世界大学生程序设计竞赛（ACM/ICPC）高级教程

第二册

程序设计中常用的解题策略

吴文虎　王建德　编著

ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST ADVANCED COURSE

THE COMMON MODE OF PROBLEM SOLVING STRATEGIES

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

世界大学生程序设计竞赛（ACM/ICPC）高级教程
第二册
程序设计中常用的解题策略

吴文虎　王建德　编著

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

内 容 简 介

本书是针对世界大学生程序设计竞赛（ACM/ICPC）而编写的第二本参考书。

ACM/ICPC 是大学生智力与计算机解题能力的竞赛，是世界公认的最具影响力的、规模最大的国际顶级赛事，被称为大学生的信息学奥林匹克。

第一册主要介绍程序设计中解题的常用思维方式。本书是第一册的继续，只是换了一个角度，分 4 方面介绍解题策略：数据关系上的构造策略；数据统计上的二分策略；动态规划中的优化策略；计算几何题的应对策略。

本书面向参加世界大学生程序设计竞赛（ACM/ICPC）的高等院校学生，也可作为程序设计爱好者的参考用书。

图书在版编目（CIP）数据

世界大学生程序设计竞赛(ACM/ICPC)高级教程. 第2册，程序设计中常用的解题策略/吴文虎，王建德编著
.—北京：中国铁道出版社，2012.7
ISBN 978-7-113-14605-4

Ⅰ.①世… Ⅱ.①吴…②王… Ⅲ.①程序设计—竞赛—高等学校—自学参考资料 Ⅳ.①TP311.1

中国版本图书馆 CIP 数据核字（2012）第 083317 号

书　　名：世界大学生程序设计竞赛（ACM/ICPC）高级教程　第二册　程序设计中常用的解题策略
作　　者：吴文虎　王建德　编著

策划编辑：秦绪好　　**读者热线：400-668-0820**
责任编辑：翟玉峰
编辑助理：赵　迎
封面设计：付　魏
封面制作：刘　颖
责任印制：李　佳

出版发行：中国铁道出版社（100054，北京市西城区右安门西街 8 号）
网　　址：http://www.51eds.com
印　　刷：北京铭成印刷有限公司
版　　次：2012 年 7 月第 1 版　　2012 年 7 月第 1 次印刷
开　　本：787 mm×960 mm　1/16　　**印张：14**　　**字数：304 千**
印　　数：1～4 000 册
书　　号：ISBN 978-7-113-14605-4
定　　价：48.00 元

前 言
FOREWORD

ACM/ICPC 是国际计算机协会（Association for Computing Machinery）组织的国际大学生程序设计竞赛（International Collegiate Programming Contest）的英文简称。这项赛事始于 1976 年的计算机学科竞赛，随着信息科技的迅猛发展，世界各国对计算机科学教育重要性的认识日益加深，该项赛事已经演变成为目前规模最大和最具影响力的全球性的高等学校之间的盛会。

这项一年一届的赛事从当年的 9 月开始，先进行各大洲地区性的预选赛，从近 2 000 所大学的 8 000 多个参赛队的预选赛中选拔出 100 个左右的队于第二年的春季参加全球总决赛。比赛强调团队精神与合作协同攻关能力，3 个学生组成一个队，共用一台计算机，一起解决 10 道左右的难题。这些题目涉及 Direct（简单题）、Computational Geometry（计算几何）、Number Theory（数论）、Combinatorics（组合数学）、Search Techniques（搜索技术）、Dynamic Programming（动态规划）、Graph Theory（图论）和 Other（其他）。可使用的计算机编程语言为：C++、C、Java 和 Pascal。但 final 赛只可以使用 C 或 C++。

赛题特点如下：

① 有实际背景，趣味性和实用性较强。

② 考查的知识范围比较全面（基础的与深层次的题目都有）。

③ 层次性较好，10 道题中会有不同水平的题。

④ 灵活、新颖。绝大部分没有固定解法，留有广阔的思维空间，有益于培养学生的创造能力。

从历年的赛题看，难度很大，涉及数学、物理、电子学、计算机科学等多种学科，特别是要用到数理逻辑、图论、集合论、组合数学、概率论、计算几何等数学知识和计算机的高效算法。在比赛现场的审题建模、构思算法、编码调试、自我测试、快速纠错的全面能力，对每一个参赛队而言都是巨大的挑战。近年，尽管已是能够进入决赛圈的队，有的也只能解对 11 道赛题中的两三道。

