青少年计算机竞赛指导丛书

青少年国际和全国信息学（计算机）奥林匹克竞赛指导

—— 组合数学的算法与程序设计

吴文虎 王建德 编著

清华大学出版社

青少年计算机竞赛指导丛书

青少年国际和全国信息学（计算机）奥林匹克竞赛指导

——组合数学的算法与程序设计

吴文虎　王建德　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

用计算机编程解题的核心问题是算法，而组合数学是算法的主要内容。组合数学对于参加信息学奥林匹克活动的青少年而言，是一门提高思维能力、分析与判断能力，以及自我构造算法的重要课程。本书力求将分析问题与自己上机编程结合起来，这样做可以化难为易。书上不但讲了组合数学的原理、概念和分析问题的思路，还讲了如何编程，并给出了参考程序，这对自学本书极为有利。这本书不仅参赛选手可读，对于一些理工科的大学生也可用作学习编程解题的参考资料。

图书在版编目(CIP)数据

青少年国际和全国信息学(计算机)奥林匹克竞赛指导——组合数学的算法与程序设计/吴文虎，王建德编著. —北京：清华大学出版社，1996
(青少年计算机竞赛指导丛书)
ISBN 7-302-02203-8

Ⅰ. 青… Ⅱ. ①吴… ②王… Ⅲ. 组合数学-算法-程序设计 Ⅳ. O157

中国版本图书馆 CIP 数据核字(96)第 00168 号

出版者：清华大学出版社(北京清华大学校内，邮编 100084)
印刷者：人民文学印刷厂
发行者：新华书店总店北京科技发行所
开 本：787×1092 1/16 印张：12.75 字数：298 千字
版 次：1997 年 3 月第 1 版 1997 年 3 月第 1 次印刷
书 号：ISBN 7-302-02203-8/TP・1060
印 数：0001—5000
定 价：13.80 元

前　言

信息科学与技术正在对人类社会的发展产生难以估量的深远影响，并将成为新世纪的一个标志。作为人类总体智慧的结晶，电脑已经成为一种新的现代文化。“计算机的普及要从娃娃做起”已经成为“科教兴国”的一项重要内容。

一个国家、一个民族要想不落伍，要想跻身于世界先进民族之林，关键在于拥有高素质的人才；综合国力的竞争，说到底也是人才的竞争。电子计算机是现代科学与技术的基础和核心，它的飞速发展，把社会生产力水平提到前所未有的高度，人类进入了信息时代。电脑对人类社会的发展所起的巨大作用，特别是对人类智能的发展所起的促进作用，已为人们普遍认识到。计算跟语言一样是人类社会每时每刻都不可缺少的。现在，人类已经拥有了帮助自己进行复杂计算与思维的工具，电子计算机起到了人脑延伸的作用。以往历史上的技术革命，只能起到创造和改进工具，用机器代替人的体力的作用；而计算机则是把人从重复性的或有固定程式的脑力劳动中解放出来，使自己的智能获得空前的发展。作为“人类通用智力工具”计算机在开发人类智能方面所起的无与伦比的作用不容忽视。这也就是计算机与基础教育相结合，能够成为当今世界的大趋势的一个原因。从信息社会要求人才具备的科学素养看，数学、物理学、化学、生命科学和信息科学是五大支柱，这正是联合国教科文组织倡导举行五项国际学科奥林匹克竞赛的内容。

国际信息学奥林匹克(International Olympiad in Informatics，简称 IOI)始于 1989 年，到 1996 年已成功地举办了 8 届。这是一种智力与应用计算机能力的大赛。从益智的角度看，是用电脑帮助开发人脑，重在提高思维能力，培养创新意识。在中国队的训练中强调德智体美全面发展；心态上自立、自尊、自信、自强，要怀着中华民族的自豪感和自信心去参赛；这种心态是学习、训练和取胜的重要条件。8 届比赛，中国队每届都取得了名列前茅的好成绩，31 人次参赛，夺得 31 块奖牌，其中金牌 17 块、银牌 6 块、铜牌 8 块。特别是 IOI'95(荷兰)突破了前 6 届比赛女孩与金奖无缘的纪录，两名中国女选手荣登金牌领奖台。在 IOI'96(匈牙利)上中国队又实现了全金的突破，四名选手，每人获得一块金牌。

