个搜寻者，分别表示为P和Q。P是真正的搜寻者，他在碰到障碍物时要设法绕开它，Q是假想的搜寻者，他沿着从s到t的直线前进而忽略障碍物。我们将对P和Q前进的距离进行比较。

规则2：如果搜寻者与障碍物的水平边AD相交于点E，他会沿着ED前进，然后向上到点F，使得EF与$s-t$直线平行，如图12-16所示，随后它重新沿着α方向前进。

搜寻者P前进的距离是$|ED| + |DF|$，对于同样的时间间隔，假想的搜寻者Q前进的距离是$|EF|$，如图12-16所示。$(|ED| + |DF|)/|EF| = \cos\psi + \sin\psi \leqslant 3/2$。从而可知，真正搜寻者遇到障碍的水平边，沿方形的水平边前进的距离不大于假想搜寻者Q的3/2倍。

如果搜寻者P遇到竖直边，那么情况较为复杂，可通过C点沿α方向作一直线，交AB于G，如图12-17所示。

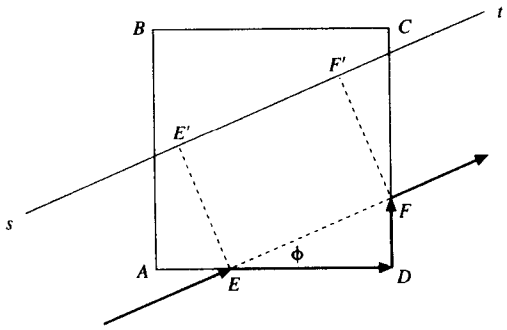


图12-16 与AD相交于E

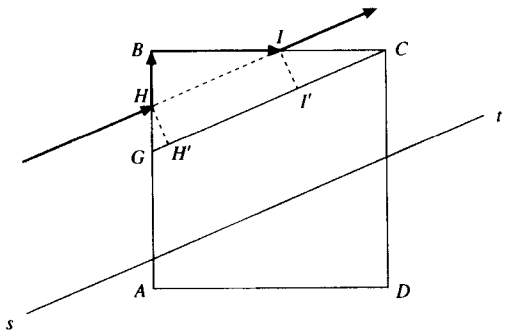


图12-17 与AB相交于H

这样得到规则3。

规则3：如果搜寻者P与障碍物的竖直边AB相交于BG段的H，那么他向上到B，再向右移动到I，使得HI与直线$s-t$平行，如图12-17所示。然后重新沿方向α前进。

易证明在此例中，真正搜寻者P前进的距离与假想搜寻者Q的比率不大于3/2。

如果搜寻者P与障碍物的竖直边AG段相交，那么规则也更为复杂。实际上，这些规则形成了平衡策略（balancing strategy）的核心。

规则4：如果搜寻者P与障碍物的竖直边AB的AG段相交，那么P既可以向上也可以向下。如果他向上移动，那么移动到B向右转直到角C，如图12-18a所示。如果它向下移动，那么移动到A向右转直到角D，如图12-18b所示。在遇到角后，重新沿方向α前进。

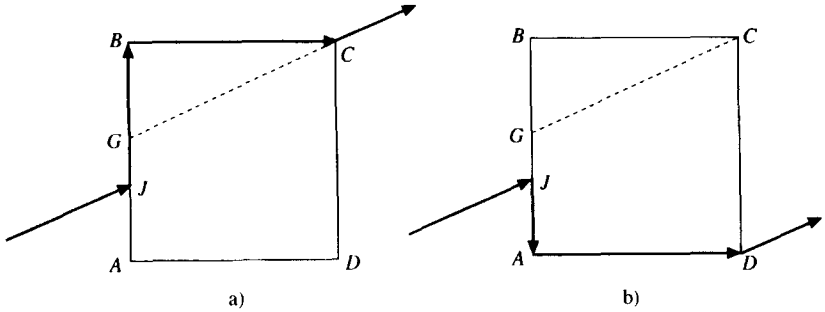


图12-18 交于边AB的两种情况

规则4说明，如果搜寻者交于AG段，那么它既可以向上也可以向下移动，现在的问题是：如何决定是向上还是向下移动呢？

如果P向上移动，那么他经过的距离为$\tau_1 = |JB|+|BC|$；如果他向下移动，那么距离为$\tau_2 = |JA|+|AD|$。而在这段时间里，与此对应的假想搜寻者Q向上或向下移动的距离分别记为π_1或π_2。距离π_1和π_2可将点J，C和D投影到$s-t$直线上，如图12-19所示。在图12-19a中，J位于直线$s-t$之下，在图12-19b中，J位于直线$s-t$之上。

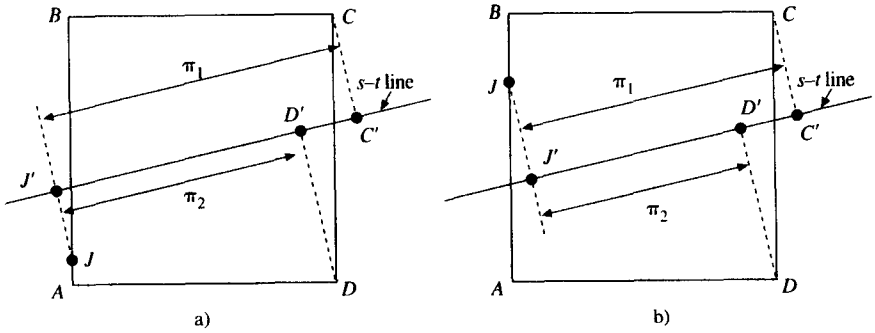


图12-19 对π_1和π_2的说明

如果τ_1/π_1和τ_2/π_2都不大于3/2，那么这将是好的。但并不总是这样的。参见图12-20中的情况，假设搜寻者向上移动，显而易见，τ_1/π_1几乎为2；同理，参见图12-21中的情况，假如搜寻者向下移动，τ_2/π_2也几乎为2。

在1978年已证明了τ_1/π_1和τ_2/π_2中至少有一个不大于3/2。注意到交点J或者位于$s-t$之上或者在其之下。显然，如果J位于$s-t$之上且$\tau_1/\pi_1 \leqslant 3/2$，那么向上移动；否则，$J$位于$s-t$之下且$\tau_2/\pi_2 \leqslant 2/3$，那么向下移动。但这会导致$P$不停地向上或向

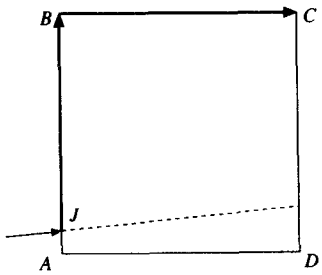


图12-20 τ_1/π_1有最坏值的情况

下移动的问题，由于将会离$s-t$越来越远，这未必是有益的。

令方形的边长为k，其中$k\leqslant 1$。可将段AG划分为长度都为$\frac{k}{\sqrt{d}}$的若干段，其中d为$s-t$间的距离。如果某段的最低点满足$\tau_1/\pi_1\leqslant 3/2$，那么它标记为向上；如果它的最低点满足$\tau_2/\pi_2\leqslant 3/2$，那么它标记为向下。一段既可标记为向上也可标记为向下。如果只标记为向上或向下，那么该段称为“纯净”（pure）段，否则称为“混合”（mixed）段。

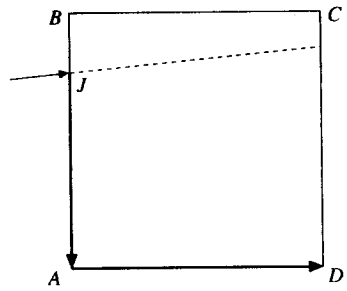


图12-21 τ_2/π_2有最坏值的情况

令J_i表示第i段的最低点，J_i'表示沿方向α的直线过点J_i与CD边的交点。若第i段为纯净段，如果它标记为上，那么$\rho_i=|DJ_i'|/|CJ_i'|$；如果标记为下，那么$\rho_i=|CJ_i'|/|DJ_i'|$。算法将为每一段保持一个参数，称为平衡（balance）。所有的平衡初值为0，后面将说明，平衡用来控制是向上还是向下移动。

规则5：假设搜寻者P交于属于第i段内部的点J。

情形1：点J位于直线$s-t$之上。查看第i段的标记，如果标记为向下，那么向下移动；否则，查看平衡参数$balance$。如果$balance\geqslant\rho_i k$，那么向下移动并从$balance$中减去$\rho_i k$；否则，$balance$加上k并向上移动。

情形2：点J位于直线$s-t$之下。查看第i段的标记，如果标记为向上，那么向上移动；否则，查看$balance$。如果$balance\geqslant\rho_i k$，那么向上移动并从$balance$中减去$\rho_i k$；否则，$balance$加上k并向下移动。

最后，得到下面的规则。

规则6：如果搜寻者P碰到与目标的横和纵坐标相等，那么它可直达目标。

以上介绍的是在线穿越障碍算法。现在对它的性能进行分析。首先，需要提醒的是搜寻者P与AD边相交或与AB边的段BG相交，那么P与Q前进的距离比率不大于3/2，因此仅对交于AG段的情况进行分析。

根据混合段的定义，无论P向上还是向下移动，τ_1/π_1或τ_2/π_2都不大于3/2。所以，只考虑纯净段的情况。以第i段为例，假设它的标记是向上，且P在直线$s-t$之上。标记为向下的情况与此相似，令$a_i(b_i)$表示P向上（向下）遇到第i段所在方形的边长之和，那么相对于第i段P移动的总距离为$a_i\tau_1+b_i\tau_2$，而相对于第i段Q移动的总距离为$a_i\pi_1+b_i\pi_2$，其中的τ_1，τ_2，π_1和π_2这些量都是相对于一个单位方形第i段的最低点。那么

$$\frac{a_i\tau_1+b_i\tau_2}{a_i\pi_1+b_i\pi_2}=\frac{\frac{a_i}{b_i}\tau_1+\tau_2}{\frac{a_i}{b_i}\pi_1+\pi_2}$$

现在的问题是：$\frac{a_i}{b_i}$是多大？注意a_i和b_i都是相对于用在算法中的平衡参数。总之，$\rho_i b_i$是从平衡参数$balance$中减去的量而a_i是加上的量。由于平衡参数$balance$不能是负数，可得$\rho_i b_i\leqslant a_i$。由于第i段标记为向上，根据定义可知$\tau_1/\pi_1\leqslant 3/2$。

接着讨论$\frac{\rho_i\tau_1+\tau_2}{\rho_i\pi_1+\pi_2}$的上界。

$$\frac{\rho_i\tau_1+\tau_2}{\rho_i\pi_1+\pi_2}=\frac{\frac{|DJ_i'|}{|CJ_i'|}\cdot\tau_1+\tau_2}{\frac{|DJ_i'|}{|CJ_i'|}\cdot\pi_1+\pi_2}$$
$$=\frac{|DJ_i'|\tau_1+|CJ_i'|\tau_2}{|DJ_i'|\pi_1+|CJ_i'|\pi_2}$$

假设有一个单位方形，那么上式的分子

$$|DJ_i'|\tau_1+|CJ_i'|\tau_2$$
$$=(|AJ_i|+\tan\psi)(2-|AJ_i|)+(1-|AJ_i|-\tan\psi)(1+|AJ_i|)$$
$$=\frac{1}{\cos\psi}((\sin\psi+\cos\psi)+2(\cos\psi-\sin\psi)|AJ_i|-2\cos\psi(|AJ_i|)^2)$$

令$|AJ_i|=y$，$\sin\psi=a$，$\cos\psi=b$，那么

$$|DJ_i'|\tau_1+|CJ_i'|\tau_2$$
$$=\frac{1}{b}((a+b)+2(b-a)y-aby^2)$$

上式的分母为

$$|DJ_i'|\pi_1+|CJ_i'|\pi_2$$
$$=(|AJ_i|+\tan\psi)(\sin\psi+\cos\psi-|AJ_i|\sin\psi)+(1-|AJ_i|-\tan\psi)(\cos\psi-|AJ_i|\sin\psi)$$
$$=\frac{1}{\cos\psi}(\sin^2\psi+\cos^2\psi)$$
$$=\frac{1}{\cos\psi}$$
$$=\frac{1}{b}$$

这样，

$$\frac{|DJ_i'|\tau_1+|CJ_i'|\tau_2}{|DJ_i'|\pi_1+|CJ_i'|\pi_2}=(a+b)+2(b-a)y-2by^2$$

令$f(y)=(a+b)+2(b-a)y-2by^2$，那么

$f'(y)=2(b-a)-4by$

当 $y=\frac{b-a}{2b}$ 时，$f(y)$取得最大值。

$$f\left(\frac{b-a}{2b}\right)=(a+b)+2(b-a)\frac{b-a}{2b}-2b\left(\frac{b-a}{2b}\right)^2$$
$$=(a+b)+\frac{(b-a)^2}{b}-\frac{(b-a)^2}{2b}$$
$$=(a+b)+\frac{(b-a)^2}{2b}$$
$$=\frac{2b^2+b^2+a^2}{2b}$$

由于$a^2+b^2=\sin^2\psi+\cos^2\psi=1$，可以得到

$$f\left(\frac{b-a}{b}\right)=\frac{2b^2+1}{2b}$$

因为$b=\cos\psi$，$\psi\leqslant\pi/4$，可以得到

$$f\left(\frac{b-a}{b}\right)=\frac{2b^2+1}{2b}\leqslant\frac{3}{2}$$

这样，

$$\frac{\sigma_i\tau_1+\tau_2}{\sigma_i\pi_1+\pi_2}\leqslant\frac{3}{2}$$

易知上述不等式对所有边长为k（$\leqslant 1$）的方形障碍物也成立，因此可得

$$\frac{\rho_i\tau_1+\tau_2}{\rho_i\pi_1+\pi_2}\leqslant\frac{3}{2}$$

$$2\rho_i\tau_1+2\tau_2\leqslant 3\rho_i\pi_1+3\pi_2$$

$$(3\pi_1-2\tau_1)\rho_i\geqslant 2\tau_2-3\pi_2$$

$$(3\pi_1-2\tau_1)\frac{a_i}{b_i}\geqslant 2\tau_2-3\pi_2 \qquad (\text{根据}\rho_i b_i\leqslant a_i)$$

$$\frac{\frac{a_i}{b_i}\tau_1+\tau_2}{\frac{a_i}{b_i}\pi_1+\pi_2}\leqslant\frac{3}{2} \qquad \left(\text{根据}\ \frac{\pi_1}{\tau_1}\leqslant\frac{3}{2}\right)$$

假设搜寻者P与某个障碍物第i段的最低点相遇，如果他位于障碍物的第i段内的某点，每次这种情况出现时，都会有$O(1/\sqrt{d})$偏差。注意，d为直线$s-t$的长度，且每个方形的大小限制于某个常数$c(<1)$，那么最坏情况下会遇到$O(d)$个方形。因此，总偏差为$O(\sqrt{d})$。

接下来，在运用规则6到达t的时候，需要知道搜寻者P到直线$s-t$之间的距离。假设P位于直线$s-t$之上且交于第i段。很明显，如果第i段的标记为向下，那么它就越接近直线$s-t$。考虑第i段的标记为上的情况，那么P要向上。根据定义，为平衡第i段而增加的次数不超过ρ_i+1，在直线$s-t$之上的竖直距离为$|CJ_i'|\times(\rho_i+1)\leqslant 1$，其中$|CJ_i'|$为$P$与第$i$段相遇时，离开直线$s-t$的竖直距离。由于每个障碍物分为$\sqrt{d}$段，在应用规则6之后，搜寻者$P$与$s-t$直线间的距离最多为$\sqrt{d}$。

基于上述讨论可知，P与Q移动的距离之比不会超过

$$\frac{3}{2}+O\left(\frac{1}{\sqrt{d}}\right)$$

如果d足够大，那么比率不会超过3/2。这样，可以得到P移动的总距离不会超过Q移动距离的3/2。因此，在线算法是3/2可竞争的。

在上述讨论中，已证明在线穿越障碍算法的竞争比是3/2。接下来将证明该算法是最优的。假设障碍物为矩形，令r表示矩形的长宽之比。我们将证明解决穿越障碍问题的任何在线算法的竞争比小于$(r/2+1)$是不可能的。在上面的例子中，假设障碍物为方形，$r=1$。这样，竞争比不可能小于3/2，故该在线算法是最优的。

为找出竞争比的下界，要对障碍物有特殊的安排。图12-22说明了障碍物的特殊布局。令n是一个大于1的整数，如图12-22所示，s和t的坐标分别为(0，0)和(2n，0)。注意在障碍物之

间，有无限小的缝隙以隔离障碍物及供搜寻者在障碍物之间通过。显然，搜寻者既可水平移动也可竖直移动。

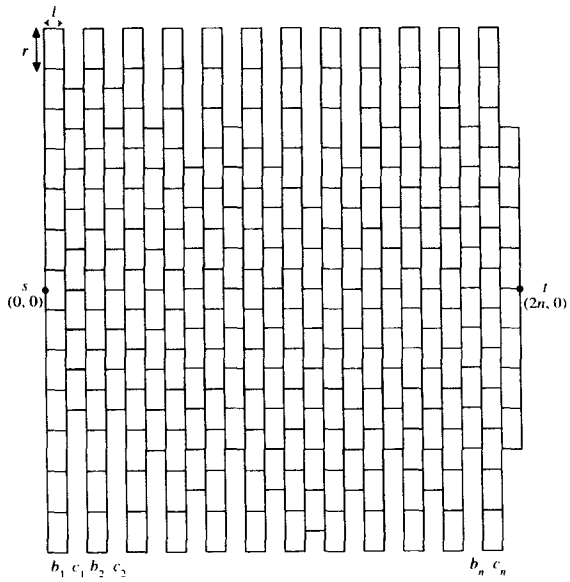


图12-22 障碍物的特殊布局

障碍物以纵队形式排列，有两种纵队b和c。每个b纵队有无穷多个障碍物组成，而c纵队只包含8个障碍物，每个b纵队与c纵队间隔排列，第i列$b(c)$纵队记为$b_i(c_i)$。

对于$1 \leqslant i \leqslant n$，根据搜寻者如何与$b_i$相遇的位置来安排第$c_i$纵队上障碍物的位置。令搜寻者$P$与第$b_i$纵队中相遇的障碍物为$\alpha_i$，$p_i$是$P$与第$b_i$纵队的障碍物$\alpha_i$相遇的点，那么在$c_i$纵队中的八个障碍物安排成$\alpha_i$位于它们的中间。图12-23所示为障碍物的一种可能排列，其中的粗线表示搜寻者P经过的轨迹。

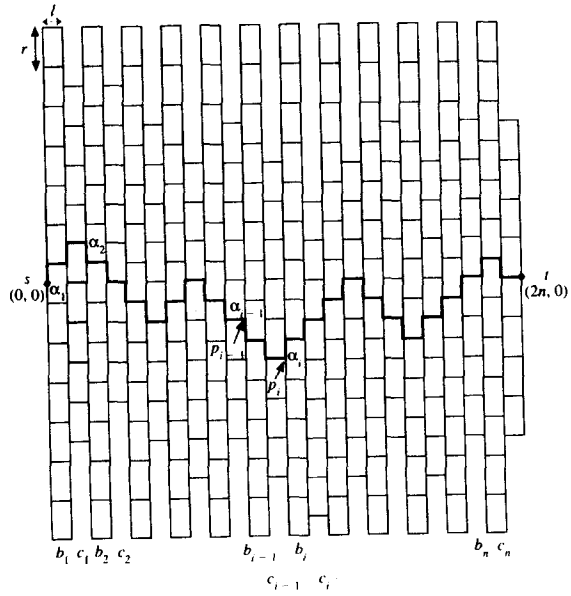


图12-23 图12-22布局中P的移动路线

显然，从p_{i-1}到p_i的最短路径长度至少为$r+2$。

因此，搜寻者P遍历的总距离至少为$(r+2)n$。

对于任意$0 \leqslant m \leqslant \sqrt{n}$，定义水平线$L_m$为$L_m$：$y=(8m+1/2)r$。

注意，对于任意的i和j，水平线L_m不与b_i中的任意障碍物相交，而可与c_j中的某一障碍物相交。由于在任何两条直线L_m和L_m'间的距离至少为$8r$，故它们不可能相交于同一纵队c_i上的障碍物。因此，L_0，L_1，…，$L_{\lceil\sqrt{n}\rceil}$最多交于c_1，c_2，…，c_n中的n个障碍物。由于有$\sqrt{n}$条直线并相交n个障碍物，那么平均每条线与$n/\sqrt{n}=\sqrt{n}$个障碍物相交，即至少存在一个整数$m \leqslant \sqrt{n}$，使得L_m与c_1，c_2，…，c_n中的$\sqrt{n}$个障碍物相交。对任意可行解，水平距离至少是$2n$。由于每个障碍物的长度是r，每次遇到一个障碍物，都遍历距离cr(c为一常数)，故构建可行的路径长度至多为

$2n+cr\sqrt{n}$对某个c

最终，可得解决穿越障碍问题的任意在线算法的竞争比至少为

$$\frac{r+n}{2n+nc\sqrt{n}}$$

如果n很大，那么竞争比至少变为$r/2+1$。

如果$r=1$，那么竞争比为3/2。

12.4 用补偿策略求解在线二分匹配问题

这本节中，讨论二分匹配问题（bipartite matching problem）。给定一个将顶点集二分为R和B的二分带权图（bipartite weighted graph）G，每个集合的基数为n。二分匹配M是边E的子集，在这个集合中没有两条边同时邻接到一个单独顶点上，且每条边都邻接R和B。匹配的代价指在这次匹配中边的权值之和。最小二分匹配问题是找出最小代价的二分匹配。在本节中，只研究最小二分匹配问题。

二分匹配问题的在线算法描述如下：集合R中的顶点预先已知，而集合B中的顶点则是依次出现。当B中的第i个顶点出现时，它必须与R中尚未匹配的顶点匹配，且此决策随后不能改变。目标是保持在线匹配代价低，对于数据也提出一个特殊的约束：所有边的权值都满足三角不等式。

接下来，首先证明从最小二分匹配算法中导出的代价下界，然后证明所有最小二分在线匹配算法的代价都小于最优离线匹配算法代价的（$2n-1$）倍。

令b_1，b_2，…，b_n为集合B的n个顶点，r_i（$1 \leqslant i \leqslant n$）表示当$b_i$出现时与$b_i$匹配的顶点，使用图12-24来说明二分图。

如图12-24所示，二分图具有如下的特征：

(1) 对于所有的i，边(b_1，r_i)的权值为1。

(2) 对于$i=2$，3，…，n，如果$j<i$，那么边(b_i，r_j)的权值为0。

(3) 对于$i=2$，3，…，n，如果$j \geqslant i$，那么边(b_i，r_j)的权值为2。

如果b_1，b_2，…，b_n是依次出现，那么任意在线算法总的匹配代价为$1+2(n-1)=2n-1$。

二分图的最优离线匹配过程如下：

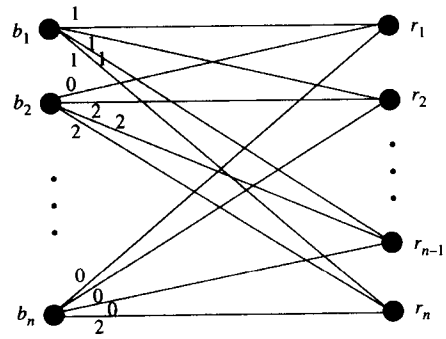


图12-24 证明在线最小二分匹配算法下界的二分图

r_1与b_2相匹配，代价为0；

r_2与b_3相匹配，代价为0；

⋮

r_n与b_1相匹配，代价为1。

因此，离线算法的最优代价为1，这意味着没有在线二分匹配算法获得小于$2n-1$的竞争比。

现在说明在线二分匹配算法。在线匹配算法基于离线匹配算法，下面举例说明。

参见图12-25所示的R集，所有的点都没有标记。两点之间边的权值为这两点间的几何距离。

现在，图12-26中所示为点b_1的出现，任何最优离线匹配会从R中找到离b_1最近的点。不失一般性，设该点为r_1，在线匹配算法也如最优离线算法那样将b_1与r_1相匹配。

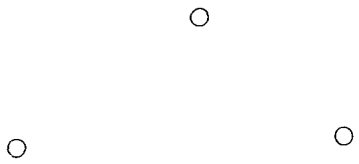


图12-25 R集

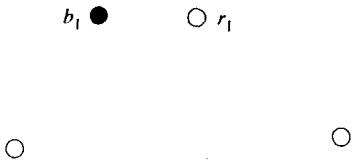


图12-26 b_1出现

图12-27中所示的是b_2出现。任何最优离线算法将b_2与r_1相匹配，而将b_1与R中一个新点匹配。不失一般性，令该点为r_2。问题是：在线匹配算法会执行哪一匹配呢？

由于在线匹配算法已将b_1与r_1匹配，不能改变这个事实。为了补偿，将b_2与新的顶点r_2相匹配。

最后，如图12-28所示b_3出现，最优离线算法将b_1与r_1匹配，b_2与R中新增加的顶点r_3匹配，b_3与r_2匹配。

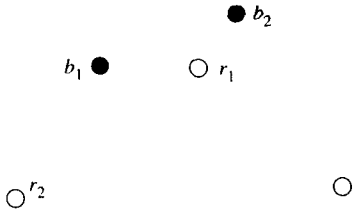


图12-27 b_2出现

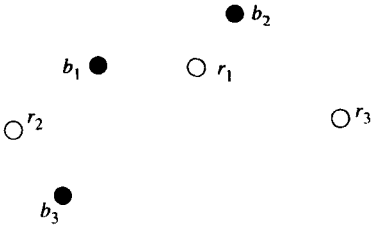
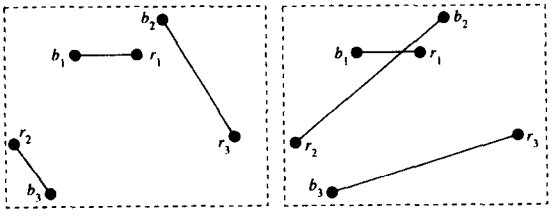


图12-28 b_3出现

在线二分匹配算法注意到r_3是新增加的顶点，所以将b_3与r_3匹配。

在图12-29中，将最优离线匹配与在线算法进行了比较。



最优离线匹配　　在线匹配

图12-29 在线与离线匹配的比较

现在正式地定义在线算法。假设在b_i出现之前，b_1，b_2，…，b_{i-1}已经依次出现，不失一般性，通过在线匹配M_{i-1}分别与r_1，r_2，…，r_{i-1}相匹配。当b_i出现时，考虑匹配集M_i'，满足下列条件：

(1) M_i'是$\{b_1, b_2, \cdots, b_i\}$与$r_1$，$r_2$，…，$r_i$之间的最优二分匹配集。

(2) 在$\{b_1, b_2, \cdots, b_i\}$与$r_1$，$r_2$，…，$r_i$之间所有的最优二分匹配集中，$|M_i' - M'_{i-1}|$是最小的。

令R_i表示在M_i'中R的顶点集，可证明R_i仅比R_{i-1}增加一个顶点。不失一般性，假设r_i是b_i出现后增加的顶点，也就是，$R_i = \{r_1, r_2, \cdots, r_{i-1}, r_i\}$。由于在$M_{i-1}$中，对于$j = 1, 2, \cdots, i-1$，$r_j$已经与$b_j$相匹配，为了补偿，在$M_i$中将$r_i$与$b_i$相匹配。

初始时，$M_1 = M_1'$。

现在讨论在线算法的一个有趣特性。在线算法产生一个匹配序列M_1'，M_2'，…，M_n'。现在考虑$M_i' \oplus M'_{i+1}$，其中的每条边是M_i'和M'_{i+1}其中之一的边。例如，图12-30所示的集合。

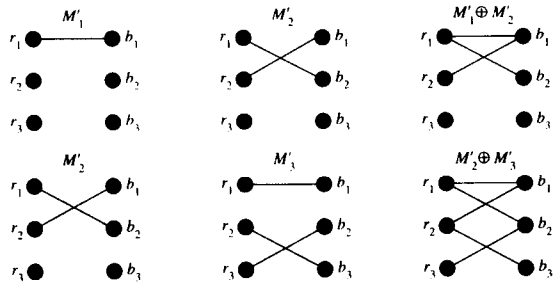


图12-30 $M_i' \oplus M_{i+1}'$

令E_i和E_j是两个边集，$E_i \oplus E_j$的一条交替路径（alternating path）（环路）是在E_i和E_j中交替出现边的一条简单路径（环路）。边在E_i和E_j中交替出现。最大交替路径（maximal alternating）（环路）是指在这个交替路径（环路）中没有子路径（子环路）。对于$M_i' \oplus M'_{i+1}$，仅有一条以b_2为终点的最大交替路径，即

$r_2 \to b_1 \to r_1 \to b_2$

类似地，在$M_2' \oplus M_3'$中仅有一条以b_3为终点的最大交替路径，即

$r_3 \to b_2 \to r_1 \to b_1 \to r_2 \to b_3$

现在要证明：对于$i = 2, \cdots, n$，在$M'_{i-1} \oplus M_i'$中，只有一条最大交替路径且这条交替路径以b_i为终点。

令H是$M'_{i-1} \oplus M_i'$，首先注意下面：

(1) 在H中b_i的度为1，因为b_i是新出现的顶点，且它必须与R中的某一顶点相匹配。

(2) 对于b_j（$1 \leqslant j \leqslant i-1$），$H$中$b_j$的度要么为0要么为2。如果$b_j$与$R$中既在$M'_{i-1}$又在$M_i'$的同一个顶点相匹配，在$H$中$b_j$的度为0；否则，因为$M'_{i-1}$中的一条边与$M_i'$中的一条边关联于$b_j$，所以它的度为2。