ACM/ICPC 这项国际顶级赛事比的是智力和计算机综合解题能力的

高低，赛场是大学生展示水平与才华的大舞台，是著名的高等学府计算机教育成果的直接体现，同时也是 IT 企业与世界顶尖计算机人才对话的最佳机会。因而吸引了越来越多的高校参赛，使得参赛队伍的水平上升很快，赛题的难度也在不断提高。

中国的大学参加这项赛事已有 15 年的历史，现在已有 100 多所学校的几百支队伍参加亚洲区的预选赛，每年都会有 10 多所学校进军总决赛，成绩是很好的。

ACM/ICPC 赛事属于因材施教活动，它是和国际接轨的，学生参加比赛是一个学习、观摩、交流、开阔眼界、考验心理素养和提高竞争能力的过程。特别是在与编程高手过招的时候，可以把知识运用的综合性、灵活性和探索性水平发挥到极致，体验和感受数学思维和算法艺术之美，在实践中提升科学思维能力。

科学思维能力的提高是学生今后成就事业的一个非常重要的因素，我们希望这本书能够对读者有所帮助。

王建德老师与我愉快地合作了 20 年，在本书的策划和写作中，王建德老师一如既往地花费了许多心血，总结出十分精彩的观点和思路。

清华大学的一些选手曾试用和验证过原稿中的某些算法，邓俊辉博士、徐明星博士、邬晓钧博士和朱全民老师对原稿提出过宝贵意见，在此一并感谢。

2009 年本书第一册（共 6 章，即第 1～6 章）出版后，得到许多读者的关怀。这里特别要感谢周广声老师，他对本书第一册中的一些用词和某些基本概念的陈述提出了非常中肯的修改意见。第二册共 5 章，即第 7～11 章。

限于作者的学识和水平，难免还会有疏漏和不足之处，敬请读者提出宝贵意见和建议。

清华大学计算机系教授，博士生导师
原国际信息学奥林匹克中国队总教练

吴文虎

2012 年 6 月

本书导学

ABSTRACT

策略是指把握总体行事的方针，而非具体方法。心理学家认为，在解决问题的过程中，如果主体所接触到的不是标准模式化了的问题，就需要进行创造性的思维，研究解决问题的“策略”。程序设计的解题策略，是指编程解题时所采取的一种基本方略，是带有全局性、概括性、综合性的思路。

思维方式和解题策略是相互联系的。从某种意义上讲，第一册所述的思维方式也是解题策略，而本册所述的解题策略也处处渗透着前述的思维方式。第一册主要是从思维方式的角度谈解题方法，而本册则侧重从行为特征的角度来谈，两册论述的角度有所不同，但目标是一致的。为了使读者对编程解题的策略有一个全面的了解，我们将按照题型和知识的分类，从 4 个方面加以讨论：

① 数据关系上的构造策略。本册介绍一些特殊类型的树和图：例如在树结构上，探讨树的划分问题、最小生成树、线段树及其扩展形式、伸展树和左偏树。利用这些特殊类型的树可优化存储结构和算法效率。另外，还引进了一种可取代树结构、且不失时空效率、容易编程实现的线性表——“跳跃表”；在图结构上，介绍利用网络流算法、匹配算法、分层图思想、平面图性质和偏序集模型解题的思路和方法，探讨选择图论模型和优化算法的基本策略。

本册在丰富数据结构知识的基础上，围绕如何合理选择数据结构来优化算法的问题，阐述构造数据关系的基本原则和方法。

② 数据统计上的二分策略。在数据统计问题上，将分治思想与相应的数据结构相结合，使得统计过程尽可能模式化，以达到提高效率的目的。

③ 动态规划上的优化策略。决定动态规划时间复杂度有 3 个因素：状态总数、每个状态转移的状态数、每次状态转移的时间。我们围绕这 3 方面，讨论优化的思路和方法。

④ 计算几何问题上的应对策略。几何题一般有 3 种类型：纯粹计算题、存在性问题和最佳值问题。我们结合实例介绍应对每种类型试题的基本策略，其中穿插计算平面多边形、空间长方体、半平面交和最大子矩形的基本方法。