从大局看，竞赛不是目的，是推动普及的手段，我们的目的只有一个：“科教兴国”。竞赛活动带有因材施教、因材施测的特点。普及是有层次的，与学科竞赛有关的普及活动，对青少年而言属于比较高的层次，当然就有相当的难度。我们编写的这套青少年计算机竞赛指导丛书，涉及程序设计语言、常用算法、组合数学、图论、人工智能搜索等的基本知识和基本方法。这些理论知识往往都是通过竞赛当中的一些实例来讲解的。目前已拟就了三本书：其一是《组合数学的算法与程序设计》，其二是《图论算法与程序设计》；其三是《国际国内青少年信息学(计算机)竞赛试题解析(1994～1995)》。前两本书试图从数学的基本概念和基本理论出发，给出算法设计和编写程序的方法，第三本书收集了最近国际国内比赛的典型试题及其解法，重点放在解题思路上。这里许多题目比较新颖，很难去套固定算法或固定模式，这中间有些招数是选手们想出来的。从中可以看出信息学奥林匹克要求创

新，鼓励创新。当然，书中给出的解法，对青少年读者而言，我们希望仅仅起到抛砖引玉的作用，并且热切盼望引出更多的玉来。作为老师，我和王建德都这样想，“精心育桃李，热望青胜兰”就是我们编写这套丛书的初衷。

中国计算机学会普及委员会主任
国际信息学奥林匹克中国队总教练
清华大学计算机科学与技术系教授
吴文虎

1996 年 11 月 20 日

目　录

第一章 导 论

作为全书的导论，本章将对组合学的内容和组合题的求解方法作一点介绍，以引导读者入门。

1.1 组合数学的研究对象

组合数学具有悠久的历史。但是，它的发展壮大还是近几十年的事情，特别是计算机的问世以及计算机的广泛应用，促使了组合数学的蓬勃发展；反过来，由于组合数学的发展，又推动了计算机科学日新月异的进步。

同样，国际和国内青少年奥林匹克信息学竞赛与组合数学的关系，也是甚为密切的。因为程序设计的核心是算法研究，而组合算法是算法的主要内容。没有组合数学的基础，就无法深入研究算法和分析算法。竞赛试题的形式和类型千变万化，但通常蕴涵某个组合数学方面的问题，这些问题很能推动人们去思索，它们的解法也常常是机智和精巧的。因此，对于参与奥林匹克信息学竞赛活动的青少年来说，组合数学是一门提高思维分析能力和自我构造算法本领的必修课程。

自然，读者会问，什么是组合数学？很难（也没有必要）在这里下一个确切的定义。我们只想从“组合数学的主要研究对象是什么”这个侧面来回答问题。而真正了解这个问题是在读完本书的终了，而不是现在。

组合数学研究的主要内容是依据一定的规则来安排某些事物的有关数学问题。这些问题包括四个方面：

1. 这种安排是否存在，即存在性问题；
2. 如果符合要求的安排是存在的，那么这样的安排又有多少，即计数问题；
3. 怎样构造这种安排，即算法构造问题；
4. 如果给出了最优化标准，又怎样得到最优安排，即最优化问题。

一、存在性问题

实际生活中的各种问题，有些可以当即判定其有解或无解，但也有不少一时难以判定。例如宴会上，奇数位客人能否在晚会上与他人握手奇数次。这不是不加思索就可判定的问题。

当然一般来说，竞赛不可能出现专门判定某问题有解或无解的试题。但往往会出现这样的情况：专家在为试题设计的一组测试数据中，故意掺杂几个“无解”的数据，引诱盲目求解的选手进入误区。因此选手在具体问题求解时，为避免盲目性，最好先解决存在性问题，即明确对哪一类数据是无解的，在判定出解的同时构造求解方法。

二、计数问题

如一个组合问题的解已知是存在的，自然会问有多少不同的解。

例如：将 8 个“车”放在 8×8 的国际象棋盘上，如果它们两两不能互吃，那么称 8 个“车”处于一个安全状态。显然，这种安全状态是存在的。问有多少种不同的安全状态。这个试题就是一个计数问题。