(3) 由于对于任一r_j，r_j的度可为0、1或2。对此的证明可分为两种情况：

情况3.1：r_j不与M'_{i-1}和M_i'中任一顶点匹配，在这种情况，r_j在H中的度为0。

情况3.2：r_j与某一顶点相匹配。在此情况下，如果它与M'_{i-1}或M_i'中的某一顶点匹配，而不与两个集合中的顶点匹配，那么可以证明r_j在H中的度为1；如果它与M'_{i-1}和M_i'中同一个点匹配，那么它的度为0；如果它与M'_{i-1}和M_i'中不同的两个点匹配，那么它的度为2。

易知在H中，每个连通分支一定是一个最大交替路径或环路。由于b_i的度为1，也一定存在以b_i作为终点的交替路径。令P表示这条路径，可以证明在H中没有别的连通分支。换句话说，P是H中唯一的连通分支。

首先假设H中包含一条最大交替路径L，其中$L \neq P$。因为$L \neq P$，且P是以b_i作为终点的唯一最大交替路径，易见b_i不在L上。因为$B_i - \{b_i\}$中的每个顶点的度为0或2，且R中每个顶点的度为0、1或2，所以L的两个终点必在R中，且L的边数一定是偶数。另外，L在R中的顶点数要比在$B_i - \{b_i\}$中的顶点数多1。不失一般性，令在L中包含$2m$条边，其中$1 \leqslant m \leqslant i-1$，且在路径$L$上的顶点序列为$r_1$，$b_1$，$r_2$，$b_2$，…，$r_m$，$b_m$，$r_{m+1}$，其中$r_1$和$r_{m+1}$为$L$的终点。$r_1$，$r_2$，…，和$r_{m+1}$在$R$中，$b_1$，$b_2$，…，和$b_m$在$B_i - \{b_i\}$中。令$A_i$ (A_{i-1})表示L中属于M_i' (M_{i-1}')的边集。不失一般性，假设$A_i = \{(b_1, r_1), (b_2, r_2), \cdots, (b_m, r_m)\}$，$A_{i-1} = \{(b_1, r_2), (b_2, r_3), \cdots, (b_m, r_{m+1})\}$。注意在$H$中，由于$R$中度为1的顶点只与$M_{i-1}'$或$M_i'$相匹配，故端点$r_1$只与$M_i'$中的点相匹配，端点$r_{m+1}$只与$M_{i-1}'$中的点相匹配。换句话说，$M_i'$ (M_{i-1}')是$B_i = \{b_1, b_2, \cdots, b_i\}$($B_{i-1} = \{b_1, b_2, \cdots, b_{i-1}\}$)与$R + \{r_{m+1}\}$($R - \{r_1\}$)的子集相匹配的边集。

由于A_i是$\{b_1, \cdots, b_m\}$与$\{r_1, \cdots, r_m\}$匹配的边集合，那么$M_i' - A_i$是$B_i - \{b_1, \cdots, b_m\}$与$R - \{r_1, \cdots, r_m, r_{m+1}\}$的子集匹配的边集合；由于$A_{i-1}$是$\{b_1, \cdots, b_m\}$与$\{r_2, \cdots, r_{m+1}\}$匹配的边集合，那么$(M_i' - A_i) \cup A_{i-1}$是$b_i$与$R - \{r_1\}$的集合匹配的边集合。类似地，可证明$(M'_{i-1} - A_{i-1}) \cup A_i$是$B_{i-1}$与$R - \{r_{m+1}\}$的子集匹配的边集合。令$W(S)$表示边集合$S$的总权值，那么对于$L$有三种情况：

- $W(A_i) > W(A_{i-1})$

在这种情况下，存在匹配$M_i'' = (M_i' - A_i) \cup A_{i-1}$，使得$W(M_i'') < W(M_i')$。这与$M_i'$的最小性相矛盾。

- $W(A_i) < W(A_{i-1})$

在这种情况下，存在匹配$M''_{i-1} = (M'_{i-1} - A_{i-1}) \cup A_i$使得$W(M''_{i-1}) < W(M'_{i-1})$。这与$M'_{i-1}$的最小性相矛盾。

- $W(A_i) = W(A_{i-1})$

在这种情况下，存在匹配$M_i'' = (M_i' - A_i) \cup A_{i-1}$，使得$|M_i'' - M'_{i-1}| < |M_i' - M'_{i-1}|$。这与$M_i'$是$M_i' - M'_{i-1}$中选出的最小边数相矛盾。

因此，在H中除了P外再没有最大交替路径。

所以，$M'_{i-1} \oplus M_i'$由只包含一条以b_i为终点的最大交替路径P所组成。

由于H仅由一条最大交替路径P所组成，且b_j的度为0或2，其中$1 \leqslant j \leqslant i-1$，故$R$中必存在一个度为1的顶点$r'$作为$P$的端点，而$R - \{r'\}$中所有顶点的度为0或2。

此外，P中边数一定为奇数。因此，连接到端点b_i的边与连接到r'的边都来自同一个边集合。由于连接到b_i的边是M_i'中的一条边，连接到r'的边也是M_i'中的一条边。注意在H中，R中度为1的顶点匹配M_i'和M'_{i-1}中之一。因此，r'在M_i'中匹配。在H中，$R - \{r'\}$中度为0的顶点与b_i的同一个顶点匹配，或者既不与M_i'也不与M_{i-1}'匹配，$R - \{r'\}$中度为2的顶点与在M_i'和M_{i-1}'中不同的b_i顶点匹配。也就是，$R - \{r'\}$中的顶点或者都在或者都不在M_i'和M'_{i-1}中匹配。所以，r'是$R_i - R_{i-1}$中唯一的顶点，故对于$1 < i \leqslant n$，$|R_i - R_{i-1}| = 1$。

上面的讨论是非常重要的，因为它是补偿策略的基础。在本节介绍的在线算法只当B中的新元素出现时起作用，与先前的最优匹配相比，新的最优匹配只增加了R中的一个新元素。

已经证明了在线算法的有效性，接下来讨论在线算法的性能。将证明算法是$(2n-1)$可竞争的。也就是，令$C(M_n)$和$C(M_n')$分别表示M_n和M_n'的匹配代价，那么

$$C(M_n) \leqslant (2n-1)C(M_n')。$$

依据归纳证明，当$i = 1$时，显然，$C(M_1) = C(M_1')$，所以，

$C(M_i) \leqslant (2i-1)C(M_i')$

假设上面公式对于$i = 1$成立，且在M_i中，b_i与r_j匹配。令M_i''表示$M'_{i-1} \cup (b_i, r_j)$，H_i'表示$M_i' \oplus M_i''$。由于匹配M_i'和M_i''是b_i与R的子集R_i相匹配，故H_i'中的每个顶点的度为0或2。在继续证明之前，先看一些例子。

对于如图12-30所示的例子，$M_2'' = M_1' \cup (b_2, r_1)$表示在图12-31a中，$H_2' = M_2' \oplus M_2''$表示在图12-31b中。

由于在H_i'中每个顶点的度为0或2，所以对于H有两种情况：

情况1：所有H_i'中顶点的度都为0。

在这种情况下，M_i'和M_i''是相同的，因此，(b_i, r_j)必定属于M_i'，令$d(b_i, r_j)$表示b_i到r_j之间的距离，那么$d(b_i, r_j) \leqslant C(M_i)$。

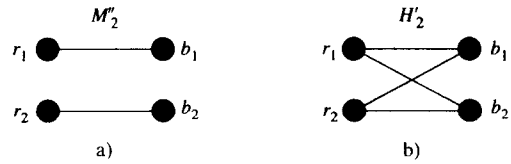


图12-31 对于图12-30情况的M_2''和H_2'

由于$C(M'_{i-1})$非负，可得$d(b_i, r_j) \leqslant C(M_i') + C(M'_{i-1})$。

情况2：H中有交替环路。可证明

$d(b_i, r_j) \leqslant C(M_i') + C(M_i'_{-1})$

对此情况证明从略。

由于$C(M_i) - C(M_{i-1}) = d(b_i, r_j)$，可得

$C(M_i) - C(M_{i-1}) = d(b_i, r_j) \leqslant C(M_i'_{-1}) + C(M')$。

$C(M_i) \leqslant C(M_{i-1}) + C(M'_{i-1}) + C(M_i')$

$\leqslant (2(i-1)-1)C(M'_{i-1}) + C(M'_{i-1}) + C(M_i')$ （根据归纳假设）

$\leqslant (2(i-1)-1)C(M'_i) + 2C(M_i')$ （根据$C(M'_{i-1}) \leqslant C(M')$）

$\leqslant (2i-1)C(M_i')$。

由上面证明可知该算法是（$2n-1$）可竞争的。那么问题是：它是否最优算法呢？在本节开始的论述可知非在线最小二分匹配算法可取得竞争比小于$(2n-1)$，故基于补偿策略的在线算法是最优的。

12.5 用适中策略解决在线m台机器调度问题

在本节中，将描述基于适中策略（moderation strategy）解决在线m台机器调度问题（on-line m-machine scheduling problem）的算法。在线m台机器调度问题定义如下：已知m台相同的机器及依次到来的任务，且当第i个任务到达时已知它的执行时间。只要有任务到达，就必须立刻给它分配其中一台机器执行。目标是在m台机器上无优先级地调度所有的任务，以最小化到最后任务完成时间的时间跨度。

一个直接的实现方法是贪心法，通过下面的例子说明。考虑六个任务j_1，j_2，…，j_6，以及它们的执行时间分别为1，2，…，6。贪心法把到达的任务以最小的总处理时间方案分配给不同的机器。最终的分配如图12-32所示。

如图12-32所示，在这种情况下，最长的完成时间为9，经常地表示为时间跨度。如果不采用贪心法，那么时间跨度可缩短。如图12-33所示的调度方案，时间跨度为7。

上述基于贪心法的算法称为列表算法（list algorithm）。可以证明该算法是（2-1/m）可竞争的。接下来介绍基于适中策略的算法。当$m \geqslant 70$时，该算法是（2−1/70）可竞争的。所以，该算法在$m \geqslant 70$时，其性能比列表算法更优。

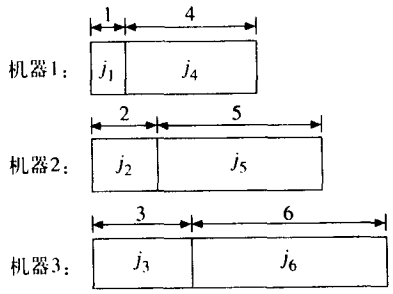


图12-32 m台机器调度问题实例

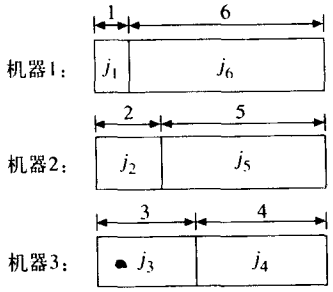


图12-33 图12-32中例子的更好调度

基本上，适中策略不是将到达任务分配给处理时间最短的机器，而是尽量分配到来任务给处理时间处于中间的机器，既不太长，也不太短。这就是为什么该策略称为适中策略的原因。

现在定义几个术语。首先，令a_i表示第i个到达任务的执行时间，其中$1 \leqslant i \leqslant n$。假设$m \geqslant 70$，$\varepsilon = 1/70$。令$\delta \in [0.445-1/(2m),\ 0.445+1/(2m)]$，且$\delta m$为整数。例如，如果$m = 80$，那么可取$\delta = 36/80$，它满足$\delta$范围的定义，且$\delta m = 36$是一个整数。

机器的高度（height）定义为已分配到该机器上的所有任务的长度总和。在任意时刻，算法都会根据机器当前的高度将它们排序成一个非递减序列。在时刻i，也就是，当第i个任务被调度时，可将该机器序列分为两个子序列：L_i和R_i。R_i是序列中前δm台机器子序列，L_i是剩余的$m-\delta m$台机器子序列，如图12-34所示。

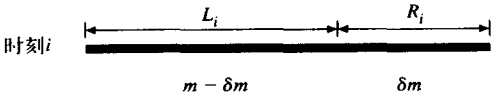


图12-34 L_i和R_i

令L_i和R_i的高度序列分别记为Lh_i和Rh_i，A_i和M_i分别表示在时刻i，m台机器的平均高度和最小高度，$A(P)$和$M(P)$分别表示在序列P中的平均高度和最小高度，其中P是一个高度序列。

基于适中策略的m台机器调度问题的在线算法描述如下：

当第$i+1$个任务到达时，若$M(Lh_i)+a_{i+1} \leqslant (2-\varepsilon)A(Rh_i)$，那么将第$i+1$个任务分配给$L_i$的第一台机器；否则，将它分配给$R_i$的第一台机器，即两个子序列机器中高度最小的机器。如果必要，可对机器重新排列，使得高度保持非递减。

接下来证明上述在线算法是$(2-\varepsilon)$可竞争的。该证明是相当复杂的，这里不提供完整的证明，只介绍证明的主要部分。

令ON_t和OPT_t分别表示在t时刻在线算法和最优算法的时间跨度。假设算法不是$(2-\varepsilon)$可竞争的，那么存在一个t使得

$$ON_t \leqslant (2-\varepsilon)OPT_t \text{ 和 } ON_{t+1} > (2-\varepsilon)\ OPT_{t+1}$$

考虑$M(Lh_t)$和$A(Rh_t)$。假设$M(Lh_t)+a_{t+1} \leqslant (2-\varepsilon)A(Rh_t)$，那么在线算法将$a_{t+1}$放在$L_i$中高度最短的机器上，因此

$$\begin{aligned} ON_{t+1} &= M(Lh_t)+a_{t+1} \\ &\leqslant (2-\varepsilon)A(Rh_t) \\ &\leqslant (2-\varepsilon)A_t \\ &\leqslant (2-\varepsilon)A_{t+1} \end{aligned} \tag{12-3}$$

因为OPT_i对于任务a_1，a_2，…，a_i来说是最优时间跨度，所以得到$m \times OPT_i \geqslant a_1+a_2+\cdots+a_i = m \times A_i$，因此，

$$OPT_i \geqslant A_i \tag{12-4}$$

把式（12-4）代入式（12-3）中，得到

$$ON_{t+1} \leqslant (2-\varepsilon)OPT_{t+1} \tag{12-5}$$

这与假设相矛盾。因此，$M(Lh_t)+a_{t+1}<(2-\varepsilon)A(Rh_t)$，且在线算法把$a_{t+1}$放在高度最短的机器上。所以，

$$ON_{t+1} = M_t + a_{t+1} \tag{12-6}$$

现在接着证明$M_t>(1-\varepsilon)A_t$。假设$M_t \leqslant (1-\varepsilon)A_t$，那么

$ON_{t+1} = M_t + a_{t+1}$ （根据公式(12-6)）

$\leqslant (1-\varepsilon)A_t + a_{t+1}$

$\leqslant (1-\varepsilon)A_{t+1} + a_{t+1}$

因为$A_{t+1} \leqslant OPT_{t+1}$和$a_{t+1} \leqslant OPT_{t+1}$，所以

$$ON_{t+1} \leqslant (2-\varepsilon)OPT_{t+1}$$

由于假设$ON_{t+1}>(2-\varepsilon)OPT_{t+1}$，所以是不可能的。因此，得到

$$M_t > (1-\varepsilon)A_t \tag{12-7}$$

不等式（12-7）表明，在时刻t执行时间长度最短的机器高度比$(1-\varepsilon)A_t$更长。事实上，可以证明一个更强的结论：在时刻t，每台机器包含的任务执行时间长度至少为$\frac{1}{2(1-\varepsilon)}A_t \geqslant \frac{1}{2(1-\varepsilon)}M_t$。后面将提供对它的证明。这里先假设其为真，这样很快建立起算法是（2−ε）可竞争的。

对于a_{t+1}，有两种可能的情况。

$a_{t+1} \leqslant (1-\varepsilon)M_t$，那么

$ON_{t+1} = M_t + a_{t+1}$ （根据公式(12-6)）

$\leqslant (2-\varepsilon)M_t$

$\leqslant (2-\varepsilon)M_{t+1}$

$\leqslant (2-\varepsilon)OPT_{t+1}$

这与$ON_{t+1} > (2-\varepsilon)OPT_{t+1}$相矛盾。

当$a_{t+1} > (1-\varepsilon)M_t$时，

$a_{t+1} > (1-\varepsilon)M_t$

$\geqslant \frac{1}{2(1-\varepsilon)}M_t$。$\left(根据(1-\varepsilon) \geqslant \frac{1}{2(1-\varepsilon)}\right)$

由于在时刻t每台机器执行有执行时间总长度至少为$\frac{1}{2(1-\varepsilon)}M_t$的任务，且$a_{t+1} \geqslant \frac{1}{2(1-\varepsilon)}M_t$，所以至少有$m+1$个任务的执行时间总长度至少为$\frac{1}{2(1-\varepsilon)}M_t$，其中必有两个任务在同一台机器上。所以

$$OPT_{t+1} \geqslant \max\left\{a_{t+1}, \frac{1}{2(1-\varepsilon)}M_t\right\}$$

这样，

$ON_{t+1} = M_t + a_{t+1}$ （根据公式(12-6)）

$\leqslant (1-\varepsilon)OPT_{t+1} + OPT_{t+1}$ $\left(根据a_{t+1} \leqslant OPT_t + 1，及\frac{1}{2(1-\varepsilon)}M_t \leqslant OPT_{t+1}\right)$

$= (2-\varepsilon)OPT_{t+1}$

这也与$ON_{t+1} > (2-\varepsilon)OPT_{t+1}$相矛盾。

无论哪种情况都有矛盾，所以，该在线算法是$(2-\varepsilon)$可竞争的。

现在证明更强的结论，在时刻t，每台机器上含有长度至少为$\frac{1}{2(1-\varepsilon)}A_t$的任务。为了证明该结论，注意上面已证明了$M_t > (1-\varepsilon)A_t$。由此，必定存在时刻$r<t$，$R_r$中的机器在时刻$r$的高度至多为$(1-\varepsilon)A_t$。

令$\tau\in[0.14-1/(2\delta m), 0.14+1/(2\delta m)]$，$m$为整数。$S_i$和$Sh_i$分别表示在时刻$t$，$m$台机器的最小高度序列和它们的高度序列，$s$表示只有在$S_s$中的及其有至多为$(1-\varepsilon)A_t$高度的时刻。易知$r<s<t$。

下面对三个关键时刻进行总结：

(1) 在r时刻，R_r中的机器的高度至多为$(1-\varepsilon)A_t$。

(2) 在s时刻，S_s中的机器的高度至多为$(1-\varepsilon)A_t$。

(3) 在t时刻，$M_t > (1-\varepsilon)A_t$。

图12-35对上述三个时刻进行了说明。

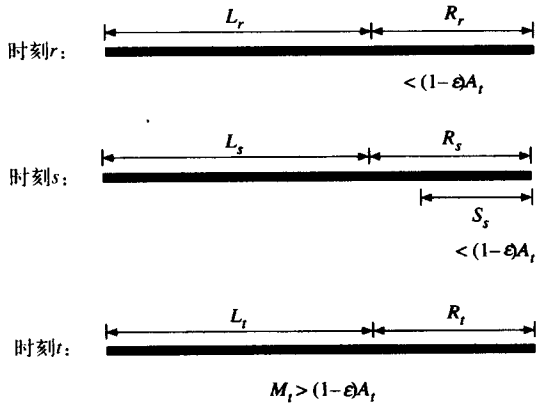


图12-35 对时刻r，s和t的说明

更进一步注意$S_s\in R_r$，可得到下面的三个断言：

断言1. 在t时刻，S_s中的每台机器含有一个至少为$\frac{1}{2(1-\varepsilon)}A_t$大小的任务。

断言2. 在t时刻，R_r中而不是S_s中的每台机器含有一个至少为$\frac{1}{2(1-\varepsilon)}A_t$的任务。

断言3. 在t时刻，L_r中的每台机器含有一个至少为$\frac{1}{2(1-\varepsilon)}A_t$的任务。

显然，一旦证明了上述三个结论，也就证明了在t时刻，每台机器含有一个长度至少为$\frac{1}{2(1-\varepsilon)}A_t$的任务。下面将证明断言1，因为另外两个断言的证明类似。

断言1的证明： 因为在s时刻，S_s中的每台机器的高度至多为$(1-\varepsilon)A_t$，且$M_t > (1-\varepsilon)A_t$（不等式（12-7）），故从s时刻到t时刻，S_s中的每台机器至少要接收一个任务。令p为S_s中的一台机器，j_{i+1}为机器p在s时刻后接收的第一个任务，其中$s < i \leqslant t-1$。由算法可知，该任务要么分

配到L_{i+1}的第一台机器，要么分配到所有机器中执行时间最短的机器。由于$p \in S_s$，故它不属于L_{i+1}。所以，p必是在$i+1$时刻执行时间最短的机器，故得到

$$M(Lh_i) + a_{i+1} > (2-\varepsilon)A(Rh_i)$$

使得

$$a_{i+1} > (2-\varepsilon)A(Rh_i) - M(Lh_i) \tag{12-8}$$

还可得到

$$\begin{aligned} A_t &\geqslant A_i \\ &= (1-\delta)A(Lh_i) + \delta A(Rh_i) \\ &\geqslant (1-\delta)M(Lh_i) + \delta A(Rh_i) \end{aligned}$$

这隐含着

$$M(Lh_i) \leqslant \frac{A_t - \delta A(Rh_i)}{1-\delta} \tag{12-9}$$

此外，

$$\begin{aligned} A(Rh_i) &\geqslant A(Rh_s) \\ &= (1-\tau)(R_s - S_s\text{的平均高度}) + \tau A(Sh_s) \\ &> (1-\tau)[(1-\varepsilon)A_t] + \tau A(Sh_s) \end{aligned} \tag{12-10}$$

（因为$R_s - S_s$中的机器高度至少为$(1-\varepsilon)A_t$）

这样可得

$$\begin{aligned} a_{i+1} &> (2-\varepsilon)A(Rh_i) - M(Lh_i) && \text{（根据不等式(12-8)）} \\ &> (2-\varepsilon)A(Rh_i) - \left[\frac{A_t - \delta A(Rh_i)}{1-\delta}\right] && \text{（根据不等式(12-9)）} \\ &= \left(2-\varepsilon+\frac{\delta}{1-\delta}\right)A(Rh_i) - \frac{\delta}{1-\delta}A_t \\ &= \left(2-\varepsilon+\frac{\delta}{1-\delta}\right)[(1-\tau)[(1-\varepsilon)A_t] + \tau A(Sh_s)] - \frac{\delta}{1-\delta}A_t && \text{（根据不等式(12-10)）} \\ &= \left\{\left[2-\varepsilon+\frac{\delta}{1-\delta}\right](1-\tau)(1-\varepsilon) - \frac{\delta}{1-\delta}\right\}A_t + \left\{\left[2-\varepsilon+\frac{\delta}{1-\delta}\right]\tau\right\}A(Sh_s) \\ &> \left\{\left[2-\varepsilon+\frac{\delta}{1-\delta}\right](1-\tau)(1-\varepsilon) - \frac{\delta}{1-\delta}\right\}A_t \end{aligned}$$

由于$\dfrac{1}{2(1-\varepsilon)} \approx 0.51$，及

$$\begin{aligned} &\left(2-\varepsilon+\frac{\delta}{1-\delta}\right)(1-\tau)(1-\varepsilon) - \frac{\delta}{1-\delta} \\ &\approx \left(2-\frac{1}{70}+\frac{0.455}{1-0.455}\right)(1-0.14)\left(1-\frac{1}{70}\right) - \frac{0.455}{1-0.455} \\ &= 0.56 \end{aligned}$$

对于$s < i \leqslant t-1$，可得

$$a_{i+1} > \frac{1}{2(1-\varepsilon)}A_t$$

12.6 基于排除策略的三个计算几何问题的在线算法

在本节中将介绍解决三个计算几何问题的在线算法。它们分别是凸包问题、最远点对问题和1圆心问题。最远点对问题（farthest pair problem）定义如下：在平面内点集S中找一对点，使它们之间的距离最大。这三个在线算法将给出近似解，但我们将证明这些近似解与精确解很接近。对于这三个问题，使用平行线遏制方法（parallel lines containment approach）。参见图12-36所示，它包含一个凸包。

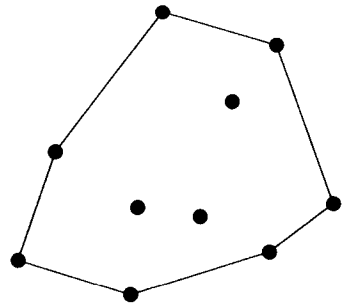


图12-36 一个凸包

设想新增一个顶点。如果这个点位于凸包里面，那么对它不作任何处理。这意味着只要记住该凸包的边界即可。如果这个点位于凸包外面，那么当然要对凸包的边界进行修改。

使用一组平行线形成输入点的边界，例如，如图12-37所示，有四对平行线，即（l_1，l_1'），（l_2，l_2'），（l_3，l_3'）和（l_4，l_4'），形成了输入点的边界。

在任意时刻一个点p到来时，首先查看它是否位于边界之外。如果是，则对边界进行修改，否则把不包括点p的l_i和l_i'看作一对平行线，不失一般性，且设l_i比l_i'更接近p。随后，不改变l_i斜率将其平行移动到点p。参见图12-38，p不在l_3和l_3'之间，可移动l_3到p。

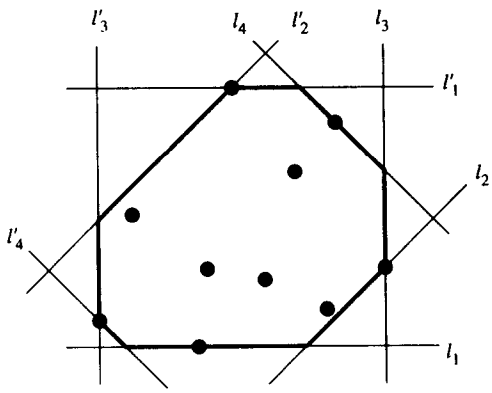


图12-37 由四对平行线组成的边界

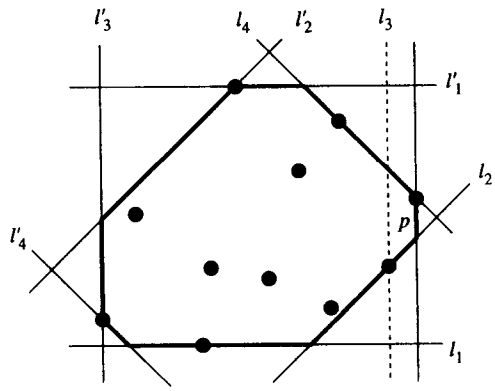


图12-38 在新增输入点后平移直线

一般，令边界由m对平行线组成，它们的斜率分别为0，$\tan(\pi/m)$，$\tan(2\pi/m)$，…，$\tan((m-1)\pi/m)$。这些平行线必须满足以下条件：

规则1：每个输入点都位于每对平行线之内。

规则2：每对平行线尽可能地接近。

现在介绍构建近似凸包的在线算法。由于每条直线上至少关联一个输入点，故可把这些点按顺时针方向连接起来，形成一个近似凸包。参见图12-39。图12-39a所示为初始近似凸包，在新增点p后，新的近似凸包如图12-39b所示。注意有一点可能位于凸多边形外面，这就是该算法称为近似算法的原因。

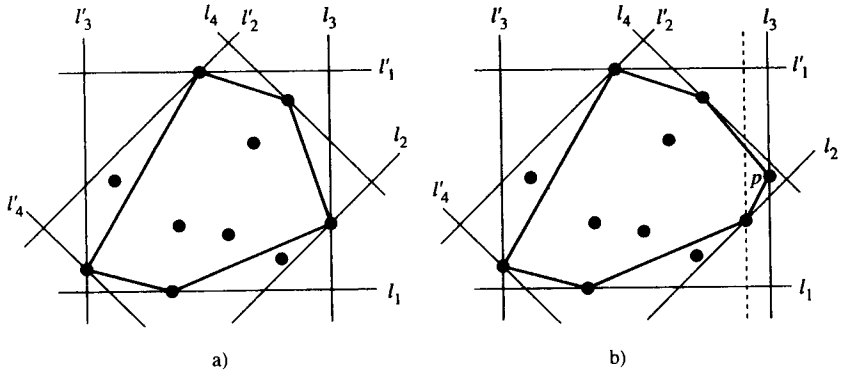


图12-39 近似凸包

在线近似凸包算法A介绍如下。

算法12-1 计算在线凸包算法A

输入：一个点序列p_1，p_2，…，及使用的平行线对数m。

输出：近似凸包序列a_1，a_2，…，其中a_i为覆盖点p_1，p_2，…，p_i的近似凸包。

初始化：构造斜率分别为0，$\tan(\pi/m)$，$\tan(2\pi/m)$，…，$\tan((m-1)\pi/m)$的m对平行线，并且定位所有的线都交于第一个输入点p_1。置$i=1$，也就是当前输入点为p_1。