由于读者大都熟悉搜索，况且第一册已经介绍了缩小搜索范围、确定搜索顺序、合理剪枝等优化策略，因此本册略去了对搜索问题的讨论。

我们以上述 4 个方面为基本构件，介绍了几十种解题策略和重要算法。对

每种解题策略和算法的原理进行了必要的分析和证明，定理证明大多采用初等数学的分析方法，公式推导尽可能做到浅显和详细，并给出了计算时间的详细分析。为了帮助读者理解，对其中一些复杂的解题策略和算法附加了图示，使其过程更加具体、直观和形象，每种解题策略和算法都有具体的应用例证。全书共解析了 72 道例题，大部分例题采用“一题多解”、“多向求解”的方式进行解析，并且尽量结合实例讲述一些常用的思维方式和解题策略，以拓宽思路，使读者学会应该怎样应用算法知识来解题，以及应该怎样选择有效的算法。

读者在学习各种解题策略时，需要注意以下 3 个方面的问题：

① 梳理良好的认知结构。注重相关理论的学习，通过不断应用得以强化，形成一个脉络分明、纵横交错的知识网络。同时注意把一些常用的解题技巧和变换方法放在记忆库里，把同类问题“贮存在一起”，使知识条理化。

② 提高解题能力。平时要多看书和多解题，从书中和编程实践中寻找再发现、再创造的契机。既要注意运算结果的正确性，也要注意知识产生的过程性(概念、法则被概括的过程，数据关系被抽象的过程，解题思维被探索的过程)。要细化书本知识，如同电视屏幕中体育大赛的慢镜头式的分解，真正使自己学有所得。

③ 注重解题策略的归类分析。就某个方面的解题策略而言，其形成过程是可以逻辑化、模式化的。因此，要多考虑将解题策略归类，可以选取一些具有代表性例题，进行有系统的、集中的分类解题策略训练，形成一套局部范围内的逻辑化、模式化的解题策略方案。

编程解题离不开解题方法，解题的成功在很大程度上依赖于选择适宜的方法，而最适宜的方法来源于正确的解题策略。

目 录
CONTENTS

第7章 利用树状结构解题的策略

树是一个具有层次结构的集合，一种限制前件数且没有回路的连通图。在 ACM/ICPC 竞赛中，许多问题的数据关系呈树状结构，因此有关树的概念、原理、操作方法和一些由树的数据结构支持的算法，一直受到选手的重视，被广泛应用于解题过程。本章将探讨树的划分问题的策略，并介绍 4 种特殊的树状结构：

① 最小生成树及其扩展形式。

② 线段树及其扩展形式。

③ 伸展树。

④ 左偏树。

利用这些特殊类型的树可优化存储结构和提升算法效率。

最后，将介绍一种在某种情况下可取代树结构、且不失时空效率、容易编程实现的线性表——“跳跃表”。

7.1 解决树的最大—最小划分问题的一般方法

树的最大—最小划分问题可以表述为如下形式：

给定一棵 n 个结点的树以及每个结点的非负权值，要求将这棵树划分为 k 棵子树，使得其中权值和最小的那棵子树最大。即定义第 i 棵子树中所有结点的权值和为 $\mathrm{sum}(i)$，每一种划分方案对应一个 x 值，$x=\min\limits_{1\leqslant i\leqslant k}\{\mathrm{sum}(i)\}$。求 x 最大值时对应的划分方案。

如今在各类竞赛中，类似树的最大 - 最小划分问题愈来愈多，表现形式也愈来愈多样。下面看一个例子。

【例题7.1】划分乡镇

已知某县有 n 个村庄（$1\leqslant n\leqslant 100$，其中县城可视为一个村庄），县城通往每一个村庄有且仅有一条路径。假定已经掌握了每个村庄的人数及村庄之间的连接情况，现需要将该县划分为 k 个

（$1<k<n$）乡。令 sum(i)表示第 i 个乡中所有村庄的人数和（$1\leqslant i\leqslant k$），x 为最小乡的人数，即 $x=\min\limits_{1\leqslant i\leqslant k}\{\text{sum}(i)\}$。你的任务就是寻找满足如下条件的划分方案：