信息学竞赛的试题一般分为两种类型：一种是计数类型，即计算具有某种特性的对象有多少，但计算过程一般比较复杂灵活，非手算所能解决。另一种类型是枚举，即把所有具有某种特性的对象完全列举出来。譬如显示上题中每个处于安全状态的棋盘格局。但即便是枚举类型的试题，我们也得学会使用组合数学中的计数理论分析问题，这是因为：

1. 可以通过计数公式设计准确的测试数据，以验证枚举结果正确与否；

2. 通过计数公式引来的组合分析，可以启迪我们巧妙构思算法，以避免盲目性，提高枚举效率。

存在性问题和计数问题构成了本书的基础——组合理论。本书第二至第七章系统地介绍了这一理论的基本知识，包括鸽笼原理、排列组合、容斥原理、母函数、递归关系、Pólya 定理等必须掌握的内容。

三、构造性算法

一个组合问题，已判知解存在，甚至也推知有几组解，但关键还在于把解构造出来，有的哪怕出一组解也好。如魔方问题、正交拉丁问题等。本书在第八章围绕这两个组合问题，初步阐述了如何从组合论的角度设计算法，使之较好地构造出解。

四、优化问题

一个问题的构造性算法，可能不止一种，自然面临如何择优，如何改进，使得答案尽快地解出来。本书在第九章、第十章侧重论述了线性规划和动态规划的基本原理、方法及其应用。

上述四个方面，只是提供问题的研究方式，求解时不见得都刻板地分这四步。究竟从哪个方面着手，使用哪种方法、怎样用，需要“具体问题具体分析”。实际上，解组合问题需要的是机智、灵活，不宜死记成规、生搬硬套，得靠自己想方设法，所谓“阵而后战，兵法之常，运用之妙，存乎一心”是也。

1.2 组合问题的基本解题方法

组合问题的求解方法层出不穷，千变万化，很难给出一个纲领式的概括。本书将通过大量例题的求解，向读者展示组合学的基本解题方法。这些方法大致可以分为两类：

一、从组合学基本概念、基本原理出发的所谓常规方法

这类方法常用于解标准题。例如利用容斥原理、Pólya 原理解计数问题；解存在性问

题的鸽笼原理、递归方法、生成函数方法等。这类方法通常比较刻板，读者将在以后各章里分别读到并学会它们。

二、通常与问题所涉及的组合数学概念无关的非常规方法

这类方法常用于解那些需要独立思考、见解独到和有所创新的非标准题。

下面介绍几种典型的非常规方法。

1. 数学归纳法

对问题的存在性进行分析时，通常使用数学归纳法中的归纳原理。

[例 1] 证明 n 个元素的集合，其子集恰为 2^n 个

证：对 n 归纳。

设 $s=\{a_1,a_2,\cdots,a_n\}$

当 $n=0$ 时，$s=\{\ \}=\varphi$，它只有一个子集(自身)，而 $1=2^0$，故命题真。

设 $n=k$ 时命题成立。现证 $n=k+1$ 时命题也成立。

先从 s 中取出一个元素 a，于是据归纳假设，s 有 2^k 个不含 a 的子集，该集合的全体子集分两类：

(1) 不含 a 的 2^k 个子集；

(2) 含 a 的 2^k 个子集。

因为含 a 的子集与不含 a 的子集是一一对应的(含 a 的子集去掉 a 便是不含 a 的子集之一；反之，不含 a 的子集加进 a 便是含 a 的子集之一)，因此该集合的子集总数为 $2^k+2^k=2^{k+1}$。

即命题在 $n=k+1$ 时亦真。根据归纳原理，对一切 n 原命题成立。

2. 一一对应技术

将问题模式转化为另一种有常规算法的问题模式。

[例 2] 将 8 个“车”放在 8×8 的国际象棋棋盘上，如果它们两两均不能互吃，那么称 8 个“车”处于一个安全状态。问共有多少种不同的安全状态？

解：8 个车处于安全状态当且仅当它们处于不同的 8 行和 8 列上。

用一个排列 $a_1,a_2,\cdots,a_8$，对应于一个安全状态，使 a_i 表示第 i 行的 a_i 列上放置一个“车”。这种对应显然是一对一的。因此，安全状态的总数等于这 8 个数的全排列总数 $8!=40320$。