步骤1：对于m对平行线中的每一对，如果点p_i位于它们之间，那么什么也不做；否则，不改变斜率地平移离p_i最近的直线，使之与p_i相交。

步骤2：顺时针方向连接位于m对直线上的$2m$个点，使之形成一个近似凸包，记为a_i。

步骤3：如果没有其他点输入，那么算法停止；否则，置$i=i+1$，令新接收点为p_i，转到步骤1。

算法A的时间复杂度显然为$O(mn)$，其中m为使用的平行线的对数，n为输入的点数。由于m是固定的，那么时间复杂度为$O(n)$。下面讨论该近似凸包的误差。与算法相关的有三种多边形：

(1) 多边形E：由$2m$条直线所围成的凸封闭多边形，所有的输入点都包含在其中。

(2) 多边形C：输入点所组成的凸包。

(3) 多边形A：由算法A所产生的近似凸包。

令$L(P)$表示多边形P的总边长，则显然$L(E)\geqslant L(C)\geqslant L(A)$。

在线近似凸包算法A产生的误差率$Err(A)$为

$$Err(A)=\frac{L(C)-L(A)}{L(C)}$$

参见图12-40。

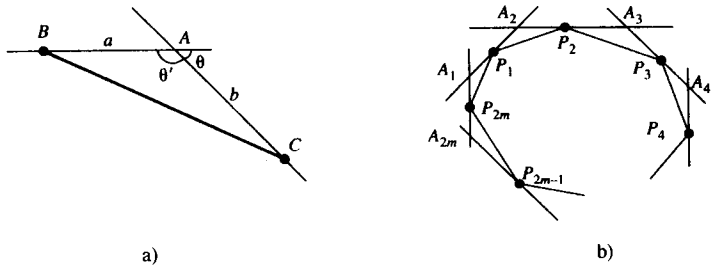


图12-40 $Err(A)$的估算

可以得到：

$$\left(\frac{\overline{AB}+\overline{AC}}{\overline{BC}}\right)^2=\frac{(a+b)^2}{a^2+b^2-2ab\cdot\cos\theta'}$$
$$=\frac{a^2+b^2+2ab}{a^2+b^2+2ab\cdot\cos\theta}$$
$$=\frac{(a^2+b^2)/2ab+1}{(a^2+b^2)/2ab+\cos\theta}$$
$$\leqslant\frac{1+1}{1+\cos\theta}$$
$$=\sec^2\frac{\theta}{2}$$

因此，$\overline{AB}+\overline{AC}\leqslant\sec\frac{\theta}{2}\cdot\overline{BC}$。

考虑图12-40b。

$$L(E)=\overline{P_{2m}A_1}+\overline{A_1P_1}+\overline{P_1A_1}+\cdots+\overline{A_{2m}P_{2m}}$$
$$\leqslant\sec\frac{\pi}{2m}\cdot(\overline{P_{2m}P_1}+\overline{P_1P_2}+\overline{P_2P_3}+\cdots+\overline{P_{2m-1}P_{2m}})$$
$$=\sec\frac{\pi}{2m}\cdot L(A)$$

所以有

$$Err(A)=\frac{L(C)-L(A)}{L(C)}\leqslant\frac{L(E)-L(A)}{L(A)}\leqslant\sec\frac{\pi}{2m}-1$$

上式说明使用的平行线越多，如所期望的误差越小。表12-1显示随m的增加，$Err(A)$如何减小。

表12-1 误差率的上界

m	$\sec\frac{\pi}{2m}$	$Err(A)$的上界
2	1.4142	0.4142
3	1.1547	0.1547
4	1.0824	0.0824
5	1.0515	0.0515
10	1.0125	0.0125
20	1.0031	0.0031

在线近似最远点对算法与前一节所介绍的在线近似凸包算法很类似。在所有平行线对中，总是找出一对点，它们间的距离最远。位于这对平行线上的这两点用作该在线近似最远点对算法的输出结果。

所以，除了步骤2之外，最远点对问题的在线算法几乎与算法A一样。步骤2可修改为：

步骤2：找出距离最远的一对平行线。置位于这对平行线上的这两点作为在线近似最远点对A_i，转到步骤3。

参见图12-41，令p_1和p_2是准确的最远点对，d_i表示第i对平行线间的距离，对于所有的i，$d_{\max}$表示d_i中的最长距离。参见图12-42，令N_i表示垂直于第i对平行线的方向，θ_i表示最远点对$\overline{P_1P_2}$与N_i间的锐角。注意$0\leqslant\theta_i\leqslant\frac{\pi}{2}$，令$o_1$

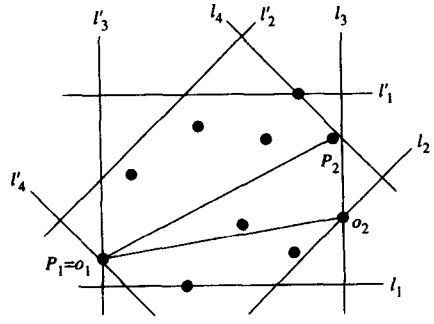


图12-41 精确和近似的最远点对的例子

和o_2为距离为$d_{\max}$的平行线对上的两个点。

可以得到如下：

$$
\begin{aligned}
\overline{P_1P_2} &\leqslant d_i \cdot \sec\theta_i, \forall i \in \{1,\cdots,m\} \\
&\leqslant \min_i\{d_i \sec\theta_i\} \\
&\leqslant \min_i\{d_{\max} \cdot \sec\theta_i\} \\
&= d_{\max} \min_i\{\sec\theta_i\} \\
&= d_{\max} \cdot \sec(\min_i\{\theta_i\}) \\
&= d_{\max} \cdot \sec\frac{\pi}{2m} \\
&= \overline{o_1o_2} \cdot \sec\frac{\pi}{2m}
\end{aligned}
$$

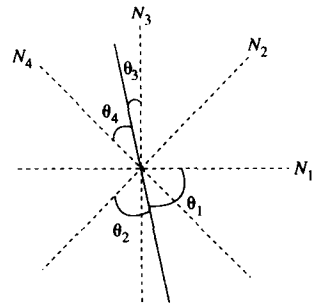


图12-42 近似最远点对与N_i间的锐角实例

因此，可得

$$
\begin{aligned}
\text{误差率} &= \frac{\overline{p_1p_2} - \overline{o_1o_2}}{\overline{p_1p_2}} \\
&\leqslant \frac{\overline{p_1p_2} - \overline{o_1o_2}}{\overline{o_1o_2}} \\
&\leqslant \sec\frac{\pi}{2m} - 1
\end{aligned}
$$

随着m的增加，误差率的上界在减小，如表12-1所示。在第6章介绍的1圆心问题定义如下：已知在欧几里得几何空间中的n个点的集合，找出一个能覆盖所有点的圆，且它的直径最短。在算法A中，有m对斜率固定的平行线，这m对平行线组成了凸封闭多边形E，它有如下性质：

性质1：E覆盖所有的输入点。

性质2：E中的顶点数至多为$2m$。

性质3：E中每条边的斜率一一对应等于$2m$条边的正多边形的各斜率。

性质4：E中的每条边上至少有一个点。

由E的上述四条性质可构建对于输入顶点的在线近似最小圆算法。同理，1圆心问题除了步骤2之外，在线近似最小圆算法与近似凸包算法A几乎一样。步骤2修改为：

步骤2：找出凸多边形E的顶点，即两直线的交点。使用任何离线1圆心算法解决这个最多包含$2m$个输入顶点的1圆心问题，令这个圆为在线近似圆A_i。

找出多边形E的顶点的时间复杂度为$O(m)$（性质2），找出覆盖$2m$个点的圆的时间复杂度是一个常数，因为只涉及一个固定的点数。

根据定义，通过算法找出的圆覆盖凸封闭多边形E，根据性质1，它也覆盖所有的输入点。下面的问题是分析在线近似最小圆有多好。所用的方法称为竞争分析（compeititive analysis）。令$d(C)$表示圆C的直径长度，已知一个点序列，令C_{opt}和C_{onl}分别表示覆盖所有这列点的最小圆和在线近似圆。如果所有的点位于平面上，那么1圆心问题的在线算法是c_k可竞争的，且$d(C_{onl}) \leqslant c_k d(C_{opt})$，后面将证明$c_k$是$\sec\frac{\pi}{2m}$。

为求C_{opt}，可构造一个$2m$条边的正多边形R和包含所有点的$2m$对相同斜率的平行线。令E表示由这$2m$对平行线所构成的多边形。如图12-43所示，$E \subset R$。令C_R表示覆盖R的最小圆，由于$E \subset R$，故此最小圆覆盖E，即C_{onl}，且小于等于C_R。由于C_{opt}为覆盖所有点的最优圆，所以有$d(C_{opt}) \leqslant d(C_{onl}) \leqslant d(C_R)$。参见图12-43，$\overline{OB}$为$C_{opt}$的半径，$\overline{OA}$为$C_R$的半径。可以得到

$$d(C_{opt}) \cdot \sec\frac{\pi}{2m} = d(C_R)$$

因此，

$$\frac{d(C_{onl})}{d(C_{opt})} \leqslant \frac{d(C_R)}{d(C_{opt})} = \frac{\overline{OA}}{\overline{OB}} = \sec\frac{\pi}{2m}$$

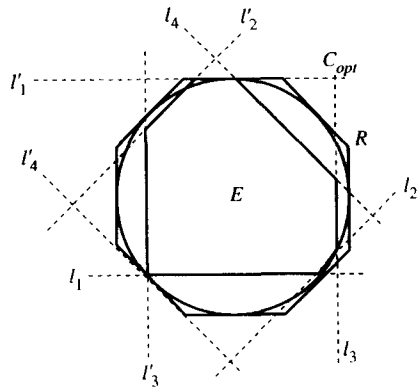


图12-43 E与R间的关系

这就是说，本节所介绍的在线算法是$\sec(\pi/(2m))$可竞争的。

12.7 基于随机策略的在线生成树算法

一般说来，在每一决策步中具有较少空间和较低时间复杂度的在线算法的性能对于在线问题是非常重要的，因为在实际应用中（例如，实时系统），当数据到达时，要尽可能快地做出决策。所以，为了满足低算法复杂度和空间要求的在线算法在决策步的随机性便成为在线算法的设计策略。在本节中，将介绍一个简单的随机算法来解决在线生成树问题，该问题在12.1节中定义。

解决欧几里得树的随机在线算法$R(m)$介绍如下。

算法12-2 计算在线欧几里得生成树的算法R(m)

输入：欧几里得空间中的$n(\geqslant 3)$个点v_1，…，v_n和一个正整数$m(\leqslant n-1)$。

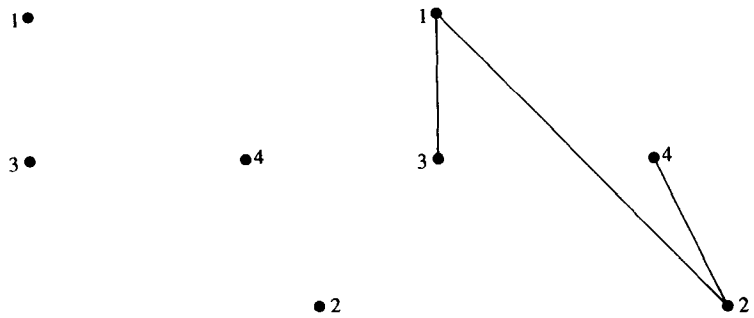
输出：欧几里得树T

```
Begin
  T = φ;
  input(m);
  input(v1);
  input(v2);
  添加连接v1和v2的边到T;
  For k = 3 to n do
      input(vk);
      If k≤m+1 then
          添加vk到v1, …, vk-1的最短边到T;
      else
          从vk到v1, …, vk-1的k-1条边中随机选择m条边，把这m 条边中的最短边添加到T;
  Endfor
  output(T);
End
```

依据输入序列的次序，算法$R(m)$依次接收输入点，当新的一个点到来时，算法做出决定构造到目前为止接收的已知点的生成树，且所有决策一经生成就不再改变。假设当前检查第i个输入点（比如v_i）。如果$2 \leqslant i \leqslant m+1$，那么算法$R(m)$把当前检查的点与先前检查过的点（比

如v_1，…，v_{i-1}）间的最短边添加到生成树中从而形成新的生成树；否则，从当前点与先前输入点形成的边中随机选择m条边，把这m条边中的最短边添加到原先的生成树中从而形成新的生成树。在每个检查步花费的时间是$O(m)$，不难看出算法$R(n-1)$是以前所介绍的确定性贪心算法。

假如已知平面上的4个点，如图12-44a所示，点的输入序列是1，2，3，4，构造一棵生成树，由算法$R(2)$将产生如图12-44b所示的生成树。表12-2给出了算法$R(2)$在输入序列为1，2，3，4的执行过程，其中$e(a, b)$表示连接点a和b的边，$|e(a, b)|$表示$e(a, b)$的欧几里得距离。



a) 输入的点　　b) 输入的序列

图12-44 算法$R(2)$的一个例子

表12-2 算法R(2)处理输入序列1，2，3，4的执行动作

当前输入的点	动　作
1	No action
2	Add $e(1, 2)$ to T
3	Add $e(3, 1)$ to T
	$(\|e(3, 1)\| = \min\{\|e(3, 1)\|, \|e(3, 2)\|\})$
4	Choose $e(4, 1)$ and $e(4, 2)$, and add $e(4, 2)$ to T
	$(\|e(4, 2)\| = \min\{\|e(4, 1)\|, \|e(4, 2)\|\})$

随机在线算法A称为c可竞争的，如果存在一个常数b，使得$E(C_A(\sigma)) \leqslant c \cdot C_{opt}(\sigma) + b$，其中$E(C_A(\sigma))$表示在输入为$\sigma$的算法$A$的期望代价。

下面将证明，如果m是个固定的常数，那么在线生成树算法$R(m)$的竞争比是$\Theta(n)$。

假设已知n个点1，2，…，n的集合σ，且点的输入序列是1，2，…，n。令$D(a, b)$表示点a和b间的距离，$N(a, k)$表示点a与目前为止所有点中的第k个最近点。$E(L_{R(m)}(\sigma))$表示由算法$R(m)$在输入为σ时所产生的生成树的期望长度。假设算法$R(m)$读入点i。如果$2 \leqslant i \leqslant m+1$，那么算法$R(m)$把$i$和先前已知的$i-1$个点中离它最近的点相连接；如果$m+2 \leqslant i \leqslant n$，那么算法$R(m)$把$i$和先前已知的$i-1$个点中离它最近的第$j$个点以概率$\binom{i-1-j}{m-1} \Big/ \binom{i-1}{m}$相连接，其中$1 \leqslant j \leqslant i-m$。对于$E(L_{R(m)}(\sigma))$有下面的公式：

$$E(L_{R(m)}(\sigma)) = \sum_{i=2}^{m+1} D(i, N(i,1)) + \sum_{i=m+2}^{n} \sum_{j=1}^{i-m} \binom{i-1-j}{m-1} \Big/ \binom{i-1}{m} \cdot D(i, N(i,j)) \tag{12-11}$$

令n个点位于同一直线且点i位于$\frac{(-1)^{i+1}}{2}+(-1)^i\left(\lceil i/2-1\rceil\right)\varepsilon$，其中$\varepsilon>0$（图12-45所示）。

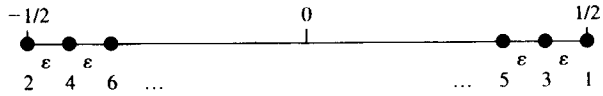


图12-45 n个点位于同一直线

如果ε的值无限小，那么可得

$$D(i,N(i,j))=\begin{cases}0 \ if\ 1\leqslant j\leqslant i-\lceil i/2\rceil-1\\ 1\ if\ i\leqslant\lceil i/2\rceil j\leqslant i-1\end{cases}$$

令σ^*表示上述的输入点，如果输入序列为1，2，…，n，那么由公式（12-11）可得：

$$\begin{aligned}
&E(L_{R(m)}(\sigma*))\\
&=1+\sum_{i=m+2}^{n}\sum_{j=1-\lceil i/2\rceil}^{i-m}\binom{i-1-j}{m-1}\Big/\binom{i-1}{m}\\
&=1+\sum_{i=m+2}^{n}1\Big/\binom{i-1}{m}\sum_{k=0}^{\lfloor i/2\rfloor-m}\binom{m-1+k}{k}\\
&=1+\sum_{i=m+2}^{n}\binom{\lfloor i/2\rfloor}{m}\Big/\binom{i-1}{m}\qquad\left(由\sum_{k=0}^{n}\binom{m+k}{k}=\binom{m+n+1}{n}得\right)\\
&\geqslant 1+\sum_{i=2m+1}^{n}\binom{(i-1)/2}{m}\Big/\binom{i-1}{m}\\
&=1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\frac{\prod_{k=\lceil m/2\rceil+1}^{m}(i-2k+1)}{\prod_{k=1}^{\lfloor m/2\rfloor}(i-2k)}\\
&=1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\prod_{k=1}^{\lfloor m/2\rfloor}\left(1-\frac{2\lceil m/2\rceil-1}{i-2k}\right)
\end{aligned}$$

由于m为常数，所以可得

$$\begin{aligned}
&\sum_{i=2m+1}^{n}\prod_{k=1}^{\lfloor m/2\rfloor}\left(1-\frac{2\lceil m/2\rceil-1}{i-2k}\right)>\sum_{i=2m+1}^{n}1-\sum_{k=1}^{\lfloor m/2\rfloor}\frac{2\lceil m/2\rceil-1}{i-2k}\\
&=(n-2m)-(2\lceil m/2\rceil-1)\left[\frac{1}{2m-1}+\frac{1}{2m-3}+\cdots+\frac{1}{2m+1-2\lceil m/2\rceil}\right.\\
&\quad+\frac{1}{2m}+\frac{1}{2m-2}+\cdots+\frac{1}{2m+2-2\lceil m/2\rceil}+\frac{1}{2m-1}+\frac{1}{2m-1}+\cdots+\frac{1}{2m+3-2\lceil m/2\rceil}+\cdots\\
&\quad\left.+\frac{1}{n-2}+\frac{1}{n-4}+\cdots+\frac{1}{n-2\lceil m/2\rceil}\right]\\
&=(n-2m)-(2\lceil m/2\rceil-1)\left[\sum_{k=1}^{\lceil m/2\rceil}(H_{n-2k}-H_{2m-2k})\right]\qquad\left(其中\ H_n=\sum_{i=1}^{n}\frac{1}{i}\right)\\
&=n-2m-o(n)
\end{aligned}$$

这样，

$$
\begin{aligned}
&E(L_{R(m)}(\sigma^*)) \\
&\geqslant 1+\frac{1}{2^m}\sum_{i=2m+1}^{n}\prod_{k=1}^{\lfloor m/2\rfloor}\left(1-\frac{2\lceil m/2\rceil-1}{i-2k}\right) \\
&=1+\frac{n-2m-o(n)}{2^m}
\end{aligned}
$$

令$L_{span}(\sigma^*)$表示输入为σ^*的最小生成树的长度，易知$L_{span}(\sigma^*)=1$，因此，

$$
\frac{E(L_{R(m)}(\sigma^*))}{L_{span}(\sigma^*)} \geqslant 1+\frac{n-2m-o(n)}{2^m}
$$

由此可知，在线生成树问题的$R(m)$算法的竞争比是$\Omega(n)$，其中n为输入点的数目，m为固定的常数。

令D表示n个点的直径，最小施泰纳树（minimum Steiner tree）与最小生成树的长度大于或等于D。因为每条边的长度小于或等于D，所以由任何在线算法所产生的生成树的长度小于或等于$(n-1)D$。因此，对于在线生成树问题，没有在线算法的竞争比大于$n-1$，其中n为输入点的数目。

基于上面的结论，可得在线生成树问题的算法$R(m)$的竞争比是$\Theta(n)$，其中n为输入点的数目，m为固定的常数。

12.8 注释与参考

在线施泰纳树问题与在线生成树问题相似，是找出一棵在线施泰纳树，其中的结点也是一个接一个地出现。在文献Imase and Waxman(1991)和Alon and Azar(1993)中，对12.1节中两个问题的贪心算法的竞争比证明是$\log_2 n$。此外，文献Alon and Azar(1993)还证明了在线生成树和施泰纳树问题的任何在线算法的竞争比的下界为$\Omega(\log n/\log\log n)$。

如果从点i到点j的距离与从j到i的距离相等，那么服务员问题称为对称的问题，否则称为非对称的问题。解决k服务员问题的在线算法要在具备一定条件下执行，即要在执行之前决定移动哪名服务员能满足当前请求而不必知道将来的请求。

例如高速缓存问题，页面分配问题，双磁盘磁头问题，线性搜索问题等，在选择了顶点与服务员数量之间距离后可抽象为服务员问题。

文献Manasse，McGeoch and Sleator(1990)于1990年引入了服务员问题，他们证明了k是解决k服务员问题算法的竞争比下限。而基于平面树的k服务员最优算法在文献Chrobak and Larmore(1991)中提出。

穿越障碍物问题是在Papadimitriou and Yannakakis(1991b)中首次提出的。他们证明当障碍物的边与坐标轴平行时的确定算法的竞争比下界为$\Omega(\sqrt{d})$，其中d为s和t间的距离。如果所有障碍物都是相同边长的方形，且边都与坐标轴平行，那么他们同时提出了渐近最优竞争比为3/2的算法。

在12.3节中使用的不等式$\frac{\tau_1}{\pi_1}<\frac{2}{3}$或$\frac{\tau_2}{\pi_2}<\frac{2}{3}$在文献Fejes(1978)中证明。

二分匹配问题是在文献Kalyanasundaram and Pruhs(1993)中提出的，在12.4节中介绍的算法及问题的下限也是在文献Kalyanasundaram and Pruhs(1993)中给出。

在1996年，Graham提出了解决m台机器调度问题的简单贪心算法，称为List。他证明了该

算法的竞争比为$\left(2-\frac{1}{m}\right)$。在12.5节中提到的竞争比为$\left(2-\frac{1}{70}\right)$的解决$m$台机器调度问题的算法是在文献Bartal，Fiat，Karloff and Vahra(1995)中提出的，其中$m\geqslant 70$。

在12.6节中解决几何问题的在线算法可在文献Chao(1992)中找到。在12.7节中解决在线生成树的随机算法在文献Tsai和Tang(1993)中有讨论。

12.9 进一步的阅读资料

文献Manasse，McGeoch and Sleator(1990)证明了对称2服务员问题的确定型在线算法的最优竞争比为2，并猜测对于$c<k$，至少有$k+1$个顶点的任意度量空间，k服务员问题没有确定的竞争比为c的在线算法。文献Koutsoupias and Papadimitriou(1995)已经证明了解决k服务员问题的工作函数算法的竞争比最大为$2k-1$，但猜测迄今还没有被证明或反驳。

文献Chan and Lam(1993)提出了对于穿越障碍物问题的非对称最优的$(1+r/2)$竞争算法，其中每个障碍物有某个常数r限界的高宽比（长边与短边的长度比）。

文献Khuller，Mitchell and Vazirani(1994)提出了带权匹配和稳定合并的在线算法。在文献Karp，Vazirani(1990)和Kao，Tate(1991)中讨论了无权图的二分匹配问题。

文献Karger，Phillips and Torng(1994)提供了一个解决m台机器调度问题的算法，对于所有的m，该算法的竞争比至多是1.945，且对于$m\geqslant 6$，它优于List算法。

许多有趣的在线算法正在研究中，如测量任务系统：Borodin，Linial and Saks(1992)；在线装箱问题：Csirik(1989)；Lee and Lee(1985)；Vliet(1992)；在线图着色问题：Vishwanathan(1992)；在线财政问题：El-Yaniv(1998)等。

下面所列的是感兴趣的较新的研究成果：Adamy and Erlebach(2003)；Albers(2002)；Albers and Koga(1998)；Albers and Leonardi(1999)；Alon，Awerbuch and Azar(2003)；Aspnes，Azar，Fiat，Plotkin and Waarts(1997)；Awerbuch and Peleg(1995)；Awerbuch and Singh(1997)；Awerbuch，Azar and Meyerson(2003)；Azar，Blum and Mansour(2003)；Azar，Kalyanasundaram，Plotkin，Pruhs and Waarts(1997)；Azar，Naor and Rom(1995)；Bachrach and ElYaniv(1997)；Bareli，Berman，Fiat and Yan(1994)；Bartal and Grove(2000)；Berman，Charikar and Karpinski(2000)；Bern，Greene，Raghunathan and Sudan(1990)；Blum，Sandholm and Zinkevich(2002)；Caragiannis，Kaklamanis and Papaioannou(2003)；Chan(1998)；Chandra and Vishwanathan(1995)；Chazelle and Matousek(1996)；Chekuri，Khanna and Zhu(2001)；Conn and Vonholdt(1965)；Coppersmith，Doyle，Raghavan and Snir(1993)；Crammer and Singer(2002)；El-Yaniv(1998)；Epstein and Ganot(2003)；Even and Shiloach(1981)；Faigle，Kern and Nawijn(1999)；Feldmann，Kao，Sgall and Teng(1993)；Galil and Seiferas(1978)；Garay，Gopal，Kutten，Mansour and Yung(1997)；Goldman，Parwatikar and Suri(2000)；Gupta，Konjevod and Varsamopoulos(2002)；Haas and Hellerstein(1999)；Halldorsson(1997)；Halperin，Sharir and Goldberg(2002)；Irani，Shukla and Gupta(2003)；Janssen，Krizanc，Narayanan and Shende(2000)；Jayram，Kimbrel，Krauthgamer，Schieber and Sviridenko(2001)；Kalyanasundaram and Pruhs(1993)；Keogh，Chu，Hart and Pazzani(2001)；Khuller，Mitchell and Vazirani(1994)；Klarlund(1999)；Kolman and Scheideler(2001)；Koo，Lam，Ngan，Sadakane and To(2003)；Kossmann，Ramsak and Rost(2002)；Lee(2003a)；Lee(2003b)；Lueker(1998)；Manacher(1975)；Mandic and Cichocki(2003)；Mansour and Schieber(1992)；Megow and Schulz(2003)；Oza and

Russell(2001)；Pandurangan and Upfal(2001)；Peserico(2003)；Pittel and Weishaar(1997)；Ramesh(1995)；Seiden(1999)；Seiden(2002)；Sgall(1996)；Tamassia(1996)；Tsai，Lin and Hsu(2002)；Tsai，Tang and Chen(1994)；Tsai，Tang and Chen(1996)；Ye and Zhang(2003)；and Young(2000)。

习题

12.1 考虑在12.4节中介绍的二分匹配问题，当B中的某点b_i到达时，把b_i与集合R中与它最近且没有匹配的点相匹配，这样的算法是（$2n-1$）可竞争的吗？证明之。

12.2 已知由点集R和B形成的二分带权图，它们的基数都为n，最大二分匹配问题是找到花费代价最大的二分匹配。假设所有的边权值满足三角不等式。如果R中的点事先已知而B中的点依次出现，那么总是将到来的顶点与R中最远的尚未匹配点相匹配的贪心算法竞争比是多少？

12.3 证明在12.2节中所介绍的k服务员算法，当服务员排成一列时其竞争比也为k。

12.4 如果方形障碍物具有任意方向，那么在12.3节中介绍的穿越障碍物算法的竞争比是3/2可竞争的吗？并证明之。

参考文献

Abel, S. (1990): A Divide and Conquer Approach to Least-Squares Estimation, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 26, pp. 423–427.

Adamy, U. and Erlebach, T. (2003): Online Coloring of Intervals with Bandwidth, *LNCS 2909*, pp. 1–12.

Agarwal, P. K. and Procopiuc. C. M. (2000): Approximation Algorithms for Projective Clustering, *Journal of Algorithms*, Vol. 34, pp. 128–147.

Agarwal, P. K. and Sharir, M. (1996): Efficient Randomized Algorithms for Some Geometric Optimization Problems, *Discrete Computational Geometry*, Vol. 16, No. 4, pp. 317–337.

Agrawal, M., Kayal, N. and Saxena, N. (2004): PRIMES is in P, *Annals of Mathematics* (forthcoming).

Aho, A. V., Ganapathi, M. and Tjang, S. (1989): Code Generation Using Tree Matching and Dynamic Programming, *ACM Transactions on Programming Languages and Systems*, Vol. 11, pp. 491–516.

Aho, A. V., Hopcroft, J. E. and Ullman, J. D. (1974): *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.

Ahuja, R. K. (1988): Minimum Cost-Reliability Ratio Path Problem, *Computer and Operations Research*, Vol. 15, No. 1.

Aiello, M., Rajagopalan, S. R. and Venkatesan, R. (1998): Design of Practical and Provably Good Random Number Generators, *Journal of Algorithms*, pp. 358–389.

Akiyoshi, S. and Takeaki, U. (1997): A Linear Time Algorithm for Finding a k-Tree Core, *Journal of Algorithms*, Vol. 23, pp. 281–290.

Akutsu, T. (1996): Protein Structure Alignment Using Dynamic Programming and Iterative Improvement, *IEICE Transactions on Information and Systems*, Vol. E78-D, No. 12, pp. 1629–1636.