① 每一乡中的村庄必然是直接或者间接的和其他村庄相连接。

② 这种划分方案所对应的 x 尽可能的大。

要求输出你所做的划分方案。

思路点拨

我们将村庄作为结点，村庄的人数作为结点的权值，可以得到一棵以县城为根的树。显然，划分乡镇问题对应这棵树的最大—最小划分问题。

解决这类问题有多种方法，其中比较典型的解法有两种：

① 将原问题转化为在给定下界的情况下划分最多子树的问题。

② 移动“割”的算法。

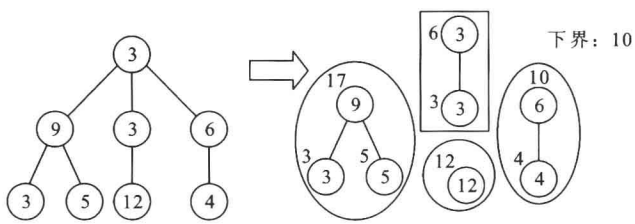
如果两种解法巧妙结合，则可以得到更优化的算法。

解法 1：将原问题转化为在给定下界时划分最多子树的问题

初次面对这个问题，或许感到不知所措，无从下手。动态规划、贪心等方法似乎都不适用，因为不知道这个最小值是多少。这时，不妨先考虑这样一个问题：对于一个确定的下界，将树划分为若干棵子树，使得每棵子树的权值和都不小于此下界。

如果可以解决这个问题，便可以再通过对下界进行二分查找，求得原问题的解。事实上，只需要一个以贪心思想为基础的下述扫描算法：

按照深度由大至小的顺序扫描整棵树，对于扫描到的每个结点，计算以此结点为根的完全子树的权值和。若权值和大于等于给定下界，则将以这个结点为根的子树划分出来，并将其从原树中删去…，直至整棵树所剩部分的权值和小于下界，再将剩余部分归入最后划分出去的子树中，最终得到的就是一种划分数目最多的最优划分。例如，图 7-1（a）给出了一棵含 8 个结点的树，圆圈内的数字为结点的权值，设定下界为 10。按照自下而上的顺序扫描这棵树，依次删除权值和为 12、17、10 的子树，所剩部分的权值和为 6（见图 7-1（b），圆内的子树被删除，方框内为剩余子树）。显然，这棵树在下界为 10 的情况下是无法划分的，因为存在一个权值为 12 的结点。



(a) (b)

图 7-1 含 8 个结点的树

算法正确性的证明比较简单，这里省略。可以看出，算法的时间复杂度是线性的，已经到达了理论的下界。由此得出解法 1：

通过二分法来找到最大的下界 x，使得划分的最大子树数目不小于 k，x 即原问题的解。

虽然找到了一种解决问题的途径，但并不能说问题已经完美解决了。当我们对算法进行深入分析的时候，发现算法效率是依赖于结点权值的。更加确切地说，如果每个结点权值的上界是 c，那么算法的时间复杂度就是 $O(n \times \log_2(n \times c))$。虽然在大多数情况下，程序的实际运行效率是比较好的，但是当结点的权值范围很大或权值是实数时，算法便不能令人满意了。于是，我们自然而然地想到：是否可以找到一个计算时间不依赖于结点权值的算法呢？

解法 2：移动“割”

在某一种划分中，如果一条边所连接的两个结点属于两个不同的子树，那么就称：在这条边上有一个“割”。注意到，每一个割对应一棵子树，这棵子树包括割下方的所有结点去掉其他割下方的结点后剩余的结点，而只有根结点所在的子树没有与之对应的割。例如图 7-2 中，割 1 对应子树的权和为 1+9，割 2 对应子树的权为 12，割 3 对应子树的权和为 3+8，根所在子树的权和为 7+9。

于是问题就可以转化为：如何将 k-1 个割分配到 k-1 条边上，使其满足人们期待的最优条件。

这样，我们换了一种思维方式，把讨论的重点由划分点转化为分配割，希望能够通过这种转化找到解决问题的新途径。

下面介绍一种被称为“移动”的技术，一次移动被定义为“将一个割从边 1 移到另一条与其相邻的边 2 上，并且保证边 2 一定是在边 1 的下一层”（见图 7-3）。也就是说，移动总是由上至下的。这样，我们就有可能通过一个给定的初始划分状态和一系列有限次的移动最后达到某个目标状态。