有了上述一一对应，枚举全部安全状态，则变成轻而易举的事：先求出 $N!$ 种全排列方案，对每个排列方案设定元素 a_i 为 i 行上车的列位置($1\leqslant i\leqslant8$)，使之对应一个安全状态。若不使用一一对应技术，盲目地逐个枚举方案，其笨拙是可以想象的。

由上可见，一一对应技术是一种重要的解题方法。在解题时，心中常要回忆已经解决的类似的问题和有关事实，这样往往会收到意想不到的好效果。

3. 殊途同归方法

从不同角度讨论计数问题，以建立组合等式。

[例 3] 对没有三条对角线交于一点的凸多边形，计算各边及对角线所组成的互不重叠的区域个数。见图 1-1。

我们从各区域的顶点总数和所有区域的内角和的总和两个角度进行分析：

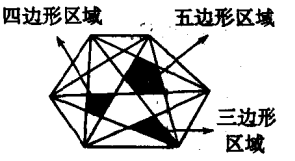

图 1-1

设：N_k——区域中 k 边形的个数。

角度 1：各区域顶点总数（包括重复计算的数目）的等式为

$$3N_3+4N_4+\cdots+mN_m=4C(n,4)+n(n-2) \quad (1\text{-}1)$$

其中 m 是各区域边数的最大值，n 是凸多边形的顶点数。

左式表示的各区域顶点总数来自两个方面：

由于每两条对角线（或四个顶点）决定一个内部区域的顶点，因此区域的顶点数是 $4C(n,4)$，即每个内部顶点在左式中计数 4 次（总是四个区域公共一个顶点）。又因为在计算中，凸多边形的每个顶点（为 $n-2$ 个三个角形的公共顶点）重复计数 $n-2$ 次，因此左式的计算中，n 个顶点被计数为 $n(n-2)$，由此得出等式(1-1)。

角度 2：所有区域的内角和的总和的等式为

$$\begin{aligned}&180°N_3+360°N_4+540°N_5+\cdots+(m-2)180°N_m\\&=C(n,4)360°+(n-2)180°\end{aligned} \quad (1\text{-}2)$$

左式表示的所有区域的内角和的总和来自两个方面：

(1) 各内部顶点处区域内角和(360°)的总和为 $C(n,4)360°$；

(2) 凸多边形的内角和为 $(n-2)180°$。

由此得出等式(1-2)。

等式 2 两边同除以 180°，得出

$$N_3+2N_4+\cdots+(m-2)N_m=2C(n,4)+(n-2) \quad (1\text{-}3)$$

殊途同归。由等式(1-1)两边同时减去等式(1-3)两边，可得出区域总数：

$$N_3+N_4+N_5+\cdots+N_m=C(n,4)+C(n-1,2)$$

这就是说，所求的区域总数为 $C(n,4)+C(n-1,2)$

4. 数论方法

应用奇偶性、整除性等数论性质进行存在性问题的分析推理。

[例 4] 设 n 位客人，在晚会上每人与他人握手 d 次，d 是奇数。证明 n 是偶数。

证：由于每一次握手均使握手的两人各增加一次与他人握手的次数，因此 n 位客人与他人握手次数的总和 nd 是偶数——握手次数的 2 倍。根据奇偶性质，已知 d 是奇数，那么 n 必定是偶数。

[例 5] L 形骨牌形状如图 1-2(a)。

L 形骨牌的完全覆盖，是指互不重叠的 L 形骨牌对图形的无间隙的覆盖，并且没有任何一块骨牌伸出图形之外。

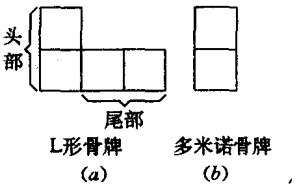

图 1-2

(1) 证明 5×4、6×6 的矩形不可用 L 形骨牌完全覆盖；

(2) 证明，如果一个 $m \times n$ 矩形是可用 L 形骨牌完全覆盖的话，那么所用的骨牌必定是偶数；

(3) 证明，若 m、n 均大于 3，并且 8 整除 mn，那么 $m \times n$ 矩形可用 L 形骨牌完全覆盖。

证(1)