Akutsu, T., Arimura, H. and Shimozono, S. (2000): On Approximation Algorithms for Local Multiple Alignment, *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, ACM Press, Tokyo, pp. 1–7.

Akutsu, T. and Halldorsson, M. M. (1994): On the Approximation of Largest Common Subtrees and Largest Common Point Sets, *Lecture Notes in Computer Science*, pp. 405–413.

Akutsu, T. and Miyano, S. (1997): On the Approximation of Protein Threading, *RECOMB*, pp. 3–8.

Akutsu, T., Miyano, S. and Kuhara, S. (2003): A Simple Greedy Algorithm for Finding Functional Relations: Efficient Implementation and Average Case Analysis, *Theoretical Computer Science*, Vol. 292, pp. 481–495.

Albers, S. (2002): On Randomized Online Scheduling, *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, ACM Press, Montreal, Canada, pp. 134–143.

Albers, S. and Koga, H. (1998): New On-Line Algorithms for the Page Replication Problem, *Journal of Algorithms*, Vol. 27, pp. 75–96.

Albers, S. and Leonardi, S. (1999): On-Line Algorithms, *ACM Computing Surveys (CSUR)*, Vol. 31, No. 3, Article No. 4, September.

Alberts, D. and Henzinger, M. R. (1995): Average Case Analysis of Dynamic Graph Algorithms, *Symposium in Discrete Algorithms*, pp. 312–321.

Aldous, D. (1989): *Probability Approximations via the Poisson Clumping Heuristic*, Springer-Verlag, Berlin.

Aleksandrov, L. and Djidjev, H. (1996): Linear Algorithms for Partitioning

Embedded Graphs of Bounded Genus, *SIAM Journal on Discrete Mathematics*, Vol. 9, No. 1, pp. 129–150.

Alon, N., Awerbuch, B. and Azar, Y. (2003): Session 2B: The Online Set Cover Problem, *Proceedings of the 35th ACM Symposium on Theory of Computing*, ACM Press, San Diego, California, pp. 100–105.

Alon, N. and Azar, Y. (1989): Finding an Approximate Maximum, *SIAM Journal on Computing*, Vol. 18, No. 2, pp. 258–267.

Alon, N. and Azar, Y. (1993): On-Line Steiner Trees in the Euclidean Plane, *Discrete Computational Geometry*, Vol. 10, No. 2, pp. 113–121.

Alon, N., Babai, L. and Itai, A. (1986): A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem, *Journal of Algorithms*, Vol. 7, No. 4, pp. 567–583.

Alpert, C. J. and Kahng, A. B. (1995): Multi-Way Partitioning via Geometric Embeddings; Orderings; and Dynamic Programming, *IEEE Transactions on CAD*, Vol. 14, pp. 1342–1358.

Amini, A. A., Weymouth, T. E. and Jain, R. C. (1990): Using Dynamic Programming for Solving Variational Problems in Vision, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 9, pp. 855–867.

Amir, A. and Farach, M. (1995): Efficient 2-Dimensional Approximate Matching of Half-Rectangular Figures, *Information and Computation*, Vol. 118, pp. 1–11.

Amir, A. and Keselman, D. (1997): Maximum Agreement Subtree in a Set of Evolutionary Trees: Metrics and Efficient Algorithms, *SIAM Journal on Computing*, Vol. 26, pp. 1656–1669.

Amir, A. and Landau, G. (1991): Fast Parallel and Serial Multidimensional Approximate Array Matching, *Theoretical Computer Science*, Vol. 81, pp. 97–115.

Anderson, R. (1987): A Parallel Algorithm for the Maximal Path Problem, *Combinatorica*, Vol. 7, No. 4, pp. 315–326.

Anderson, R. J. and Woll, H. (1997): Algorithms for the Certified Write-All Problem, *SIAM Journal on Computing*, Vol. 26, No. 5, pp. 1277–1283.

Ando, K., Fujishige, S. and Naitoh, T. (1995): A Greedy Algorithm for Minimizing a Separable Convex Function over a Finite Jump System, *Journal of the Operations Research Society of Japan*, Vol. 38, pp. 362–375.

Arkin, E. M., Chiang, Y. J., Mitchell, J. S. B., Skiena, S. S. and Yang, T. C. (1999): On the Maximum Scatter Traveling Salesperson Problem, *SIAM Journal on Computing*, Vol. 29, pp. 515–544.

Armen, C. and Stein, C. (1994): A 2 3/4-Approximation Algorithm for the Shortest Superstring Problem, *Technical Report* (PCS-TR94-214), Computer Science Department, Dartmouth College, Hanover, New Hampshire.

Armen, C. and Stein, C. (1995): Improved Length Bounds for the Shortest Superstring Problem (Shortest Common Superstring: 2 3/4-Approximation), *Lecture Notes in Computer Science*, Vol. 955, pp. 494–505.

Armen, C. and Stein, C. (1996): A 2 2/3 Approximation Algorithm for the Shortest Superstring Problem, *Lecture Notes in Computer Science*, Vol. 1075, pp. 87–101.

Armen, C. and Stein, C. (1998): 2 2/3 Superstring Approximation Algorithm, *Discrete Applied Mathematics*, Vol. 88, No. 1–3, pp. 29–57.

Arora, S. (1996): Polynomial Approximation Schemes for Euclidean TSP and Other Geometric Problems, *Foundations of Computer Science*, pp. 2–13.

Arora, S. and Brinkman, B. (2002): A Randomized Online Algorithm for Bandwidth Utilization, *Symposium on Discrete Algorithms*, pp. 535–539.

Arora, S., Lund, C., Motwani, R., Sudan, M. and Szegedy, M. (1998): Proof Verification and the Hardness of Approximation Problems (Prove NP-Complete Problem), *Journal of the ACM*, Vol. 45, No. 3, pp. 501–555.

Arratia, R., Goldstein, L. and Gordon, L. (1989): Two Moments Suffice for Poisson Approximation: The Chen-Stein Method, *The Annals of Probability*, Vol. 17, pp. 9–25.

Arratia, R., Martin, D., Reinert, G. and Waterman, M. S. (1996): Poisson Process Approximation for Sequence Repeats and Sequencing by Hybridization, *Journal of Computational Biology*, Vol. 3, pp. 425–464.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R. and Wu, A. Y. (1998): An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions, *Journal of the ACM*, Vol. 45, No. 6, pp. 891–923.

Ashenhurst, R. L. (1987): *ACM Turing Award Lectures: The First Twenty Years: 1966–1985*, ACM Press, Baltimore, Maryland.

Aspnes, J., Azar, Y., Fiat, A., Plotkin, S. and Waarts, O. (1997): On-Line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling, *Journal of the ACM*, Vol. 44, No. 3, pp. 486–504.

Atallah, M. J. and Hambrusch, S. E. (1986): An Assignment Algorithm with Applications to Integrated Circuit Layout, *Discrete Applied Mathematics*, Vol. 13, No. 1, pp. 9–22.

Auletta, V., Parente, D. and Persiano, G. (1996): Dynamic and Static Algorithms for Optimal Placement of Resources in Trees, *Theoretical Computer Science*, Vol. 165, No. 2, pp. 441–461.

Ausiello, G., Crescenzi, P. and Protasi, M. (1995): Approximate Solution of NP Optimization Problems, *Theoretical Computer Science*, Vol. 150, pp. 1–55.

Avis, D., Bose, P., Shermer, T. C., Snoeyink, J., Toussaint, G. and Zhu, B. (1996): On the Sectional Area of Convex Polytopes, *Proceedings of the Twelfth Annual Symposium on Computational Geometry*, ACM Press, Philadelphia, Pennsylvania, pp. 411–412.

Avrim, B., Jiang, T., Li, M., Tromp, J. and Yannakakis, M. (1991): Linear Approximation of Shortest Superstrings, *Proceedings of the 23rd ACM Symposium on Theory of Computation*, ACM Press, New Orleans, Louisiana, pp. 328–336.

Awerbuch, B., Azar, Y. and Meyerson, A. (2003): Reducing Truth-Telling Online Mechanisms to Online Optimization, *Proceedings of the 35th ACM Symposium on Theory of Computing*, ACM Press, San Diego, California, pp. 503–510.

Awerbuch, B. and Peleg, D. (1995): On-Line Tracking of Mobile Users, *Journal of the ACM*, Vol. 42, No. 5, pp. 1021–1058.

Awerbuch, B. and Singh, T. (1997): On-Line Algorithms for Selective Multicast and Maximal Dense Trees, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, ACM Press, El Paso, Texas, pp. 354–362.

Azar, Y., Blum, A. and Mansour, Y. (2003): Combining Online Algorithms for Rejection and Acceptance, *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, San Diego, California, pp. 159–163.

Azar, Y., Kalyanasundaram, B., Plotkin, S., Pruhs, K. and Waarts, O. (1997): On-Line Load Balancing of Temporary Tasks, *Journal of Algorithms*, Vol. 22, pp. 93–110.

Azar, Y., Naor, J. and Rom, R. (1995): The Competitiveness of On-Line Assignments, *Journal of Algorithms*, Vol. 18, pp. 221–237.

Bachrach, R. and El-Yaniv, R. (1997): Online List Accessing Algorithms and Their Applications: Recent Empirical Evidence, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, New Orleans, Louisiana, pp. 53–62.

Baeza-Yates, R. A. and Navarro, G. (1999): Faster Approximate String Matching, *Algorithmica*, Vol. 23, No. 2, pp. 127–158.

Baeza-Yates, R. A. and Perleberg, C. H. (1992): Fast and Practical Approximate String Matching, *Lecture Notes in Computer Science*, Vol. 644, pp. 185–192.

Bafna, V., Berman, P. and Fujito, T. (1999): A 2-Approximation Algorithm for the Undirected Feedback Vertex Set Problem, *SIAM Journal on Discrete Mathematics*, Vol. 12, No. 3, pp. 289–297.

Bafna, V., Lawler, E. L. and Pevzner, P. A. (1997): Approximation Algorithms for Multiple Sequence Alignment, *Theoretical Computer Science*, Vol. 182, pp. 233–244.

Bafna, V. and Pevzner, P. (1996): Genome Rearrangements and Sorting by Reversals (Approximation Algorithm), *SIAM Journal on Computing*, Vol. 25, No. 2, pp. 272–289.

Bafna, V. and Pevzner, P. A. (1998): Sorting by Transpositions, *SIAM Journal on Discrete Mathematics*, Vol. 11, No. 2, pp. 224–240.

Bagchi, A. and Mahanti, A. (1983): Search Algorithms under Different Kinds of Heuristics—A Comparative Study, *Journal of the ACM*, Vol. 30, No. 1, pp. 1–21.

Baker, B. S. (1994): Approximation Algorithms for NP-Complete Problems on Planar Graphs, *Journal of the ACM*, Vol. 41, No. 1.

Baker, B. S. and Coffman, E. G. Jr. (1982): A Two-Dimensional Bin-Packing Model of Preemptive FIFO Storage Allocation, *Journal of Algorithms*, Vol. 3, pp. 303–316.

Baker, B. S. and Giancarlo, R. (2002): Sparse Dynamic Programming for Longest Common Subsequence from Fragments, *Journal of Algorithm*, Vol. 42, pp. 231–254.

Balas, F. and Yu, C. S. (1986): Finding a Maximum Clique in an Arbitrary Graph, *SIAM Journal on Computing*, Vol. 15, No. 4, pp. 1054–1068.

Bandelloni, M., Tucci, M. and Rinaldi, R. (1994): Optimal Resource Leveling Using Non-Serial Dynamic Programming, *European Journal of Operational Research*, Vol. 78, pp. 162–177.

Barbu, V. (1991): The Dynamic Programming Equation for the Time Optimal Control Problem in Infinite Dimensions, *SIAM Journal on Control and Optimization*, Vol. 29, pp. 445–456.

Bareli, E., Berman, P., Fiat, A. and Yan, P. (1994): Online Navigation in a Room, *Journal of Algorithms*, Vol. 17, pp. 319–341.

Bartal, Y., Fiat, A., Karloff, H. and Vohra, R. (1995): New Algorithms for an Ancient Scheduling Problem, *Journal of Computer and System Sciences*, Vol. 51, No. 3, pp. 359–366.

Bartal, Y. and Grove, E. (2000): The Harmonic k-Server Algorithm is Competitive, *Journal of the ACM*, Vol. 47, No. 1, pp. 1–15.

Basse, S. and Van Gelder, A. (2000): *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Reading, Mass.

Bein, W. W. and Brucker, P. (1986): Greedy Concepts for Network Flow Problems, *Discrete Applied Mathematics*, Vol. 15, No. 2, pp. 135–144.

Bein, W. W., Brucker, P. and Tamir, A. (1985): Minimum Cost Flow Algorithm for Series-Parallel Networks, *Discrete Applied Mathematics*, Vol. 10, No. 3, pp. 117–124.

Bekesi, J., Galambos, G., Pferschy, U. and Woeginger, G. (1997): Greedy Algorithms for On-Line Data Compression, *Journal of Algorithms*, Vol. 25, pp. 274–289.

Bellman, R. and Dreyfus, S. E. (1962): *Applied Dynamic Programming*, Princeton University Press, Princeton, New Jersey.

Ben-Asher, Y., Farchi, E. and Newman, I. (1999): Optimal Search in Trees, *SIAM Journal on Mathematics*, Vol. 28, No. 6, pp. 2090–2102.

Bent, S. W., Sleator, D. D. and Tarjan, R. E. (1985): Biased Search Trees, *SIAM Journal on Computing*, Vol. 14, No. 3, pp. 545–568.

Bentley, J. L. (1980): Multidimensional Divide-and-Conquer, *Communications of*

the ACM, Vol. 23, No. 4, pp. 214–229.

Bentley, J. L., Faust, G. M. and Preparata, F. P. (1982): Approximation Algorithms for Convex Hulls, *Communications of the ACM*, Vol. 25, pp. 64–68.

Bentley, J. L. and McGeoch, C. C. (1985): Amortized Analysis of Self-Organizing Sequential Search Heuristics, *Communications of the ACM*, Vol. 28, No. 4, pp. 404–411.

Bentley, J. L. and Shamos, M. I. (1978): Divide-and-Conquer for Linear Expected Time, *Information Processing Letters*, Vol. 7, No. 2, pp. 87–91.

Berger, B. and Leighton, T. (1998): Protein Folding in the Hydrophobic-Hydrophilic (HP) Model is NP-Complete (Prove NP-Complete Problem), *Journal of Computational Biology*, Vol. 5, No. 1, pp. 27–40.

Berger, R. (1966): The Undecidability of the Domino Problem, *Memoirs of the American Mathematical Society*, No. 66.

Berman, P., Charikar, M. and Karpinski, M. (2000): On-Line Load Balancing for Related Machines, *Journal of Algorithms*, Vol. 35, pp. 108–121.

Berman, P., Hannenhalli, S. and Karpinki, M. (2001): 1.375 – Approximation Algorithm for Sorting by Reversals, *Technical Report DIMACS*, TR2001-41.

Berman, P., Karpinski, M., Larmore, L. L., Plandowski, W. and Rytter, W. (2002): On the Complexity of Pattern Matching for Highly Compressed Two-Dimensional Texts, *Journal of Computer System Sciences*, Vol. 65, No. 2, pp. 332–350.

Bern, M., Greene, D., Raghunathan, A. and Sudan, M. (1990): Online Algorithms for Locating Checkpoints, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, ACM Press, Baltimore, Maryland, pp. 359–368.

Bhagavathi, D., Grosch, C. E. and Olariu, S. (1994): A Greedy Hypercube-Labeling Algorithm, *The Computer Journal*, Vol. 37, pp. 124–128.

Bhattacharya, B. K., Jadhav, S., Mukhopadhyay, A. and Robert, J. M. (1994): Optimal Algorithms for Some Intersection Radius Problems, *Computing*, Vol. 52, No. 3, pp. 269–279.

Blankenagel, G. and Gueting, R. H. (1990): Internal and External Algorithms for the Points-in-Regions Problem, *Algorithmica*, Vol. 5, No. 2, pp. 251–276.

Blazewicz, J. and Kasprzak, M. (2003): Complexity of DNA Sequencing by Hybridization, *Theoretical Computer Science*, Vol. 290, No. 3, pp. 1459–1473.

Blot, J., Fernandez de la Vega, W., Paschos, V. T. and Saad, R. (1995): Average Case Analysis of Greedy Algorithms for Optimization Problems on Set Systems, *Theoretical Computer Science*, Vol. 147, No. 1–2, pp. 267–298.

Blum, A. (1994): New Approximation Algorithms for Graph Coloring, *Journal of the ACM*, Vol. 41, No. 3.

Blum, A., Jiang, T., Li, M., Tromp, J. and Yannakakis, M. (1994): Linear Approximation of Shortest Superstrings, *Journal of the ACM*, Vol. 41, pp. 630–647.

Blum, A., Sandholm, T. and Zinkevich, M. (2002): Online Algorithms for Market Clearing, *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, San Francisco, California, pp. 971–980.

Blum, M., Floyd, R. W., Pratt, V. R., Rivest, R. L. and Tarjan, R. E. (1972): Time Bounds for Selection, *Journal of Computer and System Sciences*, Vol. 7, No. 4, pp. 448–461.

Bodlaender, H. L. (1988): The Complexity of Finding Uniform Emulations on Fixed Graphs, *Information Processing Letters*, Vol. 29, No. 3, pp. 137–141.

Bodlaender, H. L. (1993): Complexity of Path-Forming Games, *Theoretical Computer Science*, Vol. 110, No. 1, pp. 215–245.

Bodlaender, H. L., Downey, R. G., Fellows, M. R. and Wareham, H. T. (1995): The Parameterized Complexity of Sequence Alignment and Consensus, *Theoretical Computer Science*, Vol. 147, pp. 31–54.

Bodlaender, H. L., Fellows, M. R. and Hallet, M. T. (1994): Beyond NP-

Completeness for Problems of Bounded Width: Hardness for the W Hierarchy (Extended Abstract), *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, ACM Press, New York, pp. 449–458.

Boffey, T. B. and Green, J. R. (1983): Design of Electricity Supply Networks, *Discrete Applied Mathematics*, Vol. 5, pp. 25–38.

Boldi, P. and Vigna, S. (1999): Complexity of Deciding Sense of Direction, *SIAM Journal on Computing*, Vol. 29, No. 3, pp. 779–789.

Bonizzoni, P. and Vedova, G. D. (2001): The Complexity of Multiple Sequence Alignment with SP-Score that is a Metric, *Theoretical Computer Science*, Vol. 259, pp. 63–79.

Bonizzoni, P., Vedova, G. D. and Mauri, G. (2001): Experimenting an Approximation Algorithm for the LCS, *Discrete Applied Mathematics*, Vol. 110, No. 1, pp. 13–24.

Boppana, R. B., Hastad, J. and Zachos, S. (1987): Does Co-NO Have Short Interactive Proofs? *Information Processing Letters*, Vol. 25, No. 2, pp. 127–132.

Boreale, M. and Trevisan, L. (2000): A Complexity Analysis of Bisimilarity for Value-Passing Processes, *Theoretical Computer Science*, Vol. 238, No. 1, pp. 313–345.

Borodin, A. and El-Yaniv, R. (1998): *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, England.

Borodin, A., Linial, N. and Saks, M. (1992): An Optimal On-line Algorithm for Metrical Task System, *Journal of the ACM*, Vol. 39, No. 4, pp. 745–763.

Boros, E., Crama, Y., Hammer, P. L. and Saks, M. (1994): A Complexity Index for Satisfiability Problems, *SIAM Journal on Computing*, Vol. 23, No. 1, pp. 45–49.

Boruvka, O. (1926): O Jistem Problemu Minimalmim. *Praca Moravske Prirodovedecke Spoleconosti*, Vol. 3, pp. 37–58.

Bossi, A., Cocco, N. and Colussi, L. (1983): A Divide-and-Conquer Approach to General Context-Free Parsing, *Information Processing Letters*, Vol. 16, No. 4, pp. 203–208.

Brassard, G. and Bratley, P. (1988): *Algorithmics: Theory and Practice*, Prentice-Hall, Englewood Cliffs, New Jersey.

Breen, S., Waterman, M. S. and Zhang, N. (1985): Renewal Theory for Several Patterns, *Journal of Applied Probability*, Vol. 22, pp. 228–234.

Breslauer, D., Jiang, T. and Jiang, Z. J. (1997): Rotations of Periodic Strings and Short Superstrings (2.596 – Approximation), *Algorithms*, Vol. 24, No. 2, pp. 340–353.

Bridson, R. and Tang, W. P. (2001): Multiresolution Approximate Inverse Preconditioners, *SIAM Journal on Scientific Computing*, Vol. 23, pp. 463–479.

Brigham, E. O. (1974): *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, New Jersey.

Brown, C. A. and Purdom, P. W. Jr. (1981): An Average Time Analysis of Backtracking, *SIAM Journal on Computing*, Vol. 10, pp. 583–593.

Brown, K. Q. (1979): Voronoi Diagrams from Convex Hulls, *Information Processing Letters*, Vol. 9, pp. 223–228.

Brown, M. L. and Whitney, D. E. (1994): Stochastic Dynamic Programming Applied to Planning of Robot Grinding Tasks, *IEEE Transactions on Robotics and Automation*, pp. 594–604.

Brown, M. R. and Tarjan, R. E. (1980): Design and Analysis of a Data Structure for Representing Sorted Lists, *SIAM Journal on Computing*, Vol. 9, No. 3, pp. 594–614.

Bruno, J., Coffman, E. G. Jr. and Sethi, R. (1974): Scheduling Independent Tasks to Reduce Mean Finishing Time, *Communications of the ACM*, Vol. 17, No. 7, pp. 382–387.

Bryant, D. (1998): The Complexity of the Breakpoint Median Problem, *Technical*

Report CRM-2579, pp. 1–12.

Caballero-Gil, P. (2000): New Upper Bounds on the Linear Complexity, *Computers and Mathematics with Applications*, Vol. 39, No. 3, pp. 31–38.

Cai, J. Y. and Meyer, G. E. (1987): Graph Minimal Uncolorability Is DP-Complete, *SIAM Journal on Computing*, Vol. 16, No. 2, pp. 259–277.

Caprara, A. (1997a): Sorting by Reversals Is Difficult, *Proceedings of the First Annual International Conference on Computational Molecular Biology*, ACM Press, Santa Fe, New Mexico, pp. 75–83.

Caprara, A. (1997b): Sorting Permutations by Reversals and Eulerian Cycle Decompositions, *SIAM Journal on Discrete Mathematics*, pp. 1–23.

Caprara, A. (1999): Formulations and Hardness of Multiple Sorting by Reversals, *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology*, ACM Press, Lyon, France, pp. 84–93.

Caragiannis, I., Kaklamanis, C. and Papaioannou, E. (2003): Simple On-Line Algorithms for Call Control in Cellular Networks, *Lecture Notes in Computer Science*, Vol. 2909, pp. 67–80.

Cary, M. (2001): Toward Optimal ε-Approximate Nearest Neighbor Algorithms, *Journal of Algorithms*, Vol. 41, pp. 417–428.

Chan, K. F. and Lam, T. W. (1993): An On-Line Algorithm for Navigating in Unknown Environment, *International Journal of Computational Geometry and Applications*, Vol. 3, No. 3, pp. 227–244.

Chan, T. (1998): Deterministic Algorithms for 2-D Convex Programming and 3-D Online Linear Programming, *Journal of Algorithms*, Vol. 27, pp. 147–166.

Chandra, B. and Vishwanathan, S. (1995): Constructing Reliable Communication Networks of Small Weight On-Line, *Journal of Algorithms*, Vol. 18, pp. 159–175.

Chang, C. L. and Lee, R. C. T. (1973): *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York.

Chang, K. C. and Du, H. C. (1988): Layer Assignment Problem for Three-Layer Routing, *IEEE Transactions on Computers*, Vol. 37, pp. 625–632.

Chang, W. I. and Lampe, J. (1992): Theoretical and Empirical Comparisons of Approximate String Matching Algorithms, *Lecture Notes in Computer Science*, Vol. 644, pp. 172–181.

Chang, W. I. and Lawler, E. L. (1994): Sublinear Approximate String Matching and Biological Applications, *Algorithmica*, Vol. 12, No. 4–5, pp. 327–344.

Chao, H. S. (1992): *On-line algorithms for three computational geometry problems*. Unpublished Ph.D. thesis, National Tsing Hua University, Hsinchu, Taiwan.

Chao, M. T. (1985): *Probabilistic analysis and performance measurement of algorithms for the satisfiability problem*. Unpublished Ph.D. thesis, Case Western Reserve University, Cleveland, Ohio.

Chao, M. T. and Franco, J. (1986): Probabilistic Analysis of Two Heuristics for the 3-Satisfiability Problem, *SIAM Journal on Computing*, Vol. 15, No. 4, pp. 1106–1118.

Charalambous, C. (1997): Partially Observable Nonlinear Risk-Sensitive Control Problems: Dynamic Programming and Verification Theorems, *IEEE Transactions on Automatic Control*, Vol. 42, pp. 1130–1138.

Chazelle, B., Drysdale, R. L. and Lee, D. T. (1986): Computing the Largest Empty Rectangle, *SIAM Journal on Computing*, Vol. 15, No. 1, pp. 300–315.

Chazelle, B., Edelsbrunner, H., Guibas, L., Sharir, M. and Snoeyink, J. (1993): Computing a Face in an Arrangement of Line Segments and Related Problems, *SIAM Journal on Computing*, Vol. 22, No. 6, pp. 1286–1302.

Chazelle, B. and Matousek, J. (1996): On Linear-Time Deterministic Algorithms for Optimization Problems in Fixed Dimension, *Journal of Algorithm*, Vol. 21, pp. 579–597.

Chekuri, C., Khanna, S. and Zhu, A. (2001): Algorithms for Minimizing

Weighted Flow Time, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, ACM Press, Hersonissos, Greece, pp. 84–93.

Chen, G. H., Chern, M. S. and Jang, J. H. (1990): Pipeline Architectures for Dynamic Programming Algorithms, *Parallel Computing*, Vol. 13, pp. 111–117.

Chen, G. H., Kuo, M. T. and Sheu, J. P. (1988): An Optimal Time Algorithm for Finding a Maximum Weighted Independent Set in a Tree, *BIT*, Vol. 28, pp. 353–356.

Chen, J. and Miranda, A. (2001): A Polynomial Time Approximation Scheme for General Multiprocessor Job Scheduling, *SIAM Journal on Computing*, Vol. 31, pp. 1–17.

Chen, L. H. Y. (1975): Poisson Approximation for Dependent Trials, *The Annals of Probability*, Vol. 3, pp. 534–545.

Chen, T. S., Yang, W. P. and Lee, R. C. T. (1989): Amortized Analysis of Disk Scheduling Algorithms, *Technical Report*, Department of Information Engineering, National Chiao-Tung University, Taiwan.

Chen, W. M. and Hwang, H. K. (2003): Analysis in Distribution of Two Randomized Algorithms for Finding the Maximum in a Broadcast Communication Model, *Journal of Algorithms*, Vol. 46, pp. 140–177.

Cheriyan, J. and Harerup, T. (1995): Randomized Maximum-Flow Algorithm, *SIAM Journal on Computing*, Vol. 24, No. 2, pp. 203–226.

Chin, W. and Ntafos, S. (1988): Optimum Watchman Routes, *Information Processing Letters*, Vol. 28, No. 1, pp. 39–44.

Chou, H. C. and Chung C. P. (1994): Optimal Multiprocessor Task Scheduling Using Dominance and Equivalence Relations, *Computer & Operations Research*, Vol. 21, No. 4, pp. 463–475.

Choukhmane, E. and Franco, J. (1986): An Approximation Algorithm for the Maximum Independent Set Problem in Cubic Planar Graphs, *Networks*, Vol. 16, No. 4, pp. 349–356.

Christofides, N. (1976): Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem, *Management Sciences Research Report*, No. 388.

Christos, L. and Drago, K. (1998): Quasi-Greedy Triangulations Approximating the Minimum Weight Triangulation, *Journal of Algorithms*, Vol. 27, No. 2, pp. 303–338.

Chrobak, M. and Larmore, L. L. (1991): An Optimal On-Line Algorithm for k-Servers on Trees, *SIAM Journal of Computing*, Vol. 20, No. 1, pp. 144–148.

Chu, C. and La, R. (2001): Variable-Sized Bin Packing: Tight Absolute Worst-Case Performance Ratios for Four Approximation Algorithms, *SIAM Journal on Computing*, Vol. 30, pp. 2069–2083.

Chung, M. J. and Krishnamoorthy, M. S. (1988): Algorithms of Placing Recovery Points, *Information Processing Letters*, Vol. 28, No. 4, pp. 177–181.

Chung, M. J., Makedon, F., Sudborough, I. H. and Turner, J. (1985): Polynomial Time Algorithms for the Min Cut Problem on Degree Restricted Trees, *SIAM Journal on Computing*, Vol. 14, No. 1, pp. 158–177.

Cidon, I., Kutten, S., Mansour, Y. and Peleg, D. (1995): Greedy Packet Scheduling, *SIAM Journal on Computing*, Vol. 24, pp. 148–157.

Clarkson, K. L. (1987): New Applications of Random Sampling to Computational Geometry, *Discrete and Computational Geometry*, Vol. 2, pp. 195–222.