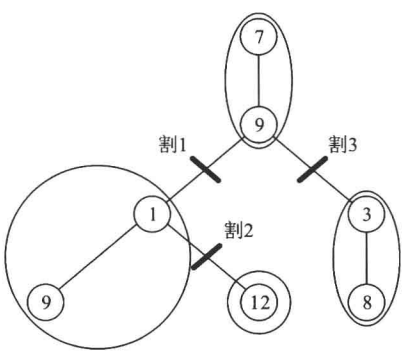


图 7-2　4 个子数有 3 个“割”

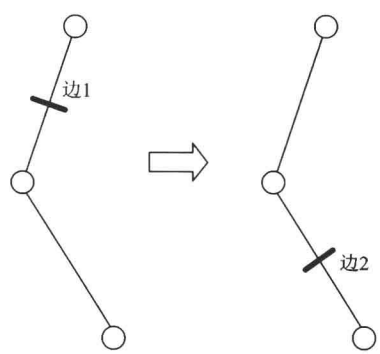


图 7-3　将一个“割”从边 1 下移至与其相邻的边 2 上

初始状态的选择并不困难，可以任取一个度为 1 的结点为根，然后在初始时将所有的割都放到唯一与根结点相连的那条边上。这样，便有可能由初始状态达到任何一个我们想要的目标状态。问题的关键是，人们该如何制定每次移动的规则，使得终止时会达到最小子树最大的要求。而实际上，规则的制定远比人们想象的要简单许多，人们依据的还是一种贪心思想：

在进行每一步时，考虑所有可能的移动，并计算出每一种移动后在新位置的割所对应的新子树的权值和，取出其中新权值和最大的那一种，与当前未移动时子树的最小权值和进行比较，若

不小于当前的最小值，则进行这步移动，否则算法就结束了。具体算法流程如下：

① 选择任何一个度为 1 的结点作为树的根，将 k–1 个割都放在与根相连的唯一一条边上。

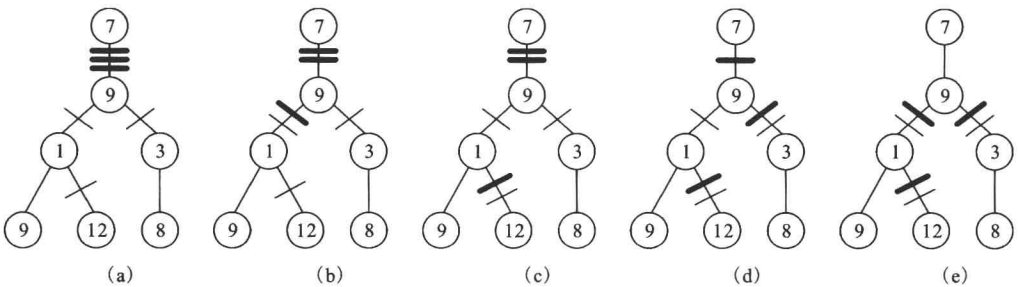
② 计算出当前划分状态下的子树权值和的最小值 W_{min}。

③ 考虑所有可能的移动，找出能使移动后的割所对应的子树权值和最大（W_{now}）的一种移动。

④ 如果 $W_{now} \geq W_{min}$，则进行这步移动，并转到步骤②。

⑤ 算法结束，W_{min} 即所求树值和的最小值，当前划分即为一种最优划分。

上述算法流程即为解法 2。这个算法用到了很多的新东西，我们无法在第一时间内用直觉来确定它是对的，需要对它进行更多的研究和分析，从理论上证明其正确性。在证明算法之前，我们用图示的方法观察移动“割”的过程。图 7–4 中，k=4，要求将 k–1=3 个割（粗线）分配到 3 条边上，使其满足最优条件（细线为最优方案）。



注：━━ 表示由算法进行而得到的划分；── 表示一种最优划分。

图 7–4　3 个“割”的分步移动过程，使其满足最优条件

按照上述算法，图 7–4（a）给出了最初 3 个割的位置，W_{min}=7。第 1 个割左下移可使得对应子树的权值和最大（图 7–4（b），W_{now}=12+1+9=22），再向右下移也满足对应子树最大权值和大于等于 W_{min} 的条件（图 7–4（c），W_{now}=12）；第 2 个割应右下移一步（图 7–4（d），W_{now}=3+8=11）；第 3 个割应左下移一步（图 7–4（e），W_{now}=1+9=10）。至此 W_{min}=1+9=10，移动过程结束。