如果 5×4、6×6 矩形可用 L 形骨牌完全覆盖，那么 L 形骨牌数为 5(5 * 4/4)块和 9(9 * 4/4)块。与结论(2)冲突。因此问题又转入对(2)的证明，只要(2)成立，则(1)也成立。

(2) 一块 L 形骨牌恰由两块多米诺骨牌〔见图 1-2(b)〕组成，分别称为 L 形骨牌的头部和尾部。一个 L 形骨牌的完全覆盖，可看作是两个多米诺骨牌的完全覆盖。因此，$m \times n$ 矩阵含偶数个方格，即 m，n 中至少一个是偶数。设 $m \times n$ 矩形可被 L 形骨牌完全覆盖。比如图 1-3 中的 5×8 矩阵。用一个水平直线 $L_i(1 \leqslant i \leqslant n-1)$来切割这个图形。$L_i$ 与 L_{i+1}的间距为一个方格长度。

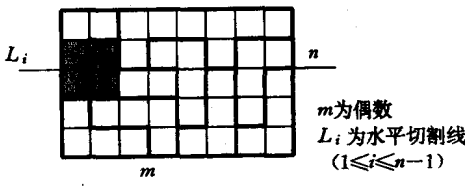

图 1-3

我们可以证得 L_i 必定切割到偶数个 L 形骨牌的头部或尾部。因为若 L_i 切割到奇数个 L 形骨牌的头部和尾部，那么，L_i 以上部分的偶数个方格在去除被切割到的奇数个方格后，剩下的奇数个方格要实现多米诺骨牌的完全覆盖是不可能的。不难明白，每一 L 形骨牌都有一个并且仅有一个头部或尾部被某一 L_i 所切割，而诸 L_i 中不可能有两条同时切割一个 L 形骨牌。因此，全部 L_i 所切割的 L 形骨牌的头部或尾部共计偶数个，从而覆盖矩形的 L 形骨牌也是偶数个。

(3) 注意两个事实

1. 2×4 矩形是 L 形骨牌可完全覆盖的；

2. 5×8 矩形也是 L 形骨牌可完全覆盖的。

现设 m 可被 4 整除，n 可被 2 整除(或 n 可被 4 整除，m 可被 2 整除)那么，矩形可分为若干个 2×4 矩形，从而明显可被 L 形骨牌所覆盖；又若 m(或 n)可被 8 整除，n(或 m)为奇数，那么矩形可分为若干个 2×8 矩形和一个 5×8 矩形，所有这些矩形都是可用 L 形骨牌完全覆盖的。综上所述，当 m，n 均大于 3，且可用 8 整除 m，n 时，$m \times n$ 矩形可用 L 形骨牌完全覆盖。

由(2)、(3)可得出 L 形骨牌完全覆盖的一个充分必要条件：$m \times n$ 矩形可用 L 形骨牌完全覆盖，当且仅当 m，n 可被 8 整除。

如果矩形不能被 L 形骨牌完全覆盖的话，我们可根据上述推论结果，分析得出由 m，n 值确定的最少空格数 STEP：

若 $m*n$ 能被 4 整除而不能被 8 整除的话，则 STEP=4；

否则　STEP=($m*n$)除以 4 的余数。

这个结论的证明，留给读者分析。我们将在 1.3 节的例 2 中，引用这个由奇偶性、整除性得出的结论来解题，可使程序效率大为提高。

上述四种基本解题方法可以帮助读者拓宽组合分析的思路，但它还不能完全解决如何编程解题的问题。组合分析＋回溯法是求解特殊类型的计数题或复杂的枚举题过程中最常用的方法，本书的程序例题很多采用了这种方法。

为了尽可能使读者请楚地了解什么是回溯法、如何应用回溯法解组合题，我们将在下一章节中对回溯法作一个比较详尽的专题讨论。

1.3　回溯法的讨论

需要用计算机求解的组合试题一般有两种类型：

1. 能够用简明正确的组合公式揭示问题。

对于这一类试题，我们尽量用解析法求解。因为一个好的数学模型建立了客观事物间准确的运算关系，运用这个数学模型求解是再合适不过了。组合数学对常见的计数问题都建立了数学模型。

