信息学奥林匹克竞赛指导丛书
吴文虎　主编

信息学奥林匹克竞赛指导

——1997—1998 竞赛试题解析

吴文虎　王建德　著

清 华 大 学 出 版 社

内 容 简 介

本书收集了1997年—1998年国际国内信息学(计算机)奥林匹克竞赛试题,重点分析解题思路和方法,其中包括如何根据题意构建数学模型与相应算法,以及如何编写程序等,可供大学、中学的电脑爱好者学习和参考。

书　名: 信息学奥林匹克竞赛指导
　　　　——1997—1998竞赛试题解析
作　者: 吴文虎　王建德　著
出版者: 清华大学出版社(北京清华大学学研大厦,邮编100084)
　　　　http://www.tup.tsinghua.edu.cn
印刷者: 北京密云胶印厂
发行者: 新华书店总店北京发行所
开　本: 787×1092 1/16　印张: 22.75　字数: 523千字
版　次: 2001年1月第1版　2001年1月第1次印刷
书　号: ISBN 7-302-04055-9/TP·2496
印　数: 0001～5000
定　价: 26.00元

前　言

国际信息学奥林匹克(International Olympiad in Informatics,IOI)从1989年到1999年,11年赛事的健康发展得益于联合国教科文组织(UNESCO)为这项赛事所做的准确定位:通过竞赛形式对有才华的青少年起到激励作用,促其能力得以发展;让青少年彼此建立联系,推动经验交流,给学校这一类课程增加活力;建立起教育工作者与专家档次上的国际联系,推进学术思想的交流。概括起来说,就是启迪思路,激励英才,发展学科,促进交流。

学科奥林匹克是智力与能力的竞赛,注重考查全面素质与创造能力。从这个意义上讲,信息学奥林匹克活动是素质教育的一个大课堂。在我国,每年国家集训队都要将"怎样做人,怎样做事,怎样求知和怎样健体"的指导思想纳入培训计划。这11年中国队共派出参赛选手43人次,累计获金牌23块、银牌11块、铜牌9块,届届名列前茅,正是因为坚持了全面素质教育的指导思想,把造就高素质有创造精神的人才作为活动的定位目标。

回顾11年的竞赛可以看出,参加高手云集的这种世界大赛是有相当难度的,第一,没有大纲,赛题范围没有界定,谁也无法去猜测每年的主办国会出什么类型的难题;第二,计算机科学与技术发展很快,层出不穷的新思路和新成果会反映到试题中来;第三,所要解决的试题往往涉及图论、组合数学、人工智能等大学开设的课程知识;第四,比较短的给定解题时间与刁难的测试数据让选手必须拿出高超和精巧的解法,无论在时间上还是空间上都是优化的解法才能取得高分。有许多赛题没有固定的现成的解法,选手要在比赛现场凭借实力,理出思路,构建数学模型,写出算法,编出程序,运行并验证整个构思是否正确,出解的时间是否能达到题目的要求,等等。可以看出,在这一过程中最重要的是要有创造能力。为激发创新精神,培养创造能力,就需要树立新的教育观念和教学方法,还要利用现代化的教学手段。引导学生学用电脑,在使用中帮助开发人脑,这可能是信息学奥林匹克活动的最重要的一个特点。我认为在这项活动中应该培养学生的四种能力:自学能力;实践动手能力;创新能力;上网获取信息,并能区分有用信息和无用信息的能力。这样做的结果使许多选手不但有能力在世界赛场上拿金牌,也有能力在学校的学习中名列前茅。

信息学奥林匹克十余年涌现出一大批出类拔萃的计算机后备人才,在他们的带动下,我国的青少年在普及计算机的大潮中阔步前进,取得了可喜的成绩。历史已雄辩地证明:计算机的普及就是要从娃娃做起,这是"科教兴国"、中华崛起的需要。为了提高普及的层次,编写竞赛辅导教材是十分必要的,也是广大青少年电脑爱好者所盼望的。这里我们将1997—1998国际国内大赛的试题集中起来进行剖析,重点讲解题思路与方法。但是必须

说明，书中的解法仅起抛砖引玉的作用。

青少年是国家的希望，不断提高青少年的科学素养是中华民族永远昂首屹立在世界东方的根基所在。“精心育桃李，切望青胜蓝”是我和王建德老师的共同心愿。

国际信息学奥林匹克中国队总教练
清华大学计算机系教授博士生导师
吴文虎
2000 年 8 月

目　　录

第 1 章　第九届国际奥林匹克信息学竞赛中国组队赛试题分析

1.1　笔　试　题

1.1.1　试题

树是计算机中最常用的数据结构之一。图 1.1.1 和图 1.1.2 是树的例子,圆圈表示结点,它们之间的连线表明了结点的连接关系,称为边。你会注意到图中的树没有回路,这对于所有的树都如此。