在观察了图 7–4 后，我们发现了一个很有趣的事实：对于算法进行的每一步移动，新产生的划分状态总是在某个最优划分的“上方”，而每次操作都是将当前的划分“向下”移动一些，最后直到和某个最优划分重合为止。这就为我们的证明提供了一个很不错的思路：

我们可以在划分之间定义一种“上方”关系，然后证明算法进行中的每一种划分状态都是在某个最优划分的上方，而当一种划分是在某个最优划分上方的时候，算法一定会继续。这样，就证明了原算法的正确性。

于是，我们便试图对划分之间的“上方”关系做一个定义。自然而然的想法就是：划分 A 在划分 A'的上方，也就是存在一种 A 的割和 A'的割的一一对应，使得每个 A 的割都在它所对应的 A' 的割的上方。

这种定义能够形象地表现出上方的含义，不失为一种不错的定义方法，但为了在证明中更好地应用“上方”这种关系，我们还希望能够找到更加实用的性质。在这之前，先定义部分子树的概念：

部分子树的概念：若一棵 T 的子树 T'包含了结点 v 连同 v 的某一个儿子以及这个儿子的所有后继，则称 T'是 T 在结点 v 处的一棵部分子树。与 v 相连的唯一一条边称为 T'的初始边（见图 7-5）。

接下来，考察在树 T 上的两个划分 A 和 A'：若 A 在 A'上方，则对于任意一棵 T 的部分子树，若 A 中有一个割 c 在子树上，因为这个割所对应的 A'中的割 c'是在 c 的下方，所以 c'也一定在这棵部分子树上。如果将划分 A 在一棵部分子树上的割的数目表示为#(A)，得到如下性质：

部分子树的性质：若划分 A 在 A'上方，则对于树 T 的任意一棵部分子树，都有#(A)≤#(A')（见图 7-6）。

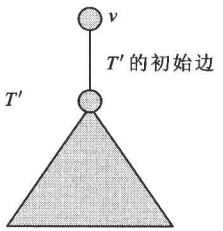


图 7-5　树 T 在结点 v 处的一棵子数 T'

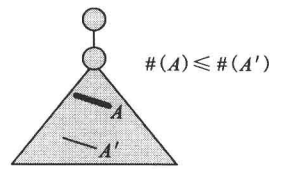


图 7-6　树 T 的任意一棵部分子树

这条性质在下面的证明过程中起到了非常重要的作用。

下面便进入解法 2 的最核心部分。为了证明算法的正确性，将试图证明如下几点：

① 在初始状态时的划分 A 是在任何一个最优划分 Q 的上方的。

② 若存在一个最优划分 Q 使得当前的划分 A 是在 Q 的上方，且 A 和 Q 不相等，则算法一定不会终止。

③ 设 A 在 Q 的上方且 A 不等于 Q，在算法进行一步后，A 变为 A'，一定还能找到一个最优划分 Q'，使得 A'在 Q'上方。

④ 算法会在有限步内终止，算法终止时的划分一定是一个最优划分。

证明：

①的正确性是很显然的，需要证明的是②、③、④。注意：字母 A 通常表示当前划分，字母 Q 表示一个最优划分，即对应最小子树权值最大的划分。用 $W_{\min}(A)$表示在当前划分下 A 的最小子树的权值。显然，对于任意一个划分 A，都有 $W_{\min}(A)\leqslant W_{\min}(Q)$。

②的证明过程：因为 A 和 Q 不相等，所以在 A 中必存在一个割使得在同一条边上没有 Q 的割，令 c 为满足此条件的深度最大的割，由于在以 c 所在的边为初始边的部分子树上，#(A)≤#(Q)，且 c 所在的边上没有 Q 的割，所以在子树中必存在一个 Q 的割 s，使得在同一条边上没有 A 的割（见图 7-7（a））。取 s 上方遇到的第一个 A 中的割 c'，将 c'向 s 的方向移动一步（图 7-7（b）～（c）），c'对应的新子树含 $v_5v_8v_9$，s 对应的子树为 v_8。由于 Q 是最优划分，c'所对应的新子树的权值和≥s 所对应子树的权值和≥$W_{\min}(Q)\geqslant W_{\min}(A)$，所以算法一定会继续，且移动后的割所对应的新子树的权值和一定不小于 $W_{\min}(Q)$（图 7-7（c）～（e））。证毕。