2. 不能对给定问题建立数学模型，或即便有数学模型但解该模型的准确方法也不一定能运用现成算法。在求解枚举类型试题时，常会遇到这类问题。

对于第二类问题，我们一般采用搜索的方法解决，即从初始状态出发，运用题目给出的条件、规则扩展所有可能情况，从中找出满足题意要求的解答。回溯法是搜索算法中的一种控制策略，亦是求解特殊类型计数题或较复杂的枚举题中使用频率最高的一种算法。

何谓回溯法，我们不妨通过一个具体实例来引出回溯思想与在计算机上实现的基本方法。

[**例 1**]　一个 $n\times n$ 的国际象棋棋盘上放置 n 个皇后，使其不能相互攻击，即任何两个皇后都不能处在棋盘的同一行、同一列、同一条斜线上，试问共有多少种摆法？

一、如何求 n 皇后问题

在分析算法思路之前，先让我们介绍几个常用的概念：

1. 状态(state)

状态是指问题求解过程中每一步的状况。n 皇后问题中，某行皇后所在的列位置 $i(1\leqslant i\leqslant n)$即为该皇后问题的状态。显然，对问题状态的描述，应与待解决问题的自然特性相似，而且应尽量做到占用空间少，又易于用算符对状态进行运算。

2. 算符(operater)

算符是把问题从一种状态变换到另一种状态的方法代号。算符通常采用合适的数据来表示。由 1.2 节中的例 2 可推知，n 皇后的一种摆法对应 n 个元素的排列方案(a_1，a_2，

$\cdots,a_n$)。排列中的每个元素 a_i 对应 i 行上皇后的列位置($1\leqslant i\leqslant n$)。由此想到，在 n 皇后问题中，采用当前行的列位置 i($1\leqslant i\leqslant n$)作为算符是再合适不过了。由于每行仅放一个皇后，因此行攻击的问题自然不存在了，但在试放当前行的一个皇后时，不是所有列位置都适用。例如(L,i)位置放一个皇后，若与第 j($1\leqslant j\leqslant L-1$)行皇后产生对角线攻击(abs($L-j$)=abs($j$行皇后的列位置$-i$))或者列攻击($i=j$ 行皇后的列位置)，那么算符 i 显然是不适用的，应当舍去。因此，不产生对角线攻击和列攻击是 n 皇后问题的约束条件，即排列(排列 $a_1,a_2,\cdots,a_i,\cdots,a_j,\cdots,a_n$)必须满足条件($|j-i|\neq|a_j-a_i|$) and ($i\neq a_j$) ($1\leqslant i,j\leqslant n$)。

3. 结点(node)

用以表明某状态特征及关联方式的基本信息单元。结点的数据结构一般为记录类型

```
type
  node=record                    {结点类型}
  operater:算符类型;
  state:状态类型;
  end;
var
  stack:array [1..maxdepth] of node;{结点数不超过 maxdepth 的一条路径}
```

由于 stack 数组下标(n 皇后问题中的行序号)已指明各结点间的逻辑关系，所以毋需另辟指针域了。在 n 皇后问题中，状态和算符同指当前行的元素值，结点类型最终简化为一个短整型。描述当前方案的路径可以直接定义为：

```
var
  stack:array [1..20] of integer;
```

现在让我们先来观察一个简单的 n 皇后问题。设 $n=4$，初始状态显然是一个空棋盘，见图 1-4。

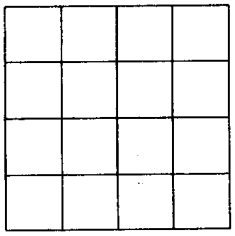

图 1-4

此时第一个皇后开始从第一行第一列位置试放，试放的顺序是从左至右、自上而下。为了说明这个事实，我们引进了 4 个结点，每个结点表征相应的状态信息(见图 1-5)：

(× × × ×)