图　1.1.1　　　　图　1.1.2

本题中,树都是有根的,我们的算法通常是从根开始。与根相连的结点称为根的孩子,根是它们的双亲。结点 A 的双亲是你从结点 A 出发,朝着根结点的方向,寻找连接的过程中遇到的第一个结点。

可以将树划分层次,第 0 层仅包含根,第 1 层包含根的儿子,第 2 层包含根的儿子的儿子……一般而言,第 N 层包含所有第 N - 1 层结点的儿子,树的深度是指树的最大层数。图 1.1.1 和图 1.1.2 中树的深度均为 2。

1 . 二叉树 (Binary tree)

二叉树是这样一种特殊的树,它的每个结点最多有两个儿子,分别称为左儿子和右儿子。图 1.1.3 表示了一棵二叉树,儿子在图中的左右位置表示它是左儿子还是右儿子,如结点 5 的左儿子是结点 2,右儿子是结点 10;结点 12 只有左儿子 11,而没有右儿子。

如果一棵树满足以下两个条件,则称它为二叉查找树。

(1) 每一个结点的左儿子的值≤该结点的值;

(2) 每一个结点的右儿子的值≥该结点的值。

【问题 1】

画一棵包含 15 个结点且深度最大的二叉查找树。

图　1.1.3

2 . 二叉堆 **(Binary Heap)**

如果树中任一结点的所有儿子的值均大于等于它们双亲的值,则称该树是堆有序的;如果一棵二叉树除了最底层外其余层都是满的,而最底层是向左填满的,则称该二叉树为完全二叉树。

图 1 .1 .4 为一棵非堆有序的完全二叉树,图 1 .1 .5 为一棵堆有序的非完全二叉树,二叉堆是一棵堆有序的完全二叉树如图 1 .1 .6 所示。

图 1 .1 .4　　　　图 1 .1 .5　　　　图 1 .1 .6

【问题 2】

(1) 如何在二叉堆中插入一个结点以形成一个新的二叉堆 ?

(2) 如何将两个二叉堆合并成一个新的二叉堆(两个二叉堆的结点数可能不同) ?

3 . 二项树 **(Binomial tree)**

我们可以用如下的递归方式定义二项树。

二项树 B_0 为一单独的结点,二项树 B_k 可以由两个二项树 B_{k-1} 构成,其中一个的根是另一个根的最左边的儿子,如图 1 .1 .7 所示。

图 1 .1 .7

我们称 k 为二项树 B_k 的度。

【问题 3】

(1) B_n 中有多少个结点?

(2) 如何将两棵度相同且堆有序的二项树合并形成一棵新的堆有序的二项树?

4 . 二项堆(Binomial Heap)

树的集合称为森林(Forest)。一个二项堆是满足下述两个条件的二项树的森林。

(1) 每棵二项树的度互不相同;

(2) 任意一棵二项树都是堆有序的。

图 1 .1 .8 是一个 7 个结点的二项堆。

图 1 .1 .8

【问题 4】

(1) 一个包含 N 个结点的二项堆最多包含多少棵二项树?

(2) 如何在一个二项堆中寻找具有最大值的结点? 算法的最坏时间复杂度是多少(设该二项堆的结点数为 N)?

(3) 如何将两个二项堆合并成一个新的二项堆?

1 .1 .2 解答

这组笔试题的内容涉及数据结构中排序和树的有关知识,有些题是数据结构书中同类型问题的深化。但是只要读者基础扎实,严格按问题的定义推理,还是可以得出正确解的。

1 . 问题 1 的解答(有关二叉树)

将 15 个结点按关键字值递增的顺序排列,形成序列 $a_1, a_2, \cdots, a_n$。根据二叉查找树中“每一个结点的右儿子的值≥该结点的值”的性质,我们将 a_1 作为树根,a_2 作为右儿子……a_i 作为 a_{i-1} 的右儿子……直至 a_{15} 作为 a_{14} 的右儿子,见图 1 .1 .9。显然,这棵含 15 个结点的二叉查找树的深度最大(深度为 14)。

2 . 问题 2 的解答(有关二叉堆)