Clarkson, K. L. (1988): A Randomized Algorithm for Closest-Point Queries, *SIAM Journal on Computing*, Vol. 17, No. 4, pp. 830–847.

Clarkson, K. L. (1994): An Algorithm for Approximate Closest-Point Queries, *Proceedings of the 10th Annual Symposium on Computational Geometry*, ACM Press, Stony Brook, New York, pp. 160–164.

Cobbs, A. (1995): Fast Approximate Matching Using Suffix Trees, *Lecture Notes*

in Computer Science, Vol. 937, pp. 41–54.

Coffman, E. G., Langston, J. and Langston, M. A. (1984): A Performance Guarantee for the Greedy Set-Partitioning Algorithm, *Acta Informatica*, Vol. 21, pp. 409–415.

Coffman, E. G. and Lueker, G. S. (1991): *Probabilistic Analysis of Packaging & Partitioning Algorithms*, John Wiley & Sons, New York.

Colbourn, C. J., Kocay, W. L. and Stinson, D. R. (1986): Some NP-Complete Problems for Hypergraph Degree Sequences, *Discrete Applied Mathematics*, Vol. 14, No. 3, pp. 239–254.

Cole, R. (1994): Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm, *SIAM Journal on Computing*, Vol. 23, No. 5, pp. 1075–1091.

Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T. and Thorup, M. (2000): An $O(n \log n)$ Algorithm for the Maximum Agreement Subtree Problem for Binary Trees, *SIAM Journal on Applied Mathematics*, Vol. 30, No. 5, pp. 1385–1404.

Cole, R. and Hariharan, R. (1997): Tighter Upper Bounds on the Exact Complexity of String Matching, *SIAM Journal on Computing*, Vol. 26, No. 3, pp. 803–856.

Cole, R., Hariharan, R., Paterson, M. and Zwick, U. (1995): Tighter Lower Bounds on the Exact Complexity of String Matching, *SIAM Journal on Computing*, Vol. 24, No. 1, pp. 30–45.

Coleman, T. F., Edenbrandt, A. and Gilbert, J. R. (1986): Predicting Fill for Sparse Orthogonal Factorization, *Journal of the ACM*, Vol. 33, No. 3, pp. 517–532.

Conn, R. and Vonholdt, R. (1965): An Online Display for the Study of Approximating Functions, *Journal of the ACM*, Vol. 12, No. 3, pp. 326–349.

Cook, S. A. (1971): The Complexity of Theorem Proving Procedures, *Proceedings of the 3rd ACM Symposium on Theory of Computing*, ACM Press, Shaker Heights, Ohio, pp. 151–158.

Cooley, J. W. and Tukey, J. W. (1965): An Algorithm for the Machine Calculation of Complex Fourier Series, *Mathematics of Computation*, Vol. 19, pp. 297–301.

Coppersmith, D., Doyle, P., Raghavan, P. and Snir, M. (1993): Random Walks on Weighted Graphs and Applications to On-line Algorithms, *Journal of the ACM*, Vol. 40, No. 3, pp. 421–453.

Cormen, T. H. (1999): Determining an Out-of-Core FFT Decomposition Strategy for Parallel Disks by Dynamic Programming, *Algorithms for Parallel Processing*, Vol. 105, pp. 307–320.

Cormen, T. H. (2001): *Introduction to Algorithms*, McGraw-Hill, New York.

Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (1990): *Introduction to Algorithms*, McGraw-Hill, New York.

Cornuejols, C., Fisher, M. L. and Nemhauser, G. L. (1977): Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms, *Management Science*, Vol. 23, No. 8, pp. 789–810.

Cowureur, C. and Bresler, Y. (2000): On the Optimality of the Backward Greedy Algorithm for the Subset Selection Problem, *SIAM Journal on Matrix Analysis and Applications*, Vol. 21, pp. 797–808.

Crammer, K. and Singer, Y. (2002): Text Categorization: A New Family of Online Algorithms for Category Ranking, *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM Press, Tampere, Finland, pp. 151–158.

Crescenzi, P., Goldman, D., Papadimitriou, C., Piccolboni, A. and Yannakakis, M. (1998): On the Complexity of Protein Folding, *Journal of Computational Biology*, Vol. 5, No. 3, pp. 423–465.

Csirik, J. (1989): An On-Line Algorithm For Variable-Sized Bin Packing, *Acta Informatica*, Vol. 26, pp. 697–709.

Csur, M. and Kao, M. Y. (2001): Provably Fast and Accurate Recovery of Evolutionary Trees through Harmonic Greedy Triplets, *SIAM Journal on Computing*, Vol. 31, pp. 306–322.

Culberson, J. C. and Rudnicki, P. (1989): A Fast Algorithm for Constructing Trees from Distance Matrices, *Information Processing Letters*, Vol. 30, No. 4, pp. 215–220.

Cunningham, W. H. (1985): Optimal Attack and Reinforcement of a Network, *Journal of the ACM*, Vol. 32, No. 3, pp. 549–561.

Czumaj, A., Gasieniec, L., Piotrow, M. and Rytter, W. (1994): Parallel and Sequential Approximation of Shortest Superstrings, *Lecture Notes in Computer Science*, Vol. 824, pp. 95–106.

d'Amore, F. and Liberatore, V. (1994): List Update Problem and the Retrieval of Sets, *Theoretical Computer Science*, Vol. 130, No. 1, pp. 101–123.

Darve, E. (2000): The Fast Multipole Method I: Error Analysis and Asymptotic Complexity, *SIAM Journal on Numerical Analysis*, Vol. 38, No. 1, pp. 98–128.

Day, W. H. (1987): Computational Complexity of Inferring Phylogenies from Dissimilarity Matrices, *Bulletin of Mathematical Biology*, Vol. 49, No. 4, pp. 461–467.

Decatur, S. E., Goldreich, O. and Ron, D. (1999): Computational Sample Complexity, *SIAM Journal on Computing*, Vol. 29, No. 3.

Dechter, R. and Pearl, J. (1985): Generalized Best-First Search Strategies and the Optimality of A*, *Journal of the ACM*, Vol. 32, No. 3, pp. 505–536.

Delcoigne, A. and Hansen, P. (1975): Sequence Comparison by Dynamic Programming, *Biometrika*, Vol. 62, pp. 661–664.

Denardo, E. V. (1982): *Dynamic Programming: Model and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey.

Deng, X. and Mahajan, S. (1991): Infinite Games: Randomization Computability and Applications to Online Problems, *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, ACM Press, New Orleans, Louisiana, pp. 289–298.

Derfel, G. and Vogl, F. (2000): Divide-and-Conquer Recurrences: Classification of Asymptotics, *Aequationes Mathematicae*, Vol. 60, pp. 243–257.

Devroye, L. (2002): Limit Laws for Sums of Functions of Subtrees of Random Binary Search Trees, *SIAM Journal on Applied Mathematics*, Vol. 32, No. 1, pp. 152–171.

Devroye, L. and Robson, J. M. (1995): On the Generation of Random Binary Search Trees, *SIAM Journal on Applied Mathematics*, Vol. 24, No. 6, pp. 1141–1156.

Dietterich, T. G. (2000): The Divide-and-Conquer Manifesto, *Lecture Notes in Artificial Intelligence*, Vol. 1968, pp. 13–26.

Dijkstra, E. W. (1959): A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik*, Vol. 1, pp. 269–271.

Dinitz, Y. and Nutov, Z. (1999): A 3-Approximation Algorithm for Finding Optimum 4, 5-Vertex-Connected Spanning Subgraphs, *Journal of Algorithms*, Vol. 32, pp. 31–40.

Dixon, B., Rauch, M. and Tarjan, R. E. (1992): Verification and Sensitivity Analysis of Minimum Spanning Trees in Linear Time, *SIAM Journal on Computing*, Vol. 21, pp. 1184–1192.

Dobkin, D. P. and Lipton, R. J. (1979): On the Complexity of Computations Under Varying Sets of Primitives, *Journal of Computer and System Sciences*, Vol. 18, pp. 86–91.

Dolan, A. and Aldus, J. (1993): *Networks and Algorithms: An Introductory Approach*, John Wiley & Sons, New York.

Dolev, D., Lynch, N. A., Pinter, S. S., Stark, E. W. and William, E. W. (1986): Reaching Approximate Agreement in the Presence of Faults, *Journal of the ACM*, Vol. 33, No. 3, pp. 499–516.

Drake, D. E. and Hougardy, S. (2003): A Simple Approximation Algorithm for the Weighted Matching Problem, *Information Processing Letters*, Vol. 85, pp. 211–213.

Dreyfus, S. E. and Law, A. M. (1977): *The Art and Theory of Dynamic Programming*, Academic Press, London.

Du, D. Z. and Book, R. V. (1989): On Inefficient Special Cases of NP-Complete Problems, *Theoretical Computer Science*, Vol. 63, No. 3, pp. 239–252.

Du, J. and Leung, J. Y. T. (1988): Scheduling Tree-Structured Tasks with Restricted Execution Times, *Information Processing Letters*, Vol. 28, No. 4, pp. 183–188.

Dwyer, R. A. (1987): A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations, *Algorithmics*, Vol. 2, pp. 137–151.

Dyer, M. E. (1984): Linear Time Algorithm for Two- and Three-Variable Linear Programs, *SIAM Journal on Computing*, Vol. 13, No. 1, pp. 31–45.

Dyer, M. E. and Frieze, A. M. (1989): Randomized Algorithm for Fixed-Dimensional Linear Programming, *Mathematical Programming*, Vol. 44, No. 2, pp. 203–212.

Edelsbrunner, H. (1987): *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin.

Edelsbrunner, H. and Guibas, L. J. (1989): Topologically Sweeping an Arrangement, *Journal of Computer and System Sciences*, Vol. 38, No. 1, pp. 165–194.

Edelsbrunner, H., Maurer, H. A., Preparata, F. P., Rosenberg, A. L., Welzl, E. and Wood, D. (1982): Stabbing Line Segments, *BIT*, Vol. 22, pp. 274–281.

Edwards, C. S. and Elphick, C. H. (1983): Lower Bounds for the Clique and the Chromatic Numbers of a Graph, *Discrete Applied Mathematics*, Vol. 5, No. 1, pp. 51–64.

Eiter, T. and Veith, H. (2002): On the Complexity of Data Disjunctions, *Theoretical Computer Science*, Vol. 288, No. 1, pp. 101–128.

Ekroot, L. and Dolinar, S. (1996): A* Decoding of Block Codes, *IEEE Transactions on Communications*, pp. 1052–1056.

ElGindy, H., Everett, H. and Toussaint, G. (1993): Slicing an Ear Using Prune-and-Search, *Pattern Recognition Letters*, Vol. 14, pp. 719–722.

El-Yaniv, R. (1998): Competitive Solutions for Online Financial Problems, *ACM Computing Surveys*, Vol. 30, No. 1, pp. 28–69.

El-Zahar, M. H. and Rival, I. (1985): Greedy Linear Extensions to Minimize Jumps, *Discrete Applied Mathematics*, Vol. 11, No. 2, pp. 143–156.

Eppstein, D., Galil, Z., Giancarlo, R. and Italiano, G. F. (1992a): Sparse Dynamic Programming I: Linear Cost Functions, *Journal of the ACM*, Vol. 39, No. 3, pp. 519–545.

Eppstein, D., Galil, Z., Giancarlo, R. and Italiano, G. F. (1992b): Sparse Dynamic Programming II: Convex and Concave Cost Functions, *Journal of the ACM*, Vol. 39, pp. 516–567.

Eppstein, D., Galil, Z., Italiano, G. F. and Spencer, T. H. (1996): Separator Based Sparsication I. Planarity Testing and Minimum Spanning Trees, *Journal of Computer and System Sciences*, Vol. 52, No. 1, pp. 3–27.

Epstein, L. and Ganot, A. (2003): Optimal On-line Algorithms to Minimize Makespan on Two Machines with Resource Augmentation, *Lecture Notes in Computer Science*, Vol. 2909, pp. 109–122.

Epstein, L., Noga, J., Seiden, S., Sgall, J. and Woeginger, G. (1999): Randomized Online Scheduling on Two Uniform Machines, *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Baltimore,

Maryland, pp. 317–326.

Erdmann, M. (1993): Randomization for Robot Tasks: Using Dynamic Programming in the Space of Knowledge States, *Algorithmica*, Vol. 10, pp. 248–291.

Erlebach, T. and Jansen, K. (1999): Efficient Implementation of an Optimal Greedy Algorithm for Wavelength Assignment in Directed Tree Networks, *ACM Journal of Experimental Algorithmics*, Vol. 4.

Esko, U. (1990): A Linear Time Algorithm for Finding Approximate Shortest Common Superstrings, *Algorithmica*, Vol. 5, pp. 313–323.

Evans, J. R. and Minieka, E. (1992): *Optimization Algorithms for Networks and Graphs*, 2nd ed., Marcel Dekker, New York.

Even, G., Naor, J., Rao, S. and Schieber, B. (2000): Divide-and-Conquer Approximation Algorithms via Spreading Metrics, *Journal of the ACM*, Vol. 47, No. 4, pp. 585–616.

Even, G., Naor, J. and Zosin, L. (2000): An 8-Approximation Algorithm for the Subset Feedback Vertex Set Problem, *SIAM Journal on Computing*, Vol. 30, pp. 1231–1252.

Even, S. (1987): *Graph Algorithms*, Computer Science Press, Rockville, Maryland.

Even, S., Itai, A. and Shamir, A. (1976): On the Complexity of Timetable and Multicommodity Problems, *SIAM Journal on Computing*, Vol. 5, pp. 691–703.

Even, S. and Shiloach, Y. (1981): An On-Line Edge-Deletion Problem, *Journal of the ACM*, Vol. 28, No. 1, pp. 1–4.

Fagin, R. (1974): Generalized First-Order Spectra and Polynomial-Time Recognizable Sets, *SIAM-AMS Proceedings*, Vol. 7, pp. 43–73.

Faigle, U. (1985): On Ordered Languages and the Optimization of Linear Functions by Greedy Algorithms, *Journal of the ACM*, Vol. 32, No. 4, pp. 861–870.

Faigle, U., Kern, W. and Nawijn, W. (1999): A Greedy On-Line Algorithm for the k-Track Assignment Problem, *Journal of Algorithms*, Vol. 31, pp. 196–210.

Farach, M. and Thorup, M. (1997): Sparse Dynamic Programming for Evolutionary Tree Comparison, *SIAM Journal on Computing*, Vol. 26, pp. 210–230.

Farber, M. and Keil, J. M. (1985): Domination in Permutation Graphs, *Journal of Algorithms*, Vol. 6, pp. 309–321.

Feder, T. and Motwani, R. (2002): Worst-Case Time Bounds for Coloring and Satisfiability Problems, *Journal of Algorithms*, Vol. 45, pp. 192–201.

Feige, U. and Krauthgamer, R. (2002): A Polylogarithmic Approximation of the Minimum Bisection, *SIAM Journal on Computing*, Vol. 31, No. 4, pp. 1090–1118.

Fejes, G. (1978): Evading Convex Discs, *Studia Science Mathematics Hungar*, Vol. 13, pp. 453–461.

Feldmann, A., Kao, M., Sgall, J. and Teng, S. H. (1993): Optimal Online Scheduling of Parallel Jobs with Dependencies, *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, ACM Press, San Diego, California, pp. 642–651.

Fellows, M. R. and Langston, M. A. (1988): Processor Utilization in a Linearly Connected Parallel Processing System, *IEEE Transactions on Computers*, Vol. 37, pp. 594–603.

Fernandez-Beca, D. and Williams, M. A. (1991): On Matroids and Hierarchical Graphs, *Information Processing Letters*, Vol. 38, No. 3, pp. 117–121.

Ferraginá, P. (1997): Dynamic Text Indexing under String Updates, *Journal of Algorithms*, Vol. 22, No. 2, pp. 296–238.

Ferreira, C. E., de Souza, C. C. and Wakabayashi, Y. (2002): Rearrangement of DNA Fragments: A Branch-and-Cut Algorithm (Prove NP-Complete Problem), *Discrete Applied Mathematics*, Vol. 116, pp. 161–177.

Fiat, A. and Woeginger, G. J. (1998): Online Algorithms: The State of the Art, *Lecture Notes in Computer Science*, Vol. 1442.

Fischel-Ghodsian, F., Mathiowitz, G. and Smith, T. F. (1990): Alignment of Protein Sequence Using Secondary Structure: A Modified Dynamic Programming Method, *Protein Engineering*, Vol. 3, No. 7, pp. 577–581.

Flajolet, P. and Prodinger, H. (1986): Register Allocation for Unary-Binary Trees, *SIAM Journal on Computing*, Vol. 15, No. 3, pp. 629–640.

Floyd, R. W. (1962): Algorithm 97: Shortest Path, *Communications of the ACM*, Vol. 5, No. 6, p. 345.

Floyd, R. W. and Rivest, R. L. (1975): Algorithm 489 (SELECT), *Communications of the ACM*, Vol. 18, No. 3, p. 173.

Fonlupt, J. and Nachef, A. (1993): Dynamic Programming and the Graphical Traveling Salesman Problem, *Journal of the Association for Computing Machinery*, Vol. 40, No. 5, pp. 1165–1187.

Foulds, L. R. and Graham, R. L. (1982): The Steiner Problem in Phylogeny is NP-Complete, *Advances Application Mathematics*, Vol. 3, pp. 43–49.

Franco, J. (1984): Probabilistic Analysis of the Pure Literal Heuristic for the Satisfiability Problem, *Annals of Operations Research*, Vol. 1, pp. 273–289.

Frederickson, G. N. (1984): Recursively Rotated Orders and Implicit Data Structures: A Lower Bound, *Theoretical Computer Science*, Vol. 29, pp. 75–85.

Fredman, M. L. (1981): A Lower Bound on the Complexity of Orthogonal Range Queries, *Journal of the ACM*, Vol. 28, No. 4, pp. 696–705.

Fredman, M. L., Sedgewick, R., Sleator, D. D. and Tarjan, R. E. (1986): The Pairing Heap: A New Form of Self-Adjusting Heap, *Algorithmica*, Vol. 1, No. 1, pp. 111–129.

Friesen, D. K. and Kuhl, F. S. (1988): Analysis of a Hybrid Algorithm for Packing Unequal Bins, *SIAM Journal on Computing*, Vol. 17, No. 1, pp. 23–40.

Friesen, D. K. and Langston, M. A. (1986): Variable Sized Bin Packing, *SIAM Journal on Computing*, Vol. 15, No. 1, pp. 222–230.

Frieze, A. M. and Kannan, R. (1991): A Random Polynomial-Time Algorithm for Approximating the Volume of Convex Bodies, *Journal of the ACM*, Vol. 38, No. 1.

Frieze, A. M., McDiarmid, C. and Reed, B. (1990): Greedy Matching on the Line, *SIAM Journal on Computing*, Vol. 19, No. 4, pp. 666–672.

Froda, S. (2000): On Assessing the Performance of Randomized Algorithms, *Journal of Algorithms*, Vol. 31, pp. 344–362.

Fu, H. C. (2001): Divide-and-Conquer Learning and Modular Perceptron Networks, *IEEE Transactions on Neural Networks*, Vol. 12, No. 2, pp. 250–263.

Fu, J. J. and Lee, R. C. T. (1991): Minimum Spanning Trees of Moving Points in the Plane, *IEEE Transactions on Computers*, Vol. 40, No. 1, pp. 113–118.

Galbiati, G., Maffioli, F. and Morrzenti, A. (1994): A Short Note on the Approximability of the Maximum Leaves Spanning Tree Problem, *Information Processing Letters*, Vol. 52, pp. 45–49.

Galil, Z. and Giancarlo, R. (1989): Speeding up Dynamic Programming with Applications to Molecular Biology, *Theoretical Computer Science*, Vol. 64, pp. 107–118.

Galil, Z., Haber, S. and Yung, M. (1989): Minimum-Knowledge Interactive Proofs for Decision Problems, *SIAM Journal on Computing*, Vol. 18, No. 4, pp. 711–739.

Galil, Z., Hoffman, C. H., Luks, E. M., Schnorr, C. P. and Weber, A. (1987): An $O(n^3 \log n)$ Deterministic and an $O(n^3)$ Las Vagas Isomorphism Test for Trivalent Graphs, *Journal of the ACM*, Vol. 34, No. 3, pp. 513–531.

Galil, Z. and Park, K. (1990): An Improved Algorithm for Approximate String Matching, *SIAM Journal on Computing*, Vol. 19, pp. 989–999.

Galil, Z. and Seiferas, J. (1978): A Linear-Time On-Line Recognition Algorithm

for "Palstar", *Journal of the ACM*, Vol. 25, No. 1, pp. 102–111.

Galil, Z. and Park, K. (1992): Dynamic Programming with Convexity Concavity and Sparsity, *Theoretical Computer Science*, Vol. 49–76.

Gallant, J., Marier, D. and Storer, J. A. (1980): On Finding Minimal Length Superstrings (Prove NP-Hard Problem), *Journal of Computer and System Sciences*, Vol. 20, pp. 50–58.

Garay, J., Gopal, I., Kutten, S., Mansour, Y. and Yung, M. (1997): Efficient On-Line Call Control Algorithms, *Journal of Algorithms*, Vol. 23, pp. 180–194.

Garey, M. R. and Johnson, D. S. (1976): Approximation algorithms for combinatorial problem: An annotated bibliography. In J. F. Traub (Ed.), *Algorithms and Complexity: New Directions and Recent Results*, Academic Press, New York, pp. 41–52.

Garey, M. R. and Johnson, D. S. (1979): *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, California.

Geiger, D., Gupta, A., Costa, L. A. and Vlontzos, J. (1995): Dynamic Programming for Detecting Tracking and Matching Deformable Contours, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, pp. 294–302.

Gelfand, M. S. and Roytberg, M. A. (1993): Prediction of the Exon-Intron Structure by a Dynamic Programming Approach, *BioSystems*, Vol. 30, pp. 173–182.

Gentleman, W. M. and Sande, G. (1966): Fast Fourier Transforms for Fun and Prot, *Proceedings of AFIPS Fall Joint Computer Conference*, Vol. 29, pp. 563–578.

Giancarlo, R. and Grossi, R. (1997): Multi-Dimensional Pattern Matching with Dimensional Wildcards: Data Structures and Optimal On-Line Search Algorithms, *Journal of Algorithms*, Vol. 24, pp. 223–265.

Gilbert, E. N. and Moore, E. F. (1959): Variable Length Encodings, *Bell System Technical Journal*, Vol. 38, No. 4, pp. 933–968.

Gilbert, J. R., Hutchinson, J. P. and Tarjan, R. E. (1984): A Separator Theorem for Graphs of Bounded Genus, *Journal of Algorithms*, Vol. 5, pp. 391–407.

Gill, I. (1987): Computational Complexity of Probabilistic Turing Machine, *SIAM Journal on Computing*, Vol. 16, No. 5, pp. 852–853.

Godbole, S. S. (1973): On Efficient Computation of Matrix Chain Products, *IEEE Transactions on Computers*, Vol. 22, No. 9, pp. 864–866.

Goemans, M. X. and Williamson, D. P. (1995): Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming, *Journal of the ACM*, Vol. 42, No. 6, pp. 1115–1145.

Goldberg, L. A., Goldberg, P. W. and Paterson, M. (2001): The Complexity of Gene Placement (Prove NP-Complete Problem), *Journal of Algorithms*, Vol. 41, pp. 225–243.

Goldman, S., Parwatikar, J. and Suri, S. (2000): Online Scheduling with Hard Deadlines, *Journal of Algorithms*, Vol. 34, pp. 370–389.

Goldstein, L. and Waterman, M. S. (1987): Mapping DNA by Stochastic Relaxation (Prove NP-Complete Problem of Double Digest Problem), *Advances in Applied Mathematics*, Vol. 8, pp. 194–207.

Goldwasser, S. and Micali, S. (1984): Probabilistic Encryption, *Journal of Computer and System Sciences*, Vol. 28, No. 2, pp. 270–298.

Goldwasser, S., Micali, S. and Rackoff, C. (1988): The Knowledge Complexity of Interactive Proof Systems, *SIAM Journal on Computing*, Vol. 18, No. 1, pp. 186–208.

Golumbic, M. C. (1980): *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York.

Gonnet, G. H. (1983): *Handbook of Algorithms and Data Structures*, Addison-Wesley, Reading, Mass.

Gonzalez, T. F. and Lee, S. L. (1987): A 1.6 Approximation Algorithm for Routing Multiterminal Nets, *SIAM Journal on Computing*, Vol. 16, No. 4, pp. 669–704.

Gonzalo, N. (2001): A Guide Tour to Approximate String Matching, *ACM*, Vol. 33, pp. 31–88.

Goodman, S. and Hedetniemi, S. (1980): *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York.

Gorodkin, J., Lyngso, R. B. and Stormo, G. D. (2001): A Mini-Greedy Algorithm for Faster Structural RNA Stem-Loop Search, *Genome Informatics*, Vol. 12, pp. 184–193.

Gotieb, L. (1981): Optimal Multi-Way Search Trees, *SIAM Journal on Computing*, Vol. 10, No. 3, pp. 422–433.

Gotieb, L. and Wood, D. (1981): The Construction of Optimal Multiway Search Trees and the Monotonicity Principle, *International Journal of Computer Mathematics*, Vol. 9, pp. 17–24.

Gould, R. (1988): *Graph Theory*, Benjamin Cummings, Redwood City, California.

Graham, R. L. (1972): An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set, *Information Processing Letters*, Vol. 1, pp. 132–133.

Grandjean, E. (1988): A Natural NP-Complete Problem with a Nontrivial Lower Bound, *SIAM Journal on Computing*, Vol. 17, No. 4, pp. 786–809.

Greene, D. H. and Knuth, D. E. (1981): *Mathematics for the Analysis of Algorithms*, Birkhauser, Boston, Mass.

Grigni, M., Koutsoupias, E. and Papadimitriou, C. (1995): Approximation Scheme for Planar Graph TSP, *Foundations of Computer Science*, pp. 640–645.

Grove, E. (1995): Online Bin Packing with Lookahead, *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, San Francisco, California, pp. 430–436.

Gudmundsson, J., Levcopoulos, C. and Narasimhan, G. (2002): Fast Greedy Algorithms for Constructing Sparse Geometric Spanners, *SIAM Journal on Computing*, Vol. 31, pp. 1479–1500.

Gueting, R. H. (1984): Optimal Divide-and-Conquer to Compute Measure and Contour for a Set of Iso-Rectangles, *Acta Informatica*, Vol. 21, pp. 271–291.

Gueting, R. H. and Schilling, W. (1987): Practical Divide-and-Conquer Algorithm for the Rectangle Intersection Problem, *Information Sciences*, Vol. 42, No. 2, pp. 95–112.

Gupta, S., Konjevod, G. and Varsamopoulos, G. (2002): A Theoretical Study of Optimization Techniques Used in Registration Area Based Location Management: Models and Online Algorithms, *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, ACM Press, Atlanta, Georgia, pp. 72–79.

Gusfield, D. (1994): Faster Implementation of a Shortest Superstring Approximation, *Information Processing Letters*, Vol. 51, pp. 271–274.

Gusfield, D. (1997): *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, Cambridge, England.

Guting, R. H. (1984): Optimal Divide-and-Conquer to Compute Measure and Contour for a Set of Iso-Rectangles, *Acta Informatica*, Vol. 21, pp. 271–291.

Haas, P. and Hellerstein, J. (1999): Ripple Joins for Online Aggregation, ACM SIGMOD Record, *Proceedings of the 1999 ACM-SIGMOD International Conference on Management of Data*, Vol. 28.

Hall, N. G. and Hochbaum, D. S. (1986): A Fast Approximation Algorithm for the Multicovering Problem, *Discrete Applied Mathematics*, Vol. 15, No. 1, pp. 35–40.

Halldorsson, M. (1997): Parallel and On-Line Graph Coloring, *Journal of*

Algorithms, Vol. 23, pp. 265–280.

Halperin, D., Sharir, M. and Goldberg, K. (2002): The 2-Center Problem with Obstacles, *Journal of Algorithms*, Vol. 42, pp. 109–134.

Han, Y. S., Hartmann, C. R. P. and Chen, C. C. (1993): Efficient Priority: First Search Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes, *IEEE Transactions on Information Theory*, pp. 1514–1523.

Han, Y. S., Hartmann C. R. P. and. Mehrotra, K. G. (1998): Decoding Linear Block Codes Using a Priority-First Search: Performance Analysis and Suboptimal Version, *IEEE Transactions on Information Theory*, pp. 1233–1246.

Hanson, F. B. (1991): Computational Dynamic Programming on a Vector Multiprocessor, *IEEE Transactions on Automatic Control*, Vol. 36, pp. 507–511.

Hariri, A. M. A. and Potts, C. N. (1983): An Algorithm for Single Machine Sequencing with Release Dates to Minimize Total Weighted Completion Time, *Discrete Applied Mathematics*, Vol. 5, No. 1, pp. 99–109.

Har-Peled, S. (2000): Constructing Planar Cuttings in Theory and Practice, *SIAM Journal on Computing*, Vol. 29, No. 6, pp. 2016–2039.

Hasegawa, M. and Horai, S. (1991): Time of the Deepest Root for Polymorphism in Human Mitochondrial DNA, *Journal of Molecular Evolution*, Vol. 32, pp. 37–42.