上面的证明还得出了另一个结论：

每次经过算法移动后的割所对应的新子树的权值和一定不小于 $W_{\min}(Q)$。

这个结论将在③的证明中再次用到。

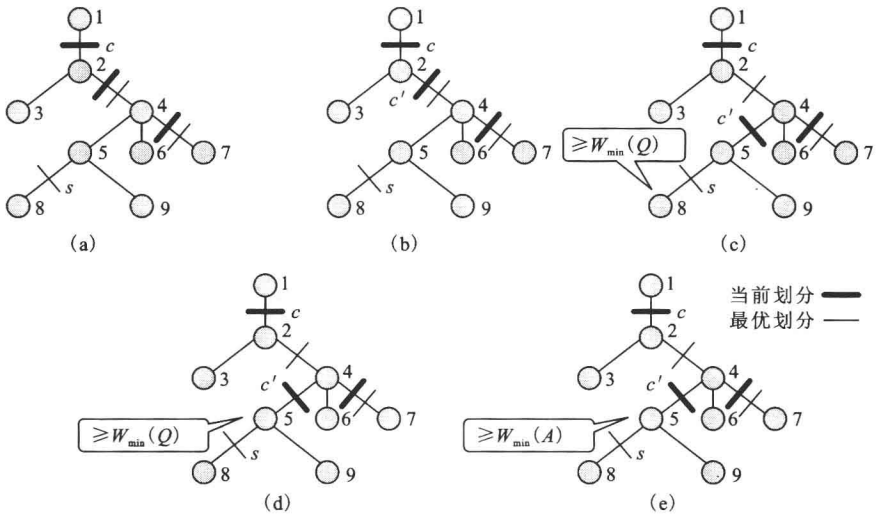


图 7-7 ②的证明过程用图

③的证明过程：不妨假设算法进行的移动是将一个割 c 从边 e_1 移动到了 e_2，设边 e_1 与 e_2 交于点 v。为了构造 Q'，分以下两种情况进行讨论：

情况 1：对于在结点 v 处的每一棵部分子树 T'，都有$\#(A)=\#(Q)$

例如图 7-8（a）中，结点 2 有两棵部分子树，在结点 2 连同以结点 3 为根的部分子树中，$\#(A)=\#(Q)=1$；结点 2 连同以结点 4 为根的部分子树中，$\#(A)=\#(Q)=2$。当前割从边 e_1 移动到 e_2 后（见图 7-8（b）），对于以 e_2 为初始边的部分子树有$\#(A')=3$，$\#(Q)=2$。由于在以 e_1 为初始边的部分子树中$\#(A)=4$，$\#(Q)=3$，故在边 e_1 上必有 Q 的一个割，不妨设为 s。将 s 从 e_1 移动到 e_2，形成 Q'（见图 7-8（c））。这样在以 e_2 为初始边的部分子树 T'中仍有$\#(A')\leqslant\#(Q')$，故 A'仍是在 Q'上方的。所以在 T'中，s 所对应的子树一定是包含 c 所对应的子树的。s 所对应子树的权值和$\geqslant c$ 对应子树的权值和$\geqslant W_{\min}(Q)$（②的证明中得出的结论），Q'也是一个最优划分，且 A'是在 Q'上方。

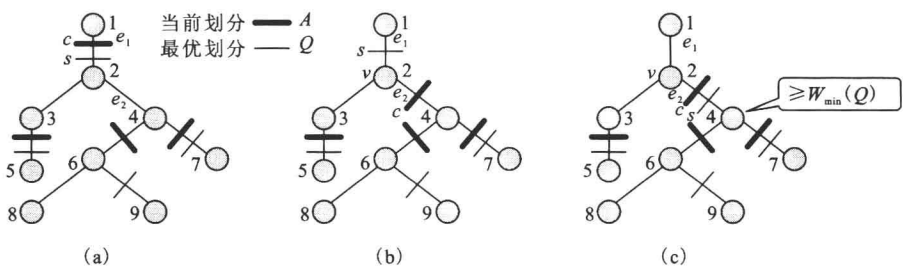


图 7-8 ③的情况 1 证明过程用图

情况 2：存在结点 v 处的某棵子树 T'，使得$\#(A)<\#(Q)$

例如图 7-9（a）中，结点 2 有两棵部分子树，在结点 2 连同以结点 3 为根的部分子树中，$\#(A)=1$，$\#(Q)=2$。将一个割 c 从边 e_1 移动到了 e_2（见图 7-9（b））。如果在以 e_2 为初始边的子树中有$\#(A)<\#(Q)$，