(1) 将一个值为 x 的结点插入二叉堆 A 中的算法是:

```
A 堆的结点个数 + 1;
在完全二叉树最后一层的最左一个空位置 i 加一片叶子;
while (i 位置非根) and (结点 i 的父亲值 > x) do
```

```
    begin
      结点 i 的值域←i 父亲的值;
      i←结点 i 的父亲位置;
    end;{while}
  x 插入结点 i 的位置;
```

图 1.1.9

显然,按照上述算法将结点插入二叉堆 A 中后,A 仍然保持二叉堆的性质。例如,在二叉堆中插入一个值为 5 的结点,见图 1.1.10。

图 1.1.10

执行上述算法的过程如图 1.1.11 所示。

(2) 由于二叉堆是一棵“任一结点的所有孩子的值均大于等于它们双亲的值”的完全二叉树(即二叉堆的根是最小的),因此将两个二叉堆 A 和 B 合并成一个新二叉堆 C 的算法应该是:

① 确定二叉堆 C 的最底层中最左的一个空位置 β。若 C 为空树,则根设为 β 位置。

② 比较二叉堆 A 的根值和 B 的根值的大小,将小者放入二叉堆 C 的 β 位置。若其中一堆为空,则将另一堆的根直接放入 C 的 β 位置。

③ 对根位置空的二叉堆进行调整:比较根的左、右儿子值,取小者填入根位置,然后再比较目前空位置的左、右儿子值,取小者填入……依次类推,直至腾出的空位置为叶结点为止。

④ 二叉堆 A 和二叉堆 B 为空,则二叉堆 C 为 A、B 两堆的合并堆;否则返回①,继续合并过程。

图 1.1.11

例如对图 1.1.12 中的两个二叉堆 A、B 进行合并。

按上述算法,合并 A、B 为 C 的过程如图 1.1.13 所示。

图 1.1.12

3.问题 3 的解答(有关二项树)

(1) B_n 中的结点数为 2^n 个结点。因为二项数 B_n 由两棵二项树 B_{n-1} 连接而成,一棵树的根成为另一棵树根的最左儿子,如图 1.1.14(a)和图 1.1.14(b)所示。所以 B_n 有 $2^{n-1}+2^{n-1}=2^n$ 个结点。

(2) 根据二项树连接方式以及堆有序的原则(树中任一结点的所有儿子的值均大于等于它们双亲的值),只要将两棵度相同且堆有序的二项树的根值进行比较,根值大的二项树作为根值小的二项树的最左的一棵子树,便可形成一棵新的二项树。例如,连接图 1.1.15(a)中的二项树 A 和二项树 B,形成一棵二项树 C,见图 1.1.15(b)。

在合并后的二项树中,由于根结点所有儿子的值大于等于它们的根,而往下的各层仍保持原两个堆的堆有序状态,因此仍是堆有序的。

4.问题 4 的解答(有关二项堆)

(1) 由问题 3 的解答(1)中可知,度为 k 的二项树 B_k 的结点数为 2^k,因此我们可以将之设为二进制数中第 k 位的权。n 的二进制数有 $[\log_2 n]+1$ 位,即所对应的二进制数为 $< b_{[\log_2 n]}, b_{[\log_2 n]-1}, \cdots, b_0 >$。由于二项堆 H 中每棵二项树的度互不相同,因此仅当 $b_k=1$ 时二项树 B_k 才出现于 H 中。这样,二项堆 H 包含的二项树的棵数为 n 对应的二进制数中值为 1 的位数和。

(2) 因为二项堆 H 中的每棵二项树 B_k 都是堆有序的,所以 B_k 中的最大关键字必在该树的叶结点中。B_k 的叶结点数为 $2^k/2=2^{k-1}$。B_0 可作为一个叶结点。我们依次比较 H 中每棵二项树的叶结点值,取其中关键字值最大的一个结点。如果二项堆 H 的结点总数为 n,则采用这种顺序比较法的最多比较次数为 $[(n+1)/2]$。因此,算法的最坏时间复

图　1.1.13

杂度为 O[(n+1)/2]。

(3) 合并二项堆必须注意下述两个问题:

① 每一次合并只能选择来自两堆且度数相同的二项树进行合并;

② 合并后的新堆中,每棵二项树的度必须互不相同。

下面我们给出将二项堆 A 并入二项堆 B 的算法:

图　1.1.14

图　1.1.15

① 按度数递增的顺序排列 A 堆中的每一棵二项树;

② 取 A 堆中度数最小的一棵二项树 T_i;

③ 如果 B 堆中有相同度数的二项树 T_i',则利用问题 3 中(2)的算法将 T_i 与 T_i'合并成 T_{i+1}(度数增 1),并按度数递增的顺序插入 A 堆的合适位置返回②,否则 T_i 并入B 堆;