Hasham, A. and Sack, J. R. (1987): Bounds for Min-Max Heaps, *BIT*, Vol. 27, pp. 315–323.

Hashimoto, R. F. and Barrera J. (2003): A Greedy Algorithm for Decomposing Convex Structuring Elements, *Journal of Mathematical Imaging and Vision*, Vol. 18, No. 3, pp. 269–286.

Haussmann, U. G. and Suo, W. (1995): Singular Optimal Stochastic Controls. II. Dynamic Programming, *SIAM Journal on Control and Optimization*, Vol. 33, pp. 937–959.

Hayward, R. B. (1987): A Lower Bound for the Optimal Crossing-Free Hamiltonian Cycle Problem, *Discrete and Computational Geometry*, Vol. 2, pp. 327–343.

Hein, J. (1989): An Optimal Algorithm to Reconstruct Trees from Additive Distance Data, *Bulletin of Mathematical Biology*, Vol. 51, pp. 597–603.

Held, M. and Karp, R. M. (1962): A Dynamic Programming Approach to Sequencing Problems, *SIAM Journal on Applied Mathematics*, Vol. 10, pp. 196–210.

Hell, P., Shamir, R. and Sharan, R. (2001): A Fully Dynamic Algorithm for Recognizing and Representing Proper Interval Graphs, *SIAM Journal on Computing*, Vol. 31, pp. 289–305.

Henzinger, M. R. (1995): Fully Dynamic Biconnectivity in Graphs, *Algorithmica*, Vol. 13, No. 6, pp. 503–538.

Hirosawa, M., Hoshida, M., Ishikawa, M. and Toya, T. (1993): MASCOT: Multiple Alignment System for Protein Sequence Based on Three-Way Dynamic Programming, *Computer Applications in the Biosciences*, Vol. 9, pp. 161–167.

Hirschberg, D. S. (1975): A Linear Space Algorithm for Computing Maximal Common Subsequences, *Communications of the ACM*, Vol. 18, No. 6, pp. 341–343.

Hirschberg, D. S. and Larmore, L. L. (1987): The Least Weight Subsequence Problem, *SIAM Journal on Computing*, Vol. 16, No. 4, pp. 628–638.

Hoang, T. M. and Thierauf, T. (2003): The Complexity of the Characteristic and the Minimal Polynomial, *Theoretical Computer Science*, Vol. 1–3, pp. 205–222.

Hoare, C. A. R. (1961): Partition (Algorithm 63), Quicksort (Algorithm 64) and Find (Algorithm 65), *Communications of the ACM*, Vol. 4, No. 7, pp. 321–322.

Hoare, C. A. R. (1962): Quicksort, *Computer Journal*, Vol. 5, No. 1, pp. 10–15.

Hochbaum, D. S. (1997): *Approximation Algorithms for NP-Hard Problems*, PWS Publisher, Boston.

Hochbaum, D. S. and Maass, W. (1987): Fast Approximation Algorithms for a Nonconvex Covering Problem, *Journal of Algorithms*, Vol. 8, No. 3, pp. 305–323.

Hochbaum, D. S. and Shmoys, D. B. (1986): A United Approach to Approximation Algorithms for Bottleneck Problems, *Journal of the ACM*, Vol. 33, No. 3, pp. 533–550.

Hochbaum, D. S. and Shmoys, D. B. (1987): Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results, *Journal of the ACM*, Vol. 34, No. 1, pp. 144–162.

Hoffman, A. J. (1988): On Greedy Algorithms for Series Parallel Graphs, *Mathematical Programming*, Vol. 40, No. 2, pp. 197–204.

Hofri, M. (1987): *Probabilistic Analysis of Algorithms*, Springer-Verlag, New York.

Holmes, I. and Durbin, R. (1998): Dynamic Programming Alignment Accuracy, *Journal of Computational Biology*, Vol. 5, pp. 493–504.

Holyer, I. (1981): The NP-Completeness of Some Edge-Partition Problems, *SIAM Journal on Computing*, Vol. 10, pp. 713–717.

Homer, S. (1986): On Simple and Creative Sets in NP, *Theoretical Computer Science*, Vol. 47, No. 2, pp. 169–180.

Homer, S. and Long, T. J. (1987): Honest Polynomial Degrees and P =? NP, *Theoretical Computer Science*, Vol. 51, No. 3, pp. 265–280.

Horowitz, E. and Sahni, S. (1974): Computing Partitions with Applications to the Knapsack Problem, *Journal of the ACM*, Vol. 21, No. 2, pp. 277–292.

Horowitz, E. and Sahni, S. (1976a): Exact and Approximate Algorithms for Scheduling Nonidentical Processors, *Journal of the ACM*, Vol. 23, No. 2, pp. 317–327.

Horowitz, E. and Sahni, S. (1976b): *Fundamentals of Data Structures*, Computer Science Press, Rockville, Maryland.

Horowitz, E. and Sahni, S. (1978): *Fundamentals of Computer Algorithm*, Computer Science Press, Rockville, Maryland.

Horowitz, E., Sahni, S. and Rajasekaran, S. (1998): *Computer Algorithms*, W. H. Freeman, New York.

Horton, J. D. (1987): A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph, *SIAM Journal on Computing*, Vol. 16, No. 2, pp. 358–366.

Horvath, E. C., Lam, S. and Sethi, R. (1977): A Level Algorithm for Preemptive Scheduling, *Journal of the ACM*, Vol. 24, No. 1, pp. 32–43.

Hsiao, J. Y., Tang, C. Y. and Chang, R. S. (1993): The Summation and Bottleneck Minimization for Single Step Searching on Weighted Graphs, *Information Sciences*, Vol. 74, pp. 1–28.

Hsu, W. L. (1984): Approximation Algorithms for the Assembly Line Crew Scheduling Problem, *Mathematics of Operations Research*, Vol. 9, pp. 376–383.

Hsu, W. L. and Nemhauser, G. L. (1979): Easy and Hard Bottleneck Location Problems, *Discrete Applied Mathematics*, Vol. 1, No. 3, pp. 209–215.

Hu, T. C. and Shing, M. T. (1982): Computation of Matrix Chain Products Part I, *SIAM Journal on Computing*, Vol. 11, No. 2, pp. 362–373.

Hu, T. C. and Shing, M. T. (1984): Computation of Matrix Chain Products Part II, *SIAM Journal of Computing*, Vol. 13, No. 2, pp. 228–251.

Hu, T. C. and Tucker, A. C. (1971): Optimal Computer Search Trees and Variable-Length Alphabetical Codes, *SIAM Journal on Applied Mathematics*, Vol. 21, No. 4, pp. 514–532.

Hu, T. H., Tang, C. Y. and Lee, R. C. T. (1992): An Average Analysis of a Resolution Principle Algorithm in Mechanical Theorem Proving, *Annals of Mathematics and Artificial Intelligence*, Vol. 6, pp. 235–252.

Huang, S. H. S., Liu, H. and Viswanathan, V. (1994): Parallel Dynamic Programming, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 3, pp. 326–328.

Huang, X. and Waterman, M. S. (1992): Dynamic Programming Algorithms for Restriction Map Comparison, *Computational Applied Biology Science*, pp. 511–520.

Huddleston, S. and Mehlhorn, K. (1982): A New Data Structure for Representing Sorted Lists, *Acta Informatica*, Vol. 17, pp. 157–184.

Huffman, D. A. (1952): A Method for the Construction of Minimum-Redundancy Codes, *Proceedings of IRE*, Vol. 40, pp. 1098–1101.

Huo, D. and Chang, G. J. (1994): The Provided Minimization Problem in Tree, *SIAM Journal of Computing*, Vol. 23, No. 1, pp. 71–81.

Huyn, N., Dechter, R. and Pearl, J. (1980): Probabilistic Analysis of the Complexity of A*, *Artificial Intelligence*, Vol. 15, pp. 241–254.

Huynh, D. T. (1986): Some Observations about the Randomness of Hard Problems, *SIAM Journal on Computing*, Vol. 15, No. 4, pp. 1101–1105.

Hyafil, L. (1976): Bounds for Selection, *SIAM Journal on Computing*, Vol. 5, No. 1, pp. 109–114.

Ibaraki, T. (1977): The Power of Dominance Relations in Branch-and-Bound Algorithms, *Journal of the ACM*, Vol. 24, No. 2, pp. 264–279.

Ibaraki, T. and Nakamura, Y. (1994): A Dynamic Programming Method for Single Machine Scheduling, *European Journal of Operational Research*, Vol. 76, p. 72.

Ibarra, O. H. and Kim, C. E. (1975): Fast Approximation Algorithms for the Knapsack and the Sum of Subset Problems, *Journal of the ACM*, Vol. 22, pp. 463–468.

Imai, H. (1993): Geometric Algorithms for Linear Programming, *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, Vol. E76-A, No. 3, pp. 259–264.

Imai, H., Kato, K. and Yamamoto, P. (1989): A Linear-Time Algorithm for Linear L1 Approximation of Points, *Algorithmica*, Vol. 4, No. 1, pp. 77–96.

Imai, H., Lee, D. T. and Yang, C. D. (1992): 1-Segment Center Problem, *ORSA Journal on Computing*, Vol. 4, No. 4, pp. 426–434.

Imase, M. and Waxman, B. M. (1991): Dynamic Steiner Tree Problem, *SIAM Journal on Discrete Mathematics*, Vol. 4, No. 3, pp. 369–384.

Irani, S., Shukla, S. and Gupta, R. (2003): Online Strategies for Dynamic Power Management in Systems with Multiple Power-Saving States, *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 2, pp. 325–346.

Italiano, G. F. (1986): Amortized Efficiency of a Path Retrieval Data Structure, *Theoretical Computer Science*, Vol. 48, No. 2, pp. 273–281.

Ivanov, A. G. (1984): Distinguishing an Approximate Word's Inclusion on Turing Machine in Real Time, *Izvestiia Academii Nauk USSSR, Series Math*, Vol. 48, pp. 520–568.

Iwamura, K. (1993): Discrete Decision Process Model Involves Greedy Algorithm Over Greedoid, *Journal of Information and Optimization Sciences*, Vol. 14, pp. 83–86.

Jadhavm, S. and Mukhopadhyay, A. (1993): Computing a Centerpoint of a Finite Planar Set of Points in Linear Time, *Proceedings of the 9th Annual Symposium on Computational Geometry*, ACM Press, San Diego, California, pp. 83–90.

Jain, K. and Vazirani, V. V. (2001): Approximation Algorithms for Metric Facility Location and *k*-Median Problems Using the Primal-Dual Schema and Lagrangian Relaxation, *Journal of the ACM*, Vol. 48, No. 2.

Janssen, J., Krizanc, D., Narayanan, L. and Shende, S. (2000): Distributed Online Frequency Assignment in Cellular Networks, *Journal of Algorithms*, Vol. 36, pp. 119–151.

Jayram, T., Kimbrel, T., Krauthgamer, R., Schieber, B. and Sviridenko, M. (2001): Online Server Allocation in a Server Farm via Benefit Task Systems, *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, ACM Press, Hersonissos, Greece, pp. 540–549.

Jerrum, M. (1985): The Complexity of Finding Minimum-Length Generator Sequences, *Theoretical Computer Science*, Vol. 36, pp. 265–289.

Jiang, T., Kearney, P. and Li, M. (1998): Orchestrating Quartets: Approximation and Data Correction, *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, IEEE Press, Palo Alto, California, pp. 416–425.

Jiang, T., Kearney, P. and Li, M. (2001): A Polynomial Time Approximation Scheme For Inferring Evolutionary Trees from Quartet Topologies and Its Application, *SIAM Journal on Computing*, Vol. 30, No. 6, pp. 1942–1961.

Jiang, T., Lawler, E. L. and Wang, L. (1994): Aligning Sequences via an Evolutionary Tree: Complexity and Approximation, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, ACM Press, Montreal, Quebec, pp. 760–769.

Jiang, T. and Li, M. (1995): On the Approximation of Shortest Common Supersequences and Longest Common Subsequences, *SIAM Journal on Computing*, Vol. 24, No. 5, pp. 1122–1139.

Jiang, T., Wang, L. and Lawler, E. L. (1996): Approximation Algorithms for Tree Alignment with a Given Phylogeny, *Algorithmica*, Vol. 16, pp. 302–315.

John, J. W. (1988): A New Lower Bound for the Set-Partitioning Problem, *SIAM Journal on Computing*, Vol. 17, No. 4, pp. 640–647.

Johnson, D. S. (1973): Near-Optimal Bin Packing Algorithms, *MIT Report MAC TR–109*.

Johnson, D. S. (1974): Approximation Algorithms for Combinatorial Problems, *Journal of Computer and System Sciences*, Vol. 9, pp. 256–278.

Johnson, D. S., Demars, A., Ullman, J. D., Garey, M. R. and Graham, R. L. (1974): Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms, *SIAM Journal on Computing*, Vol. 3, No. 4, pp. 299–325.

Johnson, D. S., Yanakakis, M. and Papadimitriou, C. H. (1988): On Generating All Maximal Independent Sets, *Information Processing Letters*, Vol. 27, No. 3, pp. 119–123.

Johnson, H. W. and Burrus, C. S. (1983): The Design of Optimal DFT Algorithms Using Dynamic Programming, *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 31, No. 2, pp. 378–387.

Jonathan, T. (1989): Approximation Algorithms for the Shortest Common Superstring Problem, *Information and Computation*, Vol. 83, pp. 1–20.

Jorma, T. and Ukkonen, E. (1988): A Greedy Approximation Algorithm for Constructing Shortest Common Superstrings, *Theoretical Computer Science*, Vol. 57, pp. 131–145.

Juedes, D. W. and Lutz, J. H. (1995): The Complexity and Distribution of Hard Problems, *SIAM Journal of Computing*, Vol. 24, No. 2, pp. 279–295.

Kalyanasundaram, B. and Pruhs, K. (1993): On-Line Weighted Matching, *Journal of Algorithms*, Vol. 14, pp. 478–488.

Kamidoi, Y., Wakabayashi, S. and Yoshida, N. (2002): A Divide-and-Conquer Approach to the Minimum k-Way Cut Problem, *Algorithmica*, Vol. 32, pp. 262–276.

Kannan, R., Mount, J. and Tayur, S. (1995): A Randomized Algorithm to Optimize Over Certain Convex Sets, *Mathematical Operating Research*, Vol. 20, No. 3, pp. 529–549.

Kannan, S., Lawler, E. L. and Warnow, T. J. (1996): Determining the Evolutionary Tree Using Experiments, *Journal of Algorithms*, Vol. 21, pp. 26–50.

Kannan, S. and Warnow, T. (1994): Inferring Evolutionary History from DNA Sequences, *SIAM Journal on Computing*, Vol. 23, pp. 713–737.

Kannan, S. and Warnow, T. (1995): Tree Reconstruction from Partial Orders, *SIAM Journal on Computing*, Vol. 24, pp. 511–519.

Kantabutra, V. (1994): Linear-Time Near-Optimum-Length Triangulation Algorithm for Convex Polygons, *Journal of Computer and System Sciences*, Vol. 49, No. 2, pp. 325–333.

Kao, E. P. C. and Queyranne, M. (1982): On Dynamic Programming Methods for Assembly Line Balancing, *Operations Research*, Vol. 30, No. 2, pp. 375–390.

Kao, M. Y., Ma, Y., Sipser, M. and Yin, Y. (1998): Optimal Constructions of Hybrid Algorithms, *Journal of Algorithms*, Vol. 29, pp. 142–164.

Kao, M. Y. and Tate, S. R. (1991): Online Matching with Blocked Input, *Information Processing Letters*, Vol. 38, pp. 113–116.

Kaplan, H. and Shamir, R. (1994): On the Complexity of DNA Physical Mapping, *Advances in Applied Mathematics*, Vol. 15, pp. 251–261.

Karger, D. R., Klein, P. N. and Tarjan, R. E. (1995): Randomized Linear-Time Algorithm to Find Minimum Spanning Trees, *Journal of the Association for Computing Machinery*, Vol. 42, No. 2, pp. 321–328.

Karger, D. R., Phillips, S. and Torng, E. (1994): A Better Algorithm for an Ancient Scheduling Problem, *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, Arlington, Virginia, pp. 132–140.

Karger, D. R. and Stein, C. (1996): New Approach to the Minimum Cut Problem, *Journal of the Association for Computing Machinery*, Vol. 43, No. 4, pp. 601–640.

Karkkainen, J., Navarro, G. and Ukkonen, E. (2000): Approximate String Matching over Ziv-Lempel Compressed Text, *Lecture Notes in Computer Science*, Vol. 1848, pp. 195–209.

Karlin, A. R., Manasse, M. S., Rudolph, L. and Sleator, D. D. (1988): Competitive Snoopy Caching, *Algorithmica*, Vol. 3, No. 1, pp. 79–119.

Karoui, N. E. and Quenez, M. C. (1995): Dynamic Programming and Pricing of Contingent Claims in an Incomplete Market, *SIAM Journal on Control and Optimization*, 1995, pp. 27–66.

Karp, R. M. (1972): Reducibility among Combinatorial Problems, in R. Miller and J. Thatcher (Eds.), *Complexity of Computer Computations* (pp. 85–103), Plenum Press, New York.

Karp, R. M. (1986): Combinatorics; Complexity and Randomness, *Communications of the ACM*, Vol. 29, No. 2, 1986, pp. 98–109.

Karp, R. M. (1994): Probabilistic Recurrence Relations, *Journal of the Association for Computing Machinery*, Vol. 41, No. 6, 1994, pp. 1136–1150.

Karp, R. M., Montwani, R. and Raghavan, P. (1988): Deferred Data Structuring, *SIAM Journal on Computing*, Vol. 17, No. 5, 1988, pp. 883–902.

Karp, R. M. and Pearl, J. (1983): Searching for an Optimal Path in a Tree with Random Costs, *Artificial Intelligence*, Vol. 21, 1983, pp. 99–116.

Karp, R. M. and Rabin, M. O. (1987): Efficient Randomized Pattern-Matching Algorithms, *IBM Journal of Research and Development*, Vol. 31, 1987, pp. 249–260.

Karp, R. M., Vazirani, U. V. and Vazirani, V. V. (1990): An Optimal Algorithm for On-Line Bipartite Matching, *Proceedings of the 22nd ACM Symposium on Theory of Computing*, ACM Press, Baltimore, Maryland, pp. 352–358.

Kearney, P., Hayward, R. B. and Meijer, H. (1997): Inferring Evolutionary Trees from Ordinal Data, *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, New Orleans, Louisiana, pp. 418–426.

Kececioglu, J. D. (1991): *Exact and approximate algorithms for sequence recognition problems in molecular biology*. Unpublished Ph.D. thesis, University of Arizona.

Kececioglu, J. D. and Myers, W. E. (1995a): Exact and Approximation Algorithms for the Sequence Reconstruction Problem, *Algorithmica*, Vol. 13, pp. 7–51.

Kececioglu, J. D. and Myers, W. E. (1995b): Combinatorial Algorithms for DNA

Sequence Assembly, *Algorithmica*, Vol. 13, pp. 7–51.

Kececioglu, J. D. and Sankoff, D. (1993): Exact and Approximation Algorithms for the Inversion Distance between Two Chromosomes, *Lecture Notes in Computer Science*, Vol. 684, pp. 87–105.

Kececioglu, J. D. and Sankoff, D. (1995): Exact and Approximate Algorithms for Sorting by Reversals with Application to Genome Rearrangement, *Algorithmica*, Vol. 13, pp. 180–210.

Keogh, E., Chu, S., Hart, D. and Pazzani, M. (2001): An Online Algorithm for Segmenting Time Series, *IEEE Computer Science Press*, pp. 289–296.

Khachian, L. G. (1979): A Polynomial Algorithm for Linear Programming, *Doklady Akademii Nauk*, USSR, Vol. 244, No. 5, pp. 1093–1096. Translated in *Soviet Math. Doklady*, Vol. 20, pp. 191–194.

Khuller, S., Mitchell, S. and Vazirani, V. V. (1994): On-Line Algorithms for Weighted Bipartite Matching and Stable Marriage, *Theoretical Computer Science*, Vol. 127, No. 2, pp. 255–267.

Kilpelainen P. and Mannila H. (1995): Ordered and Unordered Tree Inclusion, *SIAM Journal on Computing*, Vol. 24, No. 2, pp. 340–356.

Kim, D. S., Yoo, K. H., Chwa, K. Y. and Shin, S. Y. (1998): Efficient Algorithms for Computing a Complete Visibility Region in Three-Dimensional Space, *Algorithmica*, Vol. 20, pp. 201–225.

Kimura, M. (1979): The Neutral Theory of Molecular Evolution, *Scientific American*, Vol. 241, pp. 98–126.

King, T. (1992): *Dynamic Data Structures: Theory and Applications*, Academic Press, London.

Kingston, J. H. (1986): The Amortized Complexity of Henriksen's Algorithm, *BIT*, Vol. 26, No. 2, pp. 156–163.

Kirousis, L. M. and Papadimitriou, C. H. (1988): The Complexity of Recognizing Polyhedral Scenes, *Journal of Computer and System Sciences*, Vol. 37, No. 1, pp. 14–38.

Kirpatrick, D. and Snoeyink, J. (1993): Tentative Prune-and-Search for Computing Fixed-Points with Applications to Geometric Computation, *Proceedings of the Ninth Annual Symposium on Computational Geometry*, ACM Press, San Diego, California, pp. 133–142.

Kirpatrick, D. and Snoeyink, J. (1995): Tentative Prune-and-Search for Computing Fixed-Points with Applications to Geometric Computation, *Fundamenta Informaticae*, Vol. 22, pp. 353–370.

Kirschenhofer, P., Prodinger, H. and Szpankowski, W. (1994): Digital Search Trees Again Revisited: The Internal Path Length Perspective, *SIAM Journal on Applied Mathematics*, Vol. 23, No. 3, pp. 598–616.

Klarlund, N. (1999): An $n \log n$ Algorithm for Online BDD Refinement, *Journal of Algorithms*, Vol. 32, pp. 133–154.

Klawe, M. M. (1985): A Tight Bound for Black and White Pebbles on the Pyramid, *Journal of the ACM*, Vol. 32, No. 1, pp. 218–228.

Kleffe, J. and Borodovsky, M. (1992): First and Second Moment of Counts of Words in Random Texts Generated by Markov Chains, *Computer Applications in the Biosciences*, Vol. 8, pp. 433–441.

Klein, C. M. (1995): A Submodular Approach to Discrete Dynamic Programming, *European Journal of Operational Research*, Vol. 80, pp. 145–155.

Klein, P. and Subramanian, S. (1997): A Randomized Parallel Algorithm for Single-Source Shortest Paths, *Journal of Algorithm*, Vol. 25, pp. 205–220.

Kleinberg, J. and Tardos, E. (2002): Approximation Algorithms for Classification Problems with Pairwise Relationships: Metric Labeling and Markov Random Fields, *Journal of the ACM*, Vol. 49, No. 5, pp. 616–639.

Knuth, D. E. (1969): The Art of Computer Programming, *Fundamental Algorithms*, Vol. 1, p. 634.

Knuth, D. E. (1971): Optimum Binary Search Trees, *Acta Informatica*, Vol. 1, pp. 14–25.

Knuth, D. E. (1973): *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, Mass.

Ko, M. T., Lee, R. C. T. and Chang, J. S. (1990): An Optimal Approximation Algorithm for the Rectilinear *m*-Center Problem, *Algorithmica*, Vol. 5, pp. 341–352.

Kolliopoulos, S. G. and Stein, C. (2002): Approximation Algorithms for Single-Source Unsplittable Flow, *SIAM Journal on Computing*, Vol. 31, No. 3, pp. 919–946.

Kolman, P. and Scheideler, C. (2001): Simple On-Line Algorithms for the Maximum Disjoint Paths Problem, *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, Crete Island, Greece, pp. 38–47.

Kontogiannis, S. (2002): Lower Bounds and Competitive Algorithms for Online Scheduling of Unit-Size Tasks to Related Machines, *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, ACM Press, Montreal, Canada, pp. 124–133.

Koo, C. Y., Lam, T. W., Ngan, T. W., Sadakane, K. and To, K. K. (2003): On-line Scheduling with Tight Deadlines, *Theoretical Computer Science*, Vol. 295, 2003, pp. 1–12.

Korte, B. and Louasz, L. (1984): Greedoids: A Structural Framework for the Greedy Algorithms, in W. R. Pulleybland (Ed.), *Progress in Combinatorial Optimization* (pp. 221–243), Academic Press, London.

Kortsarz, G. and Peleg, D.(1995): Approximation Algorithms for Minimum-Time Broadcast, *SIAM Journal on Discrete Mathematics*, Vol. 8, pp. 407–421.

Kossmann, D., Ramsak, F. and Rost, S. (2002): Shooting Stars in the Sky: An Online Algorithm for Skyline Queries, *Proceedings of the 28th VLDB Conference*, VLDB Endowment, Hong Kong, pp. 275–286.

Kostreva, M. M. and Wiecek, M. M. (1993): Time Dependency in Multiple Objective Dynamic Programming, *Journal of Mathematical Analysis and Applications*, Vol. 173, pp. 289–307.

Kou, L., Markowsky, G. and Berman, L. (1981): A Fast Algorithm for Steiner Trees, *Acta Informatica*, Vol. 15, pp. 141–145.

Koutsoupias, E. and Nanavati, A. (2003): Online Matching Problem on a Line, *Lecture Notes in Computer Science*, Vol. 2909, pp. 179–191.

Koutsoupias, E. and Papadimitriou, C. H. (1995): On the *k*-Server Conjecture, *Journal of the ACM*, Vol. 42, No. 5, pp. 971–983.

Kozen, D. C. (1997): *The Design and Analysis of Algorithms*, Springer-Verlag, New York.

Kozhukhin, C. G. and Pevzner, P. A. (1994): Genome Inhomogeneity Is Determined Mainly by WW and SS Dinucleotides, *Computer Applications in the Biosciences*, pp. 145–151.

Krarup, J. and Pruzan, P. (1986): Assessment Approximate Algorithms: The Error Measures's Crucial Role, *BIT*, Vol. 26, No. 3, pp. 284–294.

Krivanek, M. and Moravek, J. (1986): NP-Hard Problems in Hierarchical-Tree Clustering, *Acta Informatica*, Vol. 23, No. 3, pp. 311–323.

Krogh, A., Brown, M., Mian, I. S., Sjolander, K. and Haussler, D. (1994): Hidden Markov Models in Computational Biology: Applications to Protein Modeling, *Journal of Molecular Biology*, Vol. 235, pp. 1501–1531.

Kronsjö, L. I. (1987): *Algorithms: Their Complexity and Efficiency*, John Wiley & Sons, New York.

Krumke, S. O., Marathe, M. V. and Ravi, S. S. (2001): Models and Approximation Algorithms for Channel Assignment in Radio Networks, *Wireless Networks*, Vol. 7, No. 6, pp. 575–584.

Kruskal, J. B. Jr. (1956): On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, *Proceedings of the American Mathematical Society*, Vol. 7, No. 1, pp. 48–50.

Kryazhimskiy, A. V. and Savinov V. B. (1995): Travelling-Salesman Problem with Moving Objects, *Journal of Computer and System Sciences*, Vol. 33, No. 3, pp. 144–148.

Kucera, L. (1991): *Combinatorial Algorithms*, IOP Publishing, Philadelphia.

Kumar, S., Kiran, R. and Pandu, C. (1987): Linear Space Algorithms for the LCS Problem, *Acta Informatica*, Vol. 24, No. 3, pp. 353–362.

Kung, H. T., Luccio, F. and Preparata, F. P. (1975): On Finding the Maxima of a Set of Vectors, *Journal of the ACM*, Vol. 22, No. 4, pp. 469–476.

Kurtz, S. A. (1987): A Note on Randomized Polynomial Time, *SIAM Journal on Computing*, Vol. 16, No. 5, pp. 852–853.

Lai, T. W. and Wood, D. (1998): Adaptive Heuristics for Binary Search Trees and Constant Linkage Cost, *SIAM Journal on Applied Mathematics*, Vol. 27, No. 6, pp. 1564–1591.

Landau, G. M. and Schmidt, J. P. (1993): An Algorithm for Approximate Tandem Repeats, *Lecture Notes in Computer Science*, Vol. 684, pp. 120–133.

Landau, G. M. and Vishkin, U. (1989): Fast Parallel and Serial Approximate String Matching, *Journal of Algorithms*, Vol. 10, pp. 157–169.

Langston, M. A. (1982): Improved 0=1-Interchange Scheduling, *BIT*, Vol. 22, pp. 282–290.

Lapaugh, A. S. (1980): Algorithms for Integrated Circuit Layout: Analytic Approach, *Computer Science*, pp. 155–169.

Laquer, H. T. (1981): Asymptotic Limits for a Two-Dimensional Recursion, *Studies in Applied Mathematics*, pp. 271–277.

Larson, P. A. (1984): Analysis of Hashing with Chaining in the Prime Area, *Journal of Algorithms*, Vol. 5, pp. 36–47.

Lathrop, R. H. (1994): The Protein Threading Problem with Sequence Amino Acid Interaction Preferences Is NP-Complete (Prove NP-Complete Problem), *Protein Engineering*, Vol. 7, pp. 1059–1068.