则 $W_{\min}(A')\geqslant W_{\min}(Q)$，即 Q 就是符合条件的划分；若在以 e_2 为初始边的子树中#(A)=#(Q)，则可以设存在一条从 v 出发的边 e_3 使得在以 e_3 为初始边的部分子树 T'中#(A)<#(Q)（见图 7-9（a））。设 s 为 Q 在 T'中深度最小的割（见图 7-9（b）），将 s 移至 e_2 处，构成划分 Q'（见图 7-9（c）），则 A' 是在 Q'的上方的。s 原来对应的子树含 v_5v_9，新对应的子树含 $v_4v_7v_{10}$。与情况 1 一样，s 所对应的新子树的权值和一定大于等于 $W_{\min}(Q)$。而对于结点 v 所在的子树来说，由于其包含了 s 原来对应的子树，因此 s 原对应子树的权值和也一定大于等于 $W_{\min}(Q)$（见图 7-9（d））。由于由 Q 变至 Q' 所改变的只有这两棵子树的情况，故 Q'也是一个最优划分，且 A'在 Q'上方。证毕。

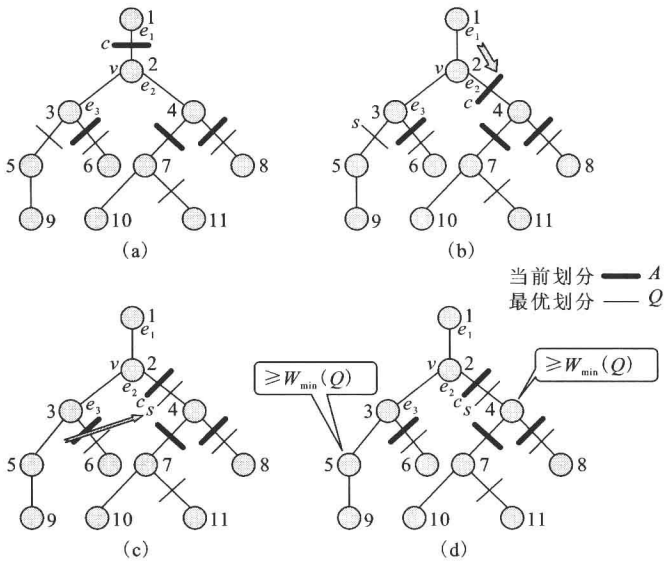


图 7-9　③的情况 2 证明过程用图

④的证明过程：首先，每步移动都是由上至下的，这些割不会被无限移动下去，故算法会在有限步内终止。若算法终止时不是最优划分，由③可知，一定存在一个最优划分 Q 使得当前划分 A 是在 Q 的上方。又因为 A 和 Q 不相等（A 不是最优划分），由②可知，算法一定还会继续，与算法终止矛盾，故算法终止时的划分一定为最优划分。

至此，算法的正确性已经证明完毕。回顾整个证明过程，“上方”的概念自始至终都起着非常重要的作用，而实际上，“上方”的概念就是一种序的关系。通过引入“上方”的概念，将状态间看似杂乱无章的关系变得有序化、有条理性，从而解决问题，这在 ACM/ICPC 竞赛中是很值得借鉴的。

再来关注一下算法的时间复杂度，虽然算法的描述并不复杂，但在实际操作中会发现，若只是单纯地按照算法的流程去做，时间效率是很低的，我们需要对算法进行一些必要的优化，使得它能够高效完成流程中的每一步操作。经过多年研究，这个算法目前已知的比较好的时间复杂度是 $O(k^2rd(T)+kn)$，其中 $rd(T)$是树的半径，即树中任意两点间最小距离的最大值。具体的优化方法比较复杂，这里不再赘述。