④ 如果 A 堆空,则合并结束,B 堆为合并后的二项堆;否则返回②,继续合并。

由于每次合并 A 堆中的 T_i,和 B 堆中的 T_i', T_{i+1} 的度数增加 1,而 A 堆和 B 堆中的二项树个数是有限的,所以算法总会终止。

例如,对图 1.1.16 中的二项堆 A 和二项堆 B 进行合并。

图 1.1.16

按上述算法合并,其过程如图 1.1.17 所示。

图 1.1.17

1.2 工 程 规 划

1.2.1 试题

SERCOI 工程组是一个讲究效率的工程小组。为了规划和管理的方便,他们将一个

工程分为若干个项目,每个项目都可以独立进行。所有项目都工作完毕时,整个工程也就完成了。每个项目都需要一定的工作时间。工程最后总耗时是从第一个项目开始到最后一个项目结束的这段时间。

各个项目之间可能存在也可能不存在相互制约关系。如果有制约关系,则可能是以下 4 种之一(设两个项目分别为 p 和 q):

(1) SAS p q (p Start after q Start,项目 p 在项目 q 开始之后才能开始);

(2) FAS p q (p Finish after q Start,项目 p 在项目 q 开始之后才能结束);

(3) SAF p q (p Start after q Finish,项目 p 在项目 q 结束之后才能开始);

(4) FAF p q (p Finish after q Start,项目 p 在项目 q 结束之后才能结束)。

如果没有制约关系,则可同时进行。

例如,SAF 1 3 表示项目 1 必须在项目 3 完成后才能开始。若项目 3 工作时间为 3,起始时刻为 2,则项目 1 最早在时刻 5 才能开始。

作为 SERCOI 小组的项目负责人,请根据各个项目的工作时间及先后关系,找出一种安排工程的方案,使整个工程尽可能快地完成。

输入

输入文件的第一行为项目总数 n($1 \leq n \leq 100$),设项目的编号依次为 1,2,…,n。下面 n 行依次为完成每个项目所需的工作时间(每个项目占一行)。这个时间为不超过 100 的正整数。

接下来若干行是一些项目间先后次序关系的列表,每行的格式为:

〈先后次序关系符〉 (〈项目 p 编号〉 (〈项目 q 编号〉

其中:〈先后次序关系符〉为 SAS、FAS、SAF、FAF 中的任意一个,“ (”表示一个空格符。

整个文件以一个字母“ # ”表示结束(单独占一行)。

输出

若问题有解,则输出文件有 n 行,依次输出项目 1 到项目 n 的最早开始时间(设整个工程从 0 时刻开始)。每行的格式为:项目编号　最早开始时间。

若问题无解,则输出文件只有一行,为一个正整数 0。

输入输出示例

```
INPUT.TXT
3
2
3
4
SAF 2 1
FAF 3 2
#

OUTPUT.TXT
1 0
```

2 2
3 1

INPUT .TXT
3
1
1
1
SAF 2 1
SAF 3 2
SAF 1 3
#

OUTPUT .TXT
0

1 .2 2 算法分析

1 . 项目间的制约关系图

一个项目间先后次序关系的列表可以对应一张有向图,其中顶点对应项目,有向弧对应活动,弧权对应活动的持续时间。与有向弧相连的每一个顶点表示在它之前的活动已经完成,在它之后的活动可以开始。当项目 p 和项目 q 是如下 4 种制约关系时, q 和 p 相连的有向弧及其权可表示为:

设 t_p , t_q 分别为项目 p 和项目 q 的工作时间:

SAS p q $q \xrightarrow{0} p$

表示项目 p 在项目 q 开始之后开始,即项目 p 的开始时间 = 项目 q 的开始时间 + 0;

FAS p q $q \xrightarrow{-t_p} p$

表示项目 p 在项目 q 开始之后结束,即项目 p 的开始时间 = 项目 q 的开始时间 - t_p;

SAF p q $q \xrightarrow{t_q} p$

表示项目 p 在项目 q 结束之后开始,即项目 p 的开始时间 = 项目 q 的开始时间 + t_q;