Lau, H. T. (1991): *Algorithms on Graphs*, TAB Books, Blue Ridge Summit, Philadelphia.

Lawler, E. L. (1976): *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G. and Shmoys, D. B. (1985): *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons, New York.

Lawler, E. L. and Moore, J. (1969): A Functional Equation and Its Application to Resource Allocation and Sequencing Problems, *Management Science*, Vol. 16, No. 1, pp. 77–84.

Lawler, E. L. and Wood, D. (1966): Branch-and-Bound Methods: A Survey, *Operations Research*, Vol. 14, pp. 699–719.

Lee, C. C. and Lee, D. T. (1985): A Simple On-Line Bin-Packing Algorithm, *Journal of the Association for Computing Machinery*, Vol. 32, No. 3, pp. 562–572.

Lee, C. T. and Sheu, C. Y. (1992): A Divide-and-Conquer Approach with Heuristics of Motion Planning for a Cartesian Manipulator, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22, No. 5, pp. 929–944.

Lee, D. T. (1982): On *k*-Nearest Neighbor Voronoi Diagrams in the Plane, *IEEE Transactions on Computers*, Vol. C–31, pp. 478–487.

Lee, D. T. and Lin, A. K. (1986): Computational Complexity of Art Gallery Problems, *IEEE Transactions on Information Theory*, Vol. IT–32, No. 2, pp. 276–282.

Lee, D. T. and Preparata, F. P. (1984): Computational Geometry: A Survey, *IEEE Transactions on Computers*, Vol. C–33, pp. 1072–1101.

Lee, J. (2003a): Online Deadline Scheduling: Team Adversary and Restart, *Lecture Notes in Computer Science*, Vol. 2909, pp. 206–213.

Lee, J. (2003b): Online Deadline Scheduling: Multiple Machines and Randomization, *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, San Diego, California, pp. 19–23.

Leighton, T. and Rao, S. (1999): Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms, *Journal of the ACM*, Vol. 46, No. 6, pp. 787–832.

Lent, J. and Mahmoud, H. M. (1996): On Tree-Growing Search Strategies, *The Annals of Applied Probability*, Vol. 6, No. 4, pp. 1284–1302.

Leonardi, S., Spaccamela, A. M., Presciutti, A. and Ros, A. (2001): On-Line Randomized Call Control Revisited, *SIAM Journal on Computing*, Vol. 31, pp. 86–112.

Leoncini, M., Manzini, G. and Margara, L. (1999): Parallel Complexity of Numerically Accurate Linear System Solvers, *SIAM Journal on Computing*, Vol. 28, No. 6, pp. 2030–2058.

Levcopoulos, C. and Lingas, A. (1987): On Approximation Behavior of the Greedy Triangulation for Convex Polygons, *Algorithmica*, Vol. 2, pp. 175–193.

Levin, L. A. (1986): Average Case Complete Problem, *SIAM Journal on Computing*, Vol. 15, No. 1, pp. 285–286.

Lew, W. and Mahmoud, H. M. (1992): The Joint Distribution of Elastic Buckets in Multiway Search Trees, *SIAM Journal on Applied Mathematics*, Vol. 23, No. 5, pp. 1050–1074.

Lewandowski, G., Condon, A. and Bach, E. (1996): Asynchronous Analysis of Parallel Dynamic Programming Algorithms, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, pp. 425–438.

Lewis, H. R. and Denenberg, L. (1991): *Data Structures and Their Algorithms*, Harper Collins, New York.

Liang, S. Y. (1985): *Parallel algorithm for a personnel assignment problem.* Unpublished Master's thesis, National Tsing Hua University, Hsinchu, Taiwan.

Liang, Y. D. (1994): On the Feedback Vertex Set Problem in Permutation Graphs, *Information Processing Letters*, Vol. 52, No. 3, pp. 123–129.

Liao, L. Z. and Shoemaker, C. A. (1991): Convergence in Unconstrained Discrete-time Differential Dynamic Programming, *IEEE Transactions Automatic Control*, Vol. 36, pp. 692–706.

Liaw, B. C. and Lee, R. C. T. (1994): Optimal Algorithm to Solve the Minimum Weakly Cooperative Guards Problem for 1-Spiral Polygons, *Information Processing Letters*, Vol. 52, No. 2, pp. 69–75.

Lin, C. K., Fan, K. C. and Lee, F. T. (1993): On-line Recognition by Deviation-Expansion Model and Dynamic Programming Matching, *Pattern Recognition*, Vol. 26, No. 2, pp. 259–268.

Lin, G., Chen, Z. Z., Jiang, T. and Wen, J. (2002): The Longest Common Subsequence Problem for Sequences with Nested Arc Annotations (Prove NP-Complete Problem), *Journal of Computer and System Sciences*, Vol. 65, pp. 465–480.

Lipton, R. J. (1995): Using DNA to Solve NP-Complete Problems, *Science*, Vol. 268, pp. 542–545.

Little, J. D. C., Murty, K. G., Sweeney, D. W. and Karel, C. (1963): An Algorithm for the Traveling Salesman Problem, *Operations Research*, Vol. 11, pp. 972–989.

Littman, M. L., Cassandra, A. R. and Kaelbling, L. P. (1996): An Algorithm for Probabilistic Planning: Efficient Dynamic-Programming Updates in Partially Observable Markov Decision Processes, *Artificial Intelligence*, Vol. 76, pp. 239–286.

Liu, C. L. (1985): *Elements of Discrete Mathematics*, McGraw-Hill, New York.

Liu, J. (2002): On Adaptive Agentlets for Distributed Divide-and-Conquer: A

Dynamical Systems Approach, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 32, No. 2, pp. 214–227.

Lo, V., Rajopadhye, S., Telle, J. A. and Zhong, X. (1996): Parallel Divide and Conquer on Meshes, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 10, pp. 1049–1058.

Long, T. J. and Selman, A. L. (1986): Relativizing Complexity Classes with Sparse Oracles, *Journal of the ACM*, Vol. 33, No. 3, pp. 618–627.

Lopez, J. and Zapata, E. (1994): Unified Architecture for Divide and Conquer Based Tridiagonal System Solvers, *IEEE Transactions on Computers*, Vol. 43, No. 12, pp. 1413–1425.

Louchard, G., Szpankowski, W. and Tang, J. (1999): Average Profile of the Generalized Digital Search Tree and the Generalized Lempel—Ziv Algorithm, *SIAM Journal on Applied Mathematics*, Vol. 28, No. 3, pp. 904–934.

Lovasz, L., Naor, M., Newman, I. and Wigderson, A. (1995): Search Problems in the Decision Tree Model, *SIAM Journal on Applied Mathematics*, Vol. 8, No. 1, pp. 119–132.

Luby, M. (1986): A Simple Parallel Algorithm for the Maximal Independent Set Problem, *SIAM Journal on Computing*, Vol. 15, No. 4, pp. 1036–1053.

Lueker, G. (1998): Average-Case Analysis of Off-Line and On-Line Knapsack Problems, *Journal of Algorithms*, Vol. 29, pp. 277–305.

Lyngso, R. B. and Pedersen, C. N. S. (2000): Pseudoknots in RNA Secondary Structure (Prove NP-Complete Problem), *ACM*, pp. 201–209.

Ma, B., Li, M. and Zhang, L. (2000): From Gene Trees to Species Trees, *SIAM Journal on Applied Mathematics*, Vol. 30, No. 3, pp. 729–752.

Maes, M. (1990): On a Cyclic String-to-String Correction Problem, *Information Processing Letters*, Vol. 35, pp. 73–78.

Maffioli, F. (1986): Randomized Algorithm in Combinatorial Optimization: A Survey, *Discrete Applied Mathematics*, Vol. 14, No. 2, June, pp. 157–170.

Maggs, B. M. and Sitaraman, R. K. (1999): Simple Algorithms for Routing on Butterfly Networks with Bounded Queues, *SIAM Journal on Computing*, Vol. 28, pp. 984–1003.

Maier, D. (1978): The Complexity of Some Problems on Subsequences and Supersequences, *Journal of the ACM*, Vol. 25, pp. 322–336.

Maier, D. and Storer, J. A. (1978): A Note on the Complexity of the Superstring Problem, *Proceedings of the 12th Conference on Information Sciences and Systems (CISS)*, The Johns Hopkins University, Baltimore, Maryland, pp. 52–56.

Makinen, E. (1987): On Top-Down Splaying, *BIT*, Vol. 27, No. 3, pp. 330–339.

Manacher, G. (1975): A New Linear-Time "On-line" Algorithm for Finding the Smallest Initial Palindrome of a String, *Journal of the ACM*, Vol. 22, No. 3, pp. 346–351.

Manasse, M., McGeoch, L. and Sleator, D. (1990): Competitive Algorithms for Server Problems, *Journal of Algorithms*, Vol. 11, No. 2, pp. 208–230.

Manber, U. (1989): *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, Reading, Mass.

Mandic, D. and Cichocki, A. (2003): An Online Algorithm for Blind Extraction of Sources with Different Dynamical Structures, *Proceedings of the 4th International Symposium on Independent Component Analysis and Blind Signal Separation (ICA2003)*, Nara, Japan, pp. 645–650.

Mandrioli, D. and Ghezzi, C. (1987): *Theoretical Foundations of Computer Science*, John Wiley & Sons, New York.

Maniezzo, V. (1998): Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem, *Research Report CSR 98–1*.

Mansour, Y. and Schieber, B. (1992): The Intractability of Bounded Protocols for On-line Sequence Transmission over Non-FIFO Channels, *Journal of the*

ACM, Vol. 39, No. 4, pp. 783–799.

Marion, J. Y. (2003): Analysing the Implicit Complexity of Programs, *Information and Computation*, Vol. 183, No. 1, pp. 2–18.

Martello, S. and Toth, P. (1990): *Knapsack Problem Algorithms & Computer Implementations*, John Wiley & Sons, New York.

Martin, G. L. and Talley, J. (1995): Recognizing Handwritten Phrases from U. S. Census Forms by Combining Neural Networks and Dynamic Programming, *Journal of Artificial Neural Networks*, Vol. 2, pp. 167–193.

Martinez, C. and Roura, S. (2001): Optimal Sampling Strategies in Quicksort and Quickselect, *SIAM Journal on Computing*, Vol. 31, No. 3, pp. 683–705.

Matousek, J. (1991): Randomized Optimal Algorithm for Slope Selection, *Information Processing Letters*, Vol. 39, No. 4, pp. 183–187.

Matousek, J. (1995): On Enclosing K Points by a Circle, *Information Processing Letters*, Vol. 53, No. 4, pp. 217–221.

Matousek, J. (1996): Derandomization in Computational Geometry, *Journal of Algorithms*, Vol. 20, pp. 545–580.

Mauri, G., Pavesi, G. and Piccolboni, A. (1999): Approximation Algorithms for Protein Folding Prediction, *Proceedings of the 10th Annual Symposium on Discrete Algorithms*, SIAM, Baltimore, Maryland, pp. 945–946.

McDiarmid, C. (1988): Average-Case Lower Bounds for Searching, *SIAM Journal on Computing*, Vol. 17, No. 5, pp. 1044–1060.

McHugh, J. A. (1990): *Algorithmic Graph Theory*, Prentice-Hall, London.

Meacham, C. A. (1981): A Probability Measure for Character Compatibility, *Mathematical Biosciences*, Vol. 57, pp. 1–18.

Megiddo, N. (1983): Linear-Time Algorithm for Linear Programming in R3 and Related Problems, *SIAM Journal on Computing*, Vol. 12, No. 4, pp. 759–776.

Megiddo, N. (1984): Linear Programming in Linear Time When the Dimension Is Fixed, *Journal of the ACM*, Vol. 31, No. 1, pp. 114–127.

Megiddo, N. (1985): Note Partitioning with Two Lines in the Plane, *Journal of Algorithms*, Vol. 6, No. 3, pp. 430–433.

Megiddo, N. and Supowit, K. J. (1984): On the Complexity of Some Common Geometric Location Problems, *SIAM Journal on Computing*, Vol. 13, No. 1, pp. 182–196.

Megiddo, N. and Zemel, E. (1986): An $O(n \log n)$ Randomizing Algorithm for the Weighted Euclidean 1-Center Problem, *Journal of Algorithms*, Vol. 7, No. 3, pp. 358–368.

Megow, N. and Schulz, A. (2003): Scheduling to Minimize Average Completion Time Revisited: Deterministic On-Line Algorithms, *Lecture Notes in Computer Science*, Vol. 2909, pp. 227–234.

Mehlhorn, K. (1984): *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, Springer-Verlag, Berlin.

Mehlhorn, K. (1984): *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin.

Mehlhorn, K. (1987): *Data Structures & Algorithms: Sorting and Searching*, Springer-Verlag, New York.

Mehlhorn, K. (1988): A Faster Approximation Algorithm for the Steiner Problems in Graphs, *Information Processing Letters*, Vol. 27, No. 3, pp. 125–128.

Mehlhorn, K., Naher, S. and Alt, H. (1988): A Lower Bound on the Complexity of the Union-Split-Find Problem, *SIAM Journal on Computing*, Vol. 17, No. 6, pp. 1093–1102.

Mehlhorn, K. and Tsakalidis, A. (1986): An Amortized Analysis of Insertions into AVL-Trees, *SIAM Journal on Computing*, Vol. 15, No. 1, pp. 22–33.

Meijer, H. and Rappaport, D. (1992): Computing the Minimum Weight Triangulation of a Set of Linearly Ordered Points, *Information Processing Letters*, Vol. 42, No. 1, pp. 35–38.

Meleis, W. M. (2001): Dual-Issue Scheduling for Binary Trees with Spills and Pipelined Loads, *SIAM Journal on Applied Mathematics*, Vol. 30, No. 6, pp. 1921–1941.

Melnik, S. and Garcia-Molina, H. (2002): Divide-and-Conquer Algorithm for Computing Set Containment Joins, *Lecture Notes in Computer Science*, Vol. 2287, pp. 427–444.

Merlet, N. and Zerubia, J. (1996): New Prospects in Line Detection by Dynamic Programming, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, pp. 426–431.

Messinger, E., Rowe, A. and Henry, R. (1991): A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 1, pp. 1–12.

Miller, G. L. and Teng, S. H. (1999): The Dynamic Parallel Complexity of Computational Circuits, *SIAM Journal on Computing*, Vol. 28, No. 5, pp. 1664–1688.

Minoux, M. (1986): *Mathematical Programming: Theory and Algorithms*, John-Wiley & Sons, New York.

Mitten, L. (1970): Branch-and-Bound Methods: General Formulation and Properties, *Operations Research*, Vol. 18, pp. 24–34.

Mohamed, M. and Gader, P. (1996): Handwritten Word Recognition Using Segmentation-Free Hidden Markov Modeling and Segmentation-Based Dynamic Programming Techniques, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, pp. 548–554.

Monien, B. and Sudborough, I. H. (1988): Min Cut Is NP-Complete for Edge Weighted Trees, *Theoretical Computer Science*, Vol. 58, No. 1–3, pp. 209–229.

Monier, L. (1980): Combinatorial Solutions of Multidimensional Divide-and-Conquer Recurrences, *Journal of Algorithms*, Vol. 1, pp. 69–74.

Moor, O. de (1994): Categories Relations and Dynamic Programming, *Mathematical Structures in Computer Science*, Vol. 4, pp. 33–69.

Moran, S. (1981): General Approximation Algorithms for Some Arithmetical Combinatorial Problems, *Theoretical Computer Science*, Vol. 14, pp. 289–303.

Moran, S., Snir, M. and Manber, U. (1985): Applications of Ramsey's Theorem to Decision Tree Complexity, *Journal of the ACM*, Vol. 32, pp. 938–949.

Moret, B. M. E. and Shapiro, H. D. (1991): *Algorithms from P to NP*, Benjamin Cummings, Redwood City, California.

Morin, T. and Marsten, R. E. (1976): Branch-and-Bound Strategies for Dynamic Programming, *Operations Research*, Vol. 24, pp. 611–627.

Motta, M. and Rampazzo, F. (1996): Dynamic Programming for Nonlinear Systems Driven by Ordinary and Impulsive Controls, *SIAM Journal on Control and Optimization*, Vol. 34, pp. 199–225.

Motwani, R. and Raghavan, P. (1995): *Randomized Algorithms*, Cambridge University Press, Cambridge, England.

Mulmuley, K. (1998): *Computational Geometry: An Introduction through Randomized Algorithms*, Prentice-Hall, Englewoods Cliffs, New Jersey.

Mulmuley, K., Vazirani, U. V. and Vazirani, V. V. (1987): Matching Is as Easy as Matrix Inversion, *Combinatorica*, Vol. 7, No. 1, pp. 105–113.

Murgolo, F. D. (1987): An Efficient Approximation Scheme for Variable-Sized Bin Packing, *SIAM Journal on Computing*, Vol. 16, No. 1, pp. 149–161.

Myers, E. and Miller, W. (1989): Approximate Matching of Regular Expression, *Bulletin of Mathematical Biology*, Vol. 51, pp. 5–37.

Myers, E. W. (1994): A Sublinear Algorithm for Approximate Keyword Searching, *Algorithmica*, Vol. 12, No. 4–5, pp. 345–374.

Myoupo, J. F. (1992): Synthesizing Linear Systolic Arrays for Dynamic Programming Problems, *Parallel Processing Letters*, Vol. 2, pp. 97–110.

Nakayama, H., Nishizeki, T. and Saito, N. (1985): Lower Bounds for Combinatorial

Problems on Graphs, *Journal of Algorithms*, Vol. 6, pp. 393–399.

Naor, M. and Ruah, S. (2001): On the Decisional Complexity of Problems Over the Reals, *Information and Computation*, Vol. 167, No. 1, pp. 27–45.

Nau, D. S., Kumar, V. and Kanal, L. (1984): General Branch and Bound and Its Relation to A* and AO*, *Artificial Intelligence*, Vol. 23, pp. 29–58.

Neapolitan, R. E. and Naimipour, K. (1996): *Foundations of Algorithms*, D.C. Heath and Company, Lexington, Mass.

Neddleman, S. B. and Wunsch, C. D. (1970): A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins, *Journal of Molecular Biology*, Vol. 48, pp. 443–453.

Nemhauser, G. L. (1966): *Introduction to Dynamic Programming*, John Wiley & Sons, New York.

Nemhauser, G. L. and Ullman, Z. (1969) Discrete Dynamic Programming and Capital Allocation, *Management Science*, Vol. 15, No. 9, pp. 494–505.

Neogi, R. and Saha, A. (1995): Embedded Parallel Divide-and-Conquer Video Decompression Algorithm and Architecture for HDTV Applications, *IEEE Transactions on Consumer Electronics*, Vol. 41, No. 1, pp. 160–171.

Ney, H. (1984): The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 2, pp. 263–271.

Ney, H. (1991): Dynamic Programming Parsing for Context-Free Grammars in Continuous Speech Recognition, *IEEE Transactions on Signal Processing*, Vol. 39, pp. 336–340.

Nilsson, N. J. (1980): *Principles of Artificial Intelligence*, Tioga Publishing Company, Palo Alto, California.

Nishizeki, T., Asano, T. and Watanabe, T. (1983): An Approximation Algorithm for the Hamiltonian Walk Problem on a Maximal Planar Graph, *Discrete Applied Mathematics*, Vol. 5, No. 2, pp. 211–222.

Nishizeki, T. and Chiba, N. (1988): *Planar Graphs: Theory and Algorithms*, Elsevier, Amsterdam.

Novak, E. and Wozniakowski, H. (2000): Complexity of Linear Problems with a Fixed Output Basis, *Journal of Complexity*, Vol. 16, No. 1, pp. 333–362.

Nuyts, J., Suetens, P., Oosterlinck, A., Roo, M. De and Mortelmans, L. (1991): Delineation of ECT Images Using Global Constraints and Dynamic Programming, *IEEE Transactions on Medical Imaging*, Vol. 10, No. 4, pp. 489–498.

Ohno, S., Wolf, U. and Atkin, N. B. (1968): Evolution from Fish to Mammals by Gene Duplication, *Hereditas*, Vol. 59, pp. 708–713.

Ohta, Y. and Kanade, T. (1985): Stereo by Intra- and Inter-Scanline Search Using Dynamic Programming, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 7, pp. 139–154.

Oishi, Y. and Sugihara, K. (1995): Topology-Oriented Divide-and-Conquer Algorithm for Voronoi Diagrams, *Graphical Models and Image Processing*, Vol. 57, No. 4, pp. 303–314.

Omura, J. K. (1969): On the Viterbi Decoding Algorithm, *IEEE Transactions on Information Theory*, pp. 177–179.

O'Rourke, J. (1987): *Art Gallery Theorems and Algorithms*, Oxford University Press, Cambridge, England.

O'Rourke, J. (1998): *Computational Geometry in C*, Cambridge University Press, Cambridge, England.

Orponen, P. and Mannila, H. (1987): On Approximation Preserving Reductions: Complete Problems and Robust Measures, *Technical Report C–1987–28*.

Ouyang, Z. and Shahidehpour, S. M. (1992): Hybrid Artificial Neural Network-Dynamic Programming Approach to Unit Commitment, *IEEE Transactions on Power Systems*, Vol. 7, pp. 236–242.

Owolabi, O. and McGregor, D. R. (1988): Fast Approximate String Matching, *Software Practice and Experience*, Vol. 18, pp. 387–393.

Oza, N. and Russell, S. (2001): Experimental Comparisons of Online and Batch Versions of Bagging and Boosting, *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, San Francisco, California, pp. 359–364.

Ozden, M. (1988): A Solution Procedure for General Knapsack Problems with a Few Constraints, *Computers and Operations Research*, Vol. 15, No. 2, pp. 145–156.

Pach, J. (1993): *New Trends in Discrete and Computational Geometry*, Springer-Verlag, New York.

Pacholski, L., Szwast, W. and Tendera, L. (2000): Complexity Results for First-Order Two-Variable Logic with Counting, *SIAM Journal on Computing*, Vol. 29, No. 4, pp. 1083–1117.

Pandurangan, G. and Upfal, E. (2001): Can Entropy Characterize Performance of Online Algorithms? *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Washington, DC, pp. 727–734.

Papadimitriou, C. H. (1977): The Euclidean TSP is NP-Complete, *Theoretical Computer Science*, Vol. 4, pp. 237–244.

Papadimitriou, C. H. (1981): Worst-Case and Probabilistic Analysis of a Geometric Location Problem, *SIAM Journal on Computing*, Vol. 10, No. 3, pp. 542–557.

Papadimitriou, C. H. (1994): *Computational Complexity*, Addison-Wesley, Reading, Mass.

Papadimitriou, C. H. and Steiglitz, K. (1982): *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New Jersey.

Papadimitriou, C. H. and Yannakakis, M. (1991a): Optimization; Approximation; and Complexity Classes, *Journal of Computer and System Sciences*, Vol. 43, pp. 425–440.

Papadimitriou, C. H. and Yannakakis, M. (1991b): Shortest Paths without a Map, *Theoretical Computer Science*, Vol. 84, pp. 127–150.

Papadimitriou, C. H. and Yannakakis, M. (1992): The Traveling Salesman Problem with Distances One and Two, *Mathematics of Operations Research*, Vol. 18, No. 1, pp. 1–11.

Papadopoulou, E. and Lee, D. T. (1998): A New Approach for the Geodesic Voronoi Diagram of Points in a Simple Polygon and Other Restricted Polygonal Domains, *Algorithmica*, Vol. 20, pp. 319–352.

Parida, L., Floratos, A. and Rigoutsos, I. (1999): An Approximation Algorithm for Alignment of Multiple Sequences Using Motif Discovery, *Journal of Combinatorial Optimization*, Vol. 3, No. 2–3, pp. 247–275.

Park, J. K. (1991): Special Case of the n-Vertex Traveling-Salesman Problem That Can Be Solved in $O(n)$ Time, *Information Processing Letters*, Vol. 40, No. 5, pp. 247–254.

Parker, R. G. and Rardin, R. L. (1984): Guaranteed Performance Heuristic for the Bottleneck Traveling Salesperson Problem, *Operations Research Letters*, Vol. 2, No. 6, pp. 269–272.

Pearl, J. (1983): Knowledge Versus Search: A Quantitative Analysis Using A*, *Artificial Intelligence*, Vol. 20, pp. 1–13.

Pearson, W. R. and Miller, W. (1992): Dynamic Programming Algorithms for Biological Sequence Comparison, *Methods in Enzymology*, Vol. 210, pp. 575–601.

Pe'er, I. and Shamir, R. (1998): The Median Problems for Breakpoints are NP-Complete, *Electronic Colloquium on Computational Complexity*, Vol. 5, No. 71, pp. 1–15.

Pe'er, I. and Shamir, R. (2000): Approximation Algorithms for the Median Problem in the Breakpoint Model, in D. Sankoff and J. H. Nadeau (Eds.),

Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and the Evolution of Gene Families, Kluwer Academic Press, Dordrecht, The Netherlands.

Peleg, D. and Rubinovich, V. (2000): A Near-Tight Lower Bound on the Time Complexity of Distributed Minimum-Weight Spanning Tree Construction, *SIAM Journal on Applied Mathematics*, Vol. 30, No. 5, pp. 1427–1442.

Peng, S., Stephens, A. B. and Yesha, Y. (1993): Algorithms for a Core and *k*-Tree Core of a Tree, *Journal of Algorithms*, Vol. 15, pp. 143–159.

Penny D., Hendy, M. D. and Steel, M. (1992): Progress with Methods for Constructing Evolutionary Trees, *Trends in Ecology and Evolution*, Vol. 7, No. 3, pp. 73–79.

Perl, Y. (1984): Optimum Split Trees, *Journal of Algorithms*, Vol. 5, pp. 367–374.

Peserico, E. (2003): Online Paging with Arbitrary Associativity, *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Baltimore, Maryland, pp. 555–564.

Petr, S. (1996): A Tight Analysis of the Greedy Algorithm for Set Cover, *ACM*, pp. 435–441.

Pevzner, P. A. (1992): Multiple Alignment; Communication Cost; and Graph Matching, *SIAM Journal on Applied Mathematics*, Vol. 52, No. 6, pp. 1763–1779.

Pevzner, P. A. (2000): *Computational Molecular Biology: An Algorithmic Approach*, The MIT Press, Boston.

Pevzner, P. A. and Waterman, M. S. (1995): Multiple Filtration and Approximate Pattern Matching, *Algorithmica*, Vol. 13, No. 1–2, pp. 135–154.

Pierce, N. A. and Winfree, E. (2002): Protein Design Is NP-Hard (Prove NP-Complete Problem), *Protein Engineering*, Vol. 15, No. 10, pp. 779–782.

Pittel, B. and Weishaar, R. (1997): On-Line Coloring of Sparse Random Graphs and Random Trees, *Journal of Algorithms*, Vol. 23, pp. 195–205.

Pohl, I. (1972): A Sorting Problem and Its Complexity, *Communications of the ACM*, Vol. 15, No. 6, pp. 462–463.

Ponzio, S. J., Radhakrishnan, J. and Venkatesh, S. (2001): The Communication Complexity of Pointer Chasing, *Journal of Computer and System Sciences*, Vol. 62, No. 2, pp. 323–355.

Preparata, F. P. and Hong, S. J. (1977): Convex Hulls of Finite Sets of Points in Two and Three Dimensions, *Communications of the ACM*, Vol. 2, No. 20, pp. 87–93.

Preparata, F. P. and Shamos, M. I. (1985): *Computational Geometry: An Introduction*, Springer-Verlag, New York.

Prim, R. C. (1957): Shortest Connection Networks and Some Generalizations, *Bell System Technical Journal*, pp. 1389–1401.

Promel, H. J. and Steger, A. (2000): A New Approximation Algorithm for the Steiner Tree Problem with Performance Ratio 5/3, *Journal of Algorithms*, Vol. 36, pp. 89–101.

Purdom, P. W. Jr. and Brown, C. A. (1985a): *The Analysis of Algorithms*, Holt, Rinehart and Winston, New York.

Purdom, P. W. Jr. and Brown, C. A. (1985b): The Pure Literal Rule and Polynomial Average Time, *SIAM Journal on Computing*, Vol. 14, No. 4, pp. 943–953.

Rabin, M. O. (1976): Probabilistic Algorithm. In J. F. Traub (Ed.), *Algorithms and Complexity: New Directions and Recent Results* (pp. 21–39), Academic Press, New York.

Raghavachari, B. and Veerasamy, J. (1999): A 3/2-Approximation Algorithm for the Mixed Postman Problem, *SIAM Journal on Discrete Mathematics*, Vol. 12, No. 4, pp. 425–433.

Raghavan, P. (1988): Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs, *Journal of Computer and System*

Sciences, Vol. 37, No. 2, pp. 130–143.

Raghavan, P. and Thompson C. (1987): Randomized Rounding: A Technique for Provably Good Algorithms and Algorithmic Proofs, *Combinatorica*, Vol. 7, No. 4, pp. 365–374.

Ramanan, P., Deogun, J. S. and Liu, C. L. (1984): A Personnel Assignment Problem, *Journal of Algorithms*, Vol. 5, No. 1, pp. 132–144.

Ramesh, H. (1995): On Traversing Layered Graphs On-Line, *Journal of Algorithms*, Vol. 18, pp. 480–512.

Rangan, C. P. (1983): On the Minimum Number of Additions Required to Compute a Quadratic Form, *Journal of Algorithms*, Vol. 4, pp. 282–285.

Reingold, E. M., Nievergelt, J. and Deo, N. (1977): *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, New Jersey.

Reingold, E. M. and Supowit, K. J. (1983): Probabilistic Analysis of Divide-and-Conquer Heuristics for Minimum Weighted Euclidean Matching, *Networks*, Vol. 13, No. 1, pp. 49–66.

Rival, I. and Zaguia, N. (1987): Greedy Linear Extensions with Constraints, *Discrete Mathematics*, Vol. 63, No. 2, pp. 249–260.

Rivas, E. and Eddy, S. R. (1999): A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots, *Journal of Molecular Biology*, Vol. 285, pp. 2053–2068.

Rivest, L. R. (1995): *Game Tree Searching by Min/Max Approximation*, MIT Laboratory for Computer Science, Cambridge, Mass.

Robinson, D. F. and Foulds, L. R. (1981): Comparison of Phylogenetic Tree, *Mathematical Biosciences*, Vol. 53, pp. 131–147.

Robinson, J. A. (1965): Machine Oriented Logic Based on the Resolution Principle, *Journal of the ACM*, Vol. 12, No. 1, pp. 23–41.

Rosenkrantz, D. J., Stearns, R. E. and Lewis, P. M. (1977): An Analysis of Several Heuristics for the Traveling Salesman Problem, *SIAM Journal on Computing*, Vol. 6, pp. 563–581.

Rosenthal, A. (1982): Dynamic Programming Is Optimal for Nonserial Optimization Problems, *SIAM Journal on Computing*, Vol. 11, No. 1, pp. 47–59.

Rosler, U. (2001): On the Analysis of Stochastic Divide and Conquer Algorithms, *Algorithmica*, Vol. 29, pp. 238–261.

Rosler, U. and Ruschendorf, L. (2001): The Contraction Method for Recursive Algorithms, *Algorithmica*, Vol. 29, pp. 3–33.

Roura, S. (2001): Improved Master Theorems for Divide-and-Conquer Recurrences, *Journal of the ACM*, Vol. 48, No. 2, pp. 170–205.

Rzhetsky, A. and Nei, M. (1992): A Simple Method for Estimating and Testing Minimum-Evolution Tree, *Molecular Biology and Evolution*, Vol. 9, pp. 945–967.

Rzhetsky, A. and Nei, M. (1992): Statistical Properties of the Ordinary Least-Squares; Generalized Least-Squares; and Minimum-Evolution Methods of Phylogenetic Inference, *Journal of Molecular Evolution*, Vol. 35, pp. 367–375.

Sahni, S. (1976): Algorithm for Scheduling Independent Tasks, *Journal of the ACM*, Vol. 23, No. 1, pp. 116–127.

Sahni, S. (1977): General Techniques for Combinatorial Approximation, *Operations Research*, Vol. 25, pp. 920–936.

Sahni, S. and Gonzalez, T. (1976): P-Complete Approximation Problems, *Journal of the ACM*, Vol. 23, pp. 555–565.

Sahni, S. and Wu, S. Y. (1988): Two NP-Hard Interchangeable Terminal Problems, *IEEE Transactions on CAD*, Vol. 7, No. 4, pp. 467–471.

Sakoe, H. and Chiba, S. (1978): Dynamic Programming Algorithm Optimization for Spoken Word Recognition, *IEEE Transactions on Acoustics Speech and Signal Processing*, Vol. 27, pp. 43–49.

Santis, A. D. and Persiano, G. (1994): Tight Upper and Lower Bounds on the Path Length of Binary Trees, *SIAM Journal on Applied Mathematics*, Vol. 23,

No. 1, pp. 12–24.

Sarrafzadeh, M. (1987): Channel-Routing Problem in the Knock-Knee Mode Is NP-Complete, *IEEE Transactions on CAD*, Vol. CAD–6, No. 4, pp. 503–506.

Schmidt, J. (1998): All Highest Scoring Paths in Weighted Grid Graphs and Their Application to Finding All Approximate Repeats in Strings, *SIAM Journal on Computing*, Vol. 27, No. 4, pp. 972–992.

Schwartz, E. S. (1964): An Optimal Encoding with Minimum Longest Code and Total Number of Digits, *Information and Control*, Vol. 7, No. 1, pp. 37–44.

Sedgewick, R. and Flajolet, D. (1996): *An Introduction to the Analysis of Algorithms*, Addison-Wesley, Reading, Mass.

Seiden, S. (1999): A Guessing Game and Randomized Online Algorithms, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, ACM Press, Portland, Oregon, pp. 592–601.

Seiden, S. (2002): On the Online Bin Packing Problem, *Journal of the ACM*, Vol. 49, No. 5, September, pp. 640–671.

Sekhon, G. S. (1982): Dynamic Programming Interpretation of Construction-Type Plant Layout Algorithms and Some Results, *Computer Aided Design*, Vol. 14, No. 3, pp. 141–144.

Sen, S and Sherali, H. D. (1985): A Branch and Bound Algorithm for Extreme Point Mathematical Programming Problem, *Discrete Applied Mathematics*, Vol. 11, No. 3, pp. 265–280.

Setubal, J. and Meidanis, J. (1997): *Introduction to Computational Biology*, PWS Publishing, Boston, Mass.

Sgall, J. (1996): Randomized On-Line Scheduling of Parallel Jobs, *Journal of Algorithms*, Vol. 21, pp. 149–175.

Shaffer, C. A. (2001): *A Practical Introduction to Data Structures and Algorithm Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

Shamos, M. I. (1978): *Computational geometry*. Unpublished Ph.D. dissertation, Yale University.

Shamos, M. I. and Hoey, D. (1975): Closest-Point Problems, *Proceedings of the Sixteenth Annual IEEE Symposium on Foundations of Computer Science*, IEEE Press, Washington, DC, pp. 151–162.

Shamos, M. I. and Hoey, D. (1976): Geometric Intersection Problems, *Proceedings of the Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, IEEE Press, Washington, DC, pp. 208–215.

Shmueli, O. and Itai, A. (1987): Complexity of Views: Tree and Cyclic Schemas, *SIAM Journal on Computing*, Vol. 16, No. 1, pp. 17–37.

Shreesh, J., Asish, M. and Binay, B. (1996): An Optimal Algorithm for the Intersection Radius of a Set of Convex Polygons, *Journal of Algorithms*, Vol. 20, No. 2, pp. 244–267.

Simon, R. and Lee, R. C. T. (1971): On the Optimal Solutions to AND/OR Series-Parallel Graphs, *Journal of the ACM*, Vol. 18, No. 3, pp. 354–372.

Slavik, P. (1997): A Tight Analysis of the Greedy Algorithm for Set Cover, *Journal of Algorithms*, Vol. 25, pp. 237–254.

Sleator, D. D. and Tarjan, R. E. (1983): A Data Structure for Dynamic Trees, *Journal of Computer and System Sciences*, Vol. 26, No. 3, pp. 362–391.

Sleator, D. D. and Tarjan, R. E. (1985a): Amortized Efficiency of List Update and Paging Rules, *Communications of the ACM*, Vol. 28, No. 2, pp. 202–208.

Sleator, D. D. and Tarjan, R. E. (1985b): Self-Adjusting Binary Search Trees, *Journal of the ACM*, Vol. 32, No. 3, pp. 652–686.

Sleator, D. D. and Tarjan, R. E. (1986): Self-Adjusting Heaps, *SIAM Journal on Computing*, Vol. 15, No. 1, February, pp. 52–69.

Smith, D. (1984): Random Trees and the Analysis of Branch-and-Bound Procedures, *Journal of the ACM*, Vol. 31, No. 1, pp. 163–188.

Smith, J. D. (1989): *Design and Analysis of Algorithms*, PWS Publishing,

Boston, Mass.

Snyder, E. E. and Stormo, G. D. (1993): Identification of Coding Regions in Genomic DNA Sequences: An Application of Dynamic Programming and Neural Networks, *Nucleic Acids Research*, Vol. 21, No. 3, pp. 607–613.

Solovay, R. and Strassen, V. (1977): A Fast Monte-Carlo Test for Primality, *SIAM Journal on Computing*, Vol. 6, No. 1, pp. 84–85.

Spirakis, P. (1988): Optimal Parallel Randomized Algorithm for Sparse Addition and Identication, *Information and Computation*, Vol. 76, No. 1, pp. 1–12.

Srimani, P. K. (1989): Probabilistic Analysis of Output Cost of a Heuristic Search Algorithm, *Information Sciences*, Vol. 47, pp. 53–62.

Srinivasan, A. (1999): Improved Approximation Guarantees for Packing and Covering Integer Programs, *SIAM Journal on Computing*, Vol. 29, pp. 648–670.

Srinivasan, A. and Teo, C. P. (2001): A Constant-Factor Approximation Algorithm for Packet Routing and Balancing Local vs. Global Criteria, *SIAM Journal on Computing*, Vol. 30, pp. 2051–2068.

Steel, M. A. (1992): The Complexity of Reconstructing Trees from Qualitative Characters and Subtrees, *Journal of Classification*, Vol. 9, pp. 91–116.

Steele, J. M. (1986): An Efron-Stein Inequality for Nonsymmetric Statistics, *Annals of Statistics*, Vol. 14, pp. 753–758.

Stewart, G. W. (1999): The QLP Approximation to the Singular Value Decomposition, *SIAM Journal on Scientific Computing*, Vol. 20, pp. 1336–1348.

Stoneking, M., Jorde, L. B., Bhatia, K. and Wilson, A. C. (1990): Geographic Variation in Human Mitochondrial DNA from Papua New Guinea, *Genetics*, Vol. 124, pp. 717–733.

Storer, J. A. (1977): NP-Completeness Results Concerning Data Compression (Prove NP-Complete Problem), *Technical Report 233*, Princeton University, Princeton, New Jersey.

Strassen, V. (1969): Gaussian Elimination Is Not Optimal, *Numerische Mathematik*, Vol. 13, pp. 354–356.

Stringer, C. B. and Andrews, P. (1988): Genetic and Fossil Evidence for the Origin of Modern Humans, *Science*, Vol. 239, pp. 1263–1268.

Sutton, R. S. (1990): Integrated Architectures for Learning, Planning and Reacting Based on Approximating Dynamic Programming, *Proceedings of the Seventh International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, California, pp. 216–224.

Sweedyk, E. S. (1995): *A 2 1/2 approximation algorithm for shortest common superstring*. Unpublished Ph.D. thesis, University of California.

Sykora, O. and Vrto, I. (1993): Edge Separators for Graphs of Bounded Genus with Applications, *Theoretical Computer Science*, Vol. 112, No. 2, pp. 419–429.

Szpankowski, W. (2001): *Average Case Analysis of Algorithms on Sequences*, John Wiley & Sons, New York.

Tamassia, R. (1996): On-line Planar Graph Embedding, *Journal of Algorithms*, Vol. 21, pp. 201–239.

Tang, C. Y., Buehrer, D. J. and Lee, R. C. T. (1985): On the Complexity of Some Multi-Attribute File Design Problems, *Information Systems*, Vol. 10, No. 1, pp. 21–25.

Tarhio, J. and Ukkonen, E. (1986): A Greedy Algorithm for Constructing Shortest Common Superstrings, *Lecture Notes in Computer Science*, Vol. 233, pp. 602–610.

Tarhio, J. and Ukkonen, E. (1988): A Greedy Approximation Algorithm for Constructing Shortest Common Superstrings, *Theoretical Computer Science*, Vol. 57, pp. 131–145.

Tarjan, R. E. (1983): Data Structures and Network Algorithms, *SIAM*, Vol. 29.

Tarjan, R. E. (1985): Amortized Computational Complexity, *SIAM Journal on Algebraic Discrete Methods*, Vol. 6, No. 2, pp. 306–318.

Tarjan, R. E. (1987): Algorithm Design, *Communications of the ACM*, Vol. 30, No. 3, pp. 204–213.

Tarjan, R. E. and Van Leeuwen, J. (1984): Worst Case Analysis of Set Union Algorithms, *Journal of the ACM*, Vol. 31, No. 2, pp. 245–281.

Tarjan, R. E. and Van Wyk, C. J. (1988): An $O(n \log \log n)$-Time Algorithm for Triangulating a Simple Polygon, *SIAM Journal on Computing*, Vol. 17, No. 1, pp. 143–178.

Tataru, D. (1992): *Viscosity Solutions for the Dynamic Programming Equations, Applied Mathematics and Optimization*, Vol. 25, pp. 109–126.

Tatman, J. A. and Shachter, R. D. (1990): Dynamic Programming and Influence Diagrams, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20, pp. 365–379.

Tatsuya, A. (2000): Dynamic Programming Algorithms for RNA Secondary Structure Prediction with Pseudoknots, *Discrete Applied Mathematics*, Vol. 104, pp. 45–62.

Teia, B. (1993): Lower Bound for Randomized List Update Algorithms, *Information Processing Letters*, Vol. 47, No. 1, pp. 5–9.

Teillaud, M. (1993): *Towards Dynamic Randomized Algorithms in Computational Geometry*, Springer-Verlag, New York.

Thomassen, C. (1997): On the Complexity of Finding a Minimum Cycle Cover of a Graph, *SIAM Journal on Computing*, Vol. 26, No. 3, pp. 675–677.

Thulasiraman, K. and Swamy, M. N. S. (1992): *Graphs: Theory and Algorithms*, John Wiley & Sons, New York.

Tidball, M. M. and Atman, E. (1996): Approximations in Dynamic Zero-Sum Games I, *SIAM Journal on Control and Optimization*, Vol. 34, No. 1, pp. 311–328.

Ting, H. F. and Yao, A. C. (1994): Randomized Algorithm for Finding Maximum with $O((\log n)^2)$, *Information Processing Letters*, Vol. 49, No. 1, pp. 39–43.

Tisseur, F. and Dongarra, J. (1999): A Parallel Divide and Conquer Algorithm for the Symmetric Eigenvalue Problem on Distributed Memory Architectures, *SIAM Journal on Scientific Computing*, Vol. 20, No. 6, pp. 2223–2236.

Tomasz, L. (1998): A Greedy Algorithm Estimating the Height of Random Trees, *SIAM Journal on Discrete Mathematics*, Vol. 11, pp. 318–329.

Tong, C. S. and Wong, M. (2002): Adaptive Approximate Nearest Neighbor Search for Fractal Image Compression, *IEEE Transactions on Image Processing*, Vol. 11, No. 6, pp. 605–615.

Traub, J. F. and Wozniakowski, H. (1984): On the Optimal Solution of Large Linear Systems, *Journal of the ACM*, Vol. 31, No. 3, pp. 545–549.

Trevisan, L. (2001): Non-Approximability Results for Optimization Problems on Bounded Degree Instances, *ACM*, pp. 453–461.

Tsai, C. J. and Katsaggelos, A. K. (1999): Dense Disparity Estimation with a Divide-and-Conquer Disparity Space Image Technique, *IEEE Transactions on Multimedia*, Vol. 1, No. 1, pp. 18–29.

Tsai, K. H. and Hsu, W. L. (1993): Fast Algorithms for the Dominating Set Problem on Permutation Graphs, *Algorithmica*, Vol. 9, No. 6, pp. 601–614.

Tsai, K. H. and Lee, D. T. (1997): K Best Cuts for Circular-Arc Graphs, *Algorithmica*, Vol. 18, pp. 198–216.

Tsai, Y. T., Lin, Y. T. and Hsu, F. R. (2002): The On-Line First-Fit Algorithm for Radio Frequency Assignment Problem, *Information Processing Letters*, Vol. 84, No. 4, pp. 195–199.

Tsai, Y. T. and Tang, C. Y. (1993): The Competitiveness of Randomized Algorithms for Online Steiner Tree and On-Line Spanning Tree Problems, *Information Processing Letters*, Vol. 48, pp. 177–182.

Tsai, Y. T., Tang, C. Y. and Chen, Y. Y. (1994): Average Performance of a Greedy Algorithm for On-Line Minimum Matching Problem on Euclidean Space, *Information Processing Letters*, Vol. 51, pp. 275–282.

Tsai, Y. T., Tang, C. Y. and Chen, Y. Y. (1996): An Average Case Analysis of a Greedy Algorithm for the On-Line Steiner Tree Problem, *Computers and Mathematics with Applications*, Vol. 31, No. 11, pp. 121–131.

Turner, J. S. (1989): Approximation Algorithms for the Shortest Common Superstring Problem, *Information and Computation*, Vol. 83, pp. 1–20.

Ukkonen, E. (1985a): Algorithms for Approximate String Matching, *Information and Control*, Vol. 64, pp. 10–118.

Ukkonen, E. (1985b): Finding Approximate Patterns in Strings, *Journal of Algorithms*, Vol. 6, pp. 132–137.

Ukkonen, E. (1990): A Linear Time Algorithms for Finding Approximate Shortest Common Superstrings, *Algorithmica*, Vol. 5, pp. 313–323.

Ukkonen, E. (1992): Approximate String-Matching with Q-Grams and Maximal Matches, *Theoretical Computer Science*, Vol. 92, pp. 191–211.

Unger, R. and Moult, J. (1993): Finding the Lowest Free Energy Conformation of a Protein Is an NP-Hard Problem: Proof and Implications (Prove NP-Complete Problem), *Bulletin of Mathematical Biology*, Vol. 55, pp. 1183–1198.

Uspensky, V. and Semenov, A. (1993): *Algorithms: Main Ideas and Applications*, Kluwer Press, Norwell, Mass.

Vaidya, P. M. (1988): Minimum Spanning Trees in k-Dimensional Space, *SIAM Journal on Computing*, Vol. 17, No. 3, pp. 572–582.

Valiant, L. G. and Vazirani, V. V. (1986): NP Is as Easy as Detecting Unique Solutions, *Theoretical Computer Science*, Vol. 47, No. 1, pp. 85–93.

Van Leeuwen, J. (1990): *Handbook of Theoretical Computer Science: Volume A: Algorithms and Complexity*, Elsevier, Amsterdam.

Vazirani, V. V. (2001): *Approximation Algorithms*, Springer-Verlag, New York.

Verma, R. M. (1997): General Techniques for Analyzing Recursive Algorithms with Applications, *SIAM Journal on Computing*, Vol. 26, No. 2, pp. 568–581.

Veroy, B. S. (1988): Average Complexity of Divide-and-Conquer Algorithms, *Information Processing Letters*, Vol. 29, No. 6, pp. 319–326.

Vintsyuk, T. K. (1968): Speech Discrimination by Dynamic Programming, *Cybernetics*, Vol. 4, No. 1, pp. 52–57.

Vishwanathan, S. (1992): Randomized Online Graph Coloring, *Journal of Algorithms*, Vol. 13, pp. 657–669.

Viterbi, A. J. (1967): Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm, *IEEE Transactions on Information Theory*, pp. 260–269.

Vliet, A. (1992): An Improved Lower Bound for On-Line Bin Packing Algorithms, *Information Processing Letters*, Vol. 43, No. 5, pp. 277–284.

Von Haeselerm A., Blum, B., Simpson, L., Strum, N. and Waterman, M. S. (1992): Computer Methods for Locating Kinetoplastid Cryptogenes, *Nucleic Acids Research*, Vol. 20, pp. 2717–2724.

Voronoi, G. (1908): Nouvelles Applications des Parameters Continus a la Theorie des Formes Quadratiques. Deuxieme M'emoire: Recherches Sur les Parall'eloedres Primitifs, *J. Reine Angew. Math.*, Vol. 134, pp. 198–287.

Vyugin, M. V. and V'yugin, V. V. (2002): On Complexity of Easy Predictable Sequences, *Information and Computation*, Vol. 178, No. 1, pp. 241–252.

Wah. B. W. and Yu, C. F. (1985): Stochastic Modeling of Branch-and-Bound Algorithms with Best-First Search, *IEEE Transactions on Software Engineering*, Vol. SE–11, No. 9, pp. 922–934.

Walsh, T. R. (1984): How Evenly Should One Divide to Conquer Quickly? *Information Processing Letters*, Vol. 19, No. 4, pp. 203–208.

Wang, B. F. (1997): Tighter Bounds on the Solution of a Divide-and-Conquer Maximum Recurrence, *Journal of Algorithms*, Vol. 23, pp. 329–344.

Wang, B. F. (2000): Tight Bounds on the Solutions of Multidimensional Divide-and-Conquer Maximum Recurrences, *Theoretical Computer Science*, Vol. 242,

pp. 377–401.

Wang, D. W. and Kuo, Y. S. (1988): A Study of Two Geometric Location Problems, *Information Processing Letters*, Vol. 28, No. 6, pp. 281–286.

Wang, J. S. and Lee, R. C. T. (1990): An Efficient Channel Routing Problem to Yield an Optimal Solution, *IEEE Transactions on Computers*, Vol. 39, No. 7, pp. 957–962.

Wang, J. T. L., Zhang, K., Jeong, K. and Shasha, D. (1994): A System for Approximate Tree Matching, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 4, pp. 559–571, 1041–4347.

Wang, L. and Gusfield, D. (1997): Improved Approximation Algorithms for Tree Alignment, *Journal of Algorithms*, Vol. 25, No. 2, pp. 255–273.

Wang, L. and Jiang, T. (1994): On the Complexity of Multiple Sequence Alignment, *Journal of Computational Biology*, Vol. 1, No. 4, pp. 337–348.

Wang, L., Jiang, T. and Gusfield, D. (2000): A More Efficient Approximation Scheme for Tree Alignment, *SIAM Journal on Applied Mathematics*, Vol. 30, No. 1, pp. 283–299.

Wang, L., Jiang, T. and Lawler, E. L. (1996): Approximation Algorithms for Tree Alignment with a Given Phylogeny, *Algorithmica*, Vol. 16, pp. 302–315.

Wang, X., He, L., Tang, Y. and Wee, W. G. (2003): A Divide and Conquer Deformable Contour Method with a Model Based Searching Algorithm, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 33, No. 5, pp. 738–751.

Wareham, H. T. (1995): A Simplified Proof of the NP- and MAX SNP-Hardness of Multiple Sequence Tree Alignments (Prove NP-Complete Problem), *Journal of Computational Biology*, Vol. 2, No. 4.

Waterman, M. S. (1995): *Introduction to Computational Biology: Maps, Sequences and Genomes*, Chapman & Hall/CRC, New York.

Waterman, M. S. and Smith, T. F. (1978): RNA Secondary Structure: A Complete Mathematical Analysis, *Mathematical Bioscience*, Vol. 42, pp. 257–266.

Waterman, M. S. and Smith, T. F. (1986): Rapid Dynamic Programming Algorithms for RNA Secondary Structure, *Advances in Applied Mathematics*, Vol. 7, pp. 455–464.

Waterman, M. S. and Vingron, M. (1994): Sequence Comparison Significance and Poisson Approximation, *Statistical Science*, Vol. 2, pp. 367–381.

Weide, B. (1977): A Survey of Analysis Techniques for Discrete Algorithms, *ACM Computing Surveys*, Vol. 9, No. 4, pp. 291–313.

Weiss, M. A. (1992): *Data Structures and Algorithm Analysis*, Benjamin Cummings, Redwood City, California.

Wenger, R. (1997): Randomized Quickhull, *Algorithmica*, Vol. 17, No. 3, pp. 322–329.

Westbrook, J. and Tarjan, R. E. (1989): Amortized Analysis of Algorithms for Set Union with Backtracking, *SIAM Journal on Computing*, Vol. 18, No. 1, pp. 1–11.

Wilf, H. S. (1986): *Algorithms & Complexity*, Prentice-Hall, Engelwood Cliffs, New Jersey.

Williams, J. W. J. (1964): Heapsort: Algorithm 232, *Communications of the ACM*, Vol. 7, pp. 347–348.

Wood, D. (1993): *Data Structures, Algorithms and Performance*, Addison-Wesley, Reading, Mass.

Wright, A. H. (1994): Approximate String Matching Using Withinword Parallelism, *Software: Practice and Experience*, Vol. 24, pp. 337–362.

Wu, B. Y., Lancia, G., Bafna, V., Chao, K. M., Ravi, R. and Tang, C. Y. (2000): A Polynomial-Time Approximation Scheme for Minimum Routing Cost Spanning Trees, *SIAM Journal on Computing*, Vol. 29, No. 3, pp. 761–778.

Wu, L. C. and Tang, C. Y. (1992): Solving the Satisability Problem by Using Randomized Approach, *Information Processing Letters*, Vol. 41, No. 4, pp. 187–190.

Wu, Q. S., Chao, K. M. and Lee, R. C. T. (1998): The NPO-Completeness of the Longest Hamiltonian Cycle Problem, *Information Processing Letters*, Vol. 65, pp. 119–123.

Wu, S. and Manber, U. (1992): Fast Text Searching Allowing Errors, *Communications of the ACM*, Vol. 35, pp. 83–90.

Wu, S. and Myers, G. (1996): A Subquadratic Algorithm for Approximate Limited Expression Matching, *Algorithmica*, Vol. 15, pp. 50–67.

Wu, T. (1996): A Segment-Based Dynamic Programming Algorithm for Predicting Gene Structure, *Journal of Computational Biology*, Vol. 3, pp. 375–394.

Wu, Y. F., Widmayer, P. and Wong, C. K. (1986): A Faster Approximation Algorithm for the Steiner Problem in Graphs, *Acta Informatica*, Vol. 23, No. 2, pp. 223–229.

Xu, S. (1990): *Dynamic programming algorithms for alignment hyperplanes*. Unpublished Master's thesis, University of Southern California.

Yagle, A. E. (1998): Divide-and-Conquer 2-D Phase Retrieval Using Subband Decomposition and Filter Banks, *IEEE Transactions on Signal Processing*, Vol. 46, No. 4, pp. 1152–1154.

Yang, C. I., Wang, J. S. and Lee, R. C. T. (1989): A Branch-and-Bound Algorithm to Solve the Equal-Execution-Time Job Scheduling Problem with Precedence Constraint and Prole, *Computers and Operations Research*, Vol. 16, No. 3, pp. 257–269.

Yannakakis, M. (1985): A Polynomial Algorithm for the Min-Cut Linear Arrangement of Trees, *Journal of the Association for Computing Machinery*, Vol. 32, No. 4, pp. 950–988.

Yannakakis, M. (1989): Embedding Planar Graphs in Four Pages, *Journal of Computer and System Sciences*, Vol. 38, No. 1, pp. 36–67.

Yao, A. C. (1981): Should Tables be Sorted, *Journal of the ACM*, Vol. 28, No. 3, pp. 615–628.

Yao, A. C. (1985): On the Complexity of Maintaining Partial Sums, *SIAM Journal on Computing*, Vol. 14, No. 2, pp. 277–288.

Yao, A. C. (1991): Lower Bounds to Randomized Algorithms for Graph Properties, *Journal of Computer and System Sciences*, Vol. 42, No. 3, pp. 267–287.

Ye, D. and Zhang, G. (2003): On-Line Extensible Bin Packing with Unequal Bin Sizes, *Lecture Notes in Computer Science*, Vol. 2909, pp. 235–247.

Yen, C. C. and Lee, R. C. T. (1990): The Weighted Perfect Domination Problem, *Information Processing Letters*, Vol. 35, pp. 295–299.

Yen, C. C. and Lee, R. C. T. (1994): Linear Time Algorithm to Solve the Weighted Perfect Domination Problem in Series-Parallel Graphs, *European Journal of Operational Research*, Vol. 73, No. 1, pp. 192–198.

Yen, C. K. and Tang, C. Y. (1995): An Optimal Algorithm for Solving the Searchlight Guarding Problem on Weighted Trees, *Information Sciences*, Vol. 87, pp. 79–105.

Yen, F. M. and Kuo, S. Y. (1997): Variable Ordering for Ordered Binary Decision Diagrams by a Divide-and-Conquer Approach, *IEE Proceedings: Computer and Digital Techniques*, Vol. 144, No. 5, pp. 261–266.

Yoo, J., Smith, K. F. and Gopalarkishnan, G. (1997): A Fast Parallel Squarer Based on Divide-and-Conquer, *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 6, pp. 909–912.

Young, N. (2000): On-Line Paging Against Adversarially Biased Random Inputs, *Journal of Algorithms*, Vol. 37, pp. 218–235.

Younger, D. H. (1967): Recognition and Parsing of Context-Free Languages in Time n^3, *Information and Control*, Vol. 10, No. 2, pp. 189–208.

Zelikovsky, A. (1993): An 11/6 Approximation Algorithm for the Steiner Tree Problem in Graph, *Information Processing Letters*, Vol. 46, pp. 317–323.

Zemel, E. (1987): A Linear Randomized Algorithm for Searching Rank

Functions, *Algorithmica*, Vol. 2, No. 1, pp. 81–90.

Zhang, K. and Jiang, T. (1994): Some Max SNP-Hard Results Concerning Unordered Labeled Trees, *Information Processing Letters*, Vol. 49, pp. 249–254.

Zhang, Z., Schwartz, S., Wagner, L. and Miller, W. (2003): A Greedy Algorithm for Aligning DNA Sequences, *Journal of Computational Biology*, Vol. 7, pp. 203–214.

Zuker, M. (1989): The Use of Dynamic Programming Algorithms in RNA Secondary Structure Prediction, in M. S. Waterman (Ed.), *Mathematical Methods for DNA Sequences* (pp. 159–185), CRC Press, Boca Raton, Florida.