FAF p q $q \xrightarrow{t_q - t_p} p$

表示项目 p 在项目 q 结束之后结束,即项目 p 的开始时间 = 项目 q 的开始时间 + t_q - t_p。

相当多的选手采用了求关键路径的算法。他们增设一个入度为 0 的源点 s, s 向每一个入度为 0 的顶点引出一条权为 0 的有向弧;增设一个出度为 0 的汇点 t, 每一个出度为 0 的顶点向 t 引出一条权为 0 的有向弧, 使得项目的制约关系图从表面上看与《数据结构》书上的 AOE 网(带权有向无环图)很相似, 以至于许多选手套用了求关键路径的标准算法, 他们力图通过求 s 至 t 的关键路径得到问题的解。在某些情况下(所有项目间的制约关系形成一条回路), 由于源点 s 和汇点 t 成为孤立顶点, 使得这些选手的企图落空。例

如项目 1 和项目 2 的先后次序为(参见表 1.2.1):

FAF 1 2　　FAF 2 1

对应的有向图见图 1.2.1。

表 1.2.1

项目	工作时间
1	1
2	1

图 1.2.1

显然,图 1.2.1 中不能求出 s 至 t 的关键路径,但这是否意味问题无解呢?不一定。实际上这是试题设计者巧设的一个误区,以此告诫选手,寻求正确算法应从试题给出的条件和目标出发,而不是生搬硬套“本本”上的算法。

2.每个项目最早开始时间的计算方法

正确的算法应该是,在项目间制约关系图的基础上,增设一个入度为 0 的源点 s,表示工程开始。s 向每一个顶点引出一条权为 0 的有向弧。为了便于判别图中可能出现的权和非零的有向环,除 s 外的每一个顶点连一条权为 0 的自反弧。例如上述例子对应的新图 G′见图 1.2.2。

图 1.2.2

由于在新图 G′中的每个项目可以独立并行的进行,所以项目 $i[1 \le i \le n$(项目数)]的最早开始时间是从源点 s 到顶点 i 的最长路径的长度,即最长路径上各活动持续时间之和。显然在图 1.2.2 中,项目 1 的最早开始时间 = 0,项目 2 的最早开始时间 = 0。

如果在图中出现权和大于零的有向环,意味着有向环上的项目应以自己为先决条件,其最早开始时间的计算沿这个有向环死循环下去,势必得出一个无穷大的值,显然这是荒谬的。因此,一旦在计算过程中发现路径长度大于零的有向环,则算法应以无解而告终。

例如,$t_1 = 1, t_2 = 1, t_3 = 1$,SAF 2 1, SAF 3 2, SAF 1 3,见图 1.2.3。

由于存在一条项目 1→项目 2→项目 3→项目 1 的长度为正的有向环,所以问题无解。

我们采用动态规划算法求源点 s 至顶点 i 的最长路径长度(floyd 算法)。设:

项目数为 n,源点 s 的序号设为 n + 1。

阶段 i——路径的中间顶点,显然可以分成 n + 1 个阶段(i = 1, 2, …, n + 1);

图 1.2.3

状态 j——当前路径的出发点($1 \le j \le n+1$);

决策 k——当前路径的终点($1 \le k \le n+1$);

map[i,j]——传递闭包。

即:

$$map[i,j]=\begin{cases}1, & i\text{至}j\text{有路};\\ 0, & i\text{至}j\text{无路。}\end{cases}$$

$$(1 \le i,j \le n+1)$$

显然初始时,map[n+1,i]和 map[i,i]为 1,若项目 i 和项目 j 存在如下 4 种制约关系之一时:

SAS i j、FAS i j、SAF i j、FAF i j

则 map[j,i]=1。若在计算过程中得出顶点 j 至顶点 k 的最长路径长度(dist[j,k]),则 map[j,k]=1。

dist[i,j]——顶点 i 至顶点 j 的最长路径长度。显然初始时 dist[n+1,i]=0,dist[i,i]=0,dist[i,j]为图中(i,j)弧上的权。动态规划后得出的 dist[n+1,1…n]即为项目 1 至项目 n 的最早开始时间。

动态规划方程如下:

当(map[j,i]=1) and (map[i,k]=1) and ((dist[j,i]+dist[i,k]>dist[j,k])or (map[j,k]=0))时:

dist[j,k]= dist[j,i] +dist[i,k],map[j,k]=1

若 j=k,则说明图中出现长度非零的有向环,程序应无解退出。

$$(1 \le i \le n+1,\ 1 \le j \le n+1,\ 1 \le k \le n+1)$$

1.2.3 程序题解

```
{$A+,B-,D+,E+,F-,G-,I-,L+,N-,O-,P-,Q-,R-,S-,T-,V+,X+,Y+}
{$M 65520,0,655360}
Program Project (input,output);
Const  MaxN            = 101;          {项目最大数}
       Inputfilename   = 'Input.txt';  {输入文件名串}
       OutPutfilename  = 'Output.txt'; {输出文件名串}
```