图 1-5

每个结点共有 4 个数据。第 i($1\leqslant i\leqslant 4$)个数据指明当前方案中第 i 个皇后置放在第 i

行的列位置。若该数据为0，表明所在行尚未放置皇后。从初始的空棋盘出发，第1个皇后可以分别试放第1行的4个列位置，扩展出4个子结点，见图1-6。

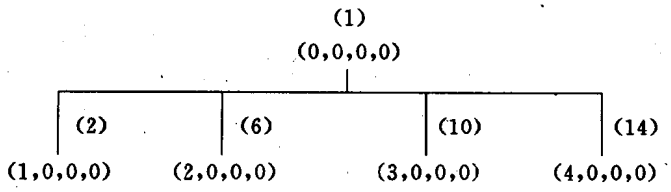

图 1-6

图1-6中，在结点右上方给出按回溯法扩展顺序定义的结点序号。现在我们也可以用相同方法找出这些结点的第二行的可能列位置，如此反复进行，一旦出现新结点的四个数据全非空，那就寻到了一种满足题意要求的摆法。当尝试了所有可能方案，即获得了问题的解答，于是得到了图1-7的图形。

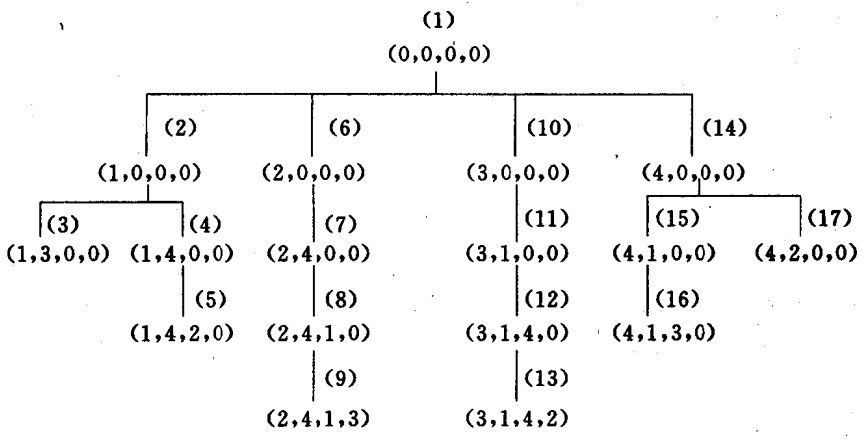

图 1-7

该图形像一棵倒悬的树。其初始结点 V_1 叫根结点，而最下端的结点 V_3、V_5、V_9、V_{13}、V_{16}、V_{17}称为叶结点，其中4个数据全非零的叶结点，亦即本题的目标结点。由根结点到每一个目标结点之间，揭示了一种成功摆法的形成过程。显然，4皇后问题存在由 V_9、V_{13}表示的二种方案。图1-7被称作解答树。树中的每一结点都是当前方案中满足约束条件的元素状态。除了根结点、叶结点以外的结点都称作分枝结点。分枝结点愈接近根结点者，辈分愈高；反之，愈远离根结点者，辈分愈低。图1-7中结点 V_7是结点 V_8 的父结点(又称前件)，结点 V_{15}是结点 V_{14}的子结点(又称后件)。某结点所拥有的子结点的个数称作该结点的次数。显而易见，所有叶结点的次数为0。树中各结点次数最大值，被称作为该树的次数。算符的个数即为解答树的次数。由图1-7可见，4皇后的解答树是4次树。

一棵树中的某个分枝结点也可视作为“子根”，以此结点为根的树则称作“子树”。

由以上讨论可以看出解答树的结构：

1. 初始状态构成(主)树的根结点；

2. 除根结点以外，每个结点都具有一个、且只有一个父结点。对应于 n 皇后问题来说，置放 i 行皇后的子结点，只有在置放了前 $i-1$ 行皇后的一个父结点基础上产生；

3. 每个非根结点都有一条路径通往根结点，其路径长度(代价)定义为这条路径的边数。对应于 n 皇后来说，当前行序号即为路径代价。当路径代价为 $n+1$ 时，说明 n 个皇后已置放完毕，一种成功的摆法产生。

有了以上的基础知识和对 n 皇后问题的初步分析，我们已经清楚地看到，求解 n 皇后问题，无非就是做两件事：

1. 从左至右逐条树枝地构造和检查解答树 t；

2. 检查 t 的结点是否对应问题的目标状态。

上述两件事同时进行。为了加快检查速度，一般规定：

1. 在扩展一个分枝结点前进行检查，只要它不满足约束条件，则不再构造以它为根的子树；

2. 已处理过的结点若以后不会再用，则不必保留。即回溯过程中经过的结点不再保留。例如图 1-7 当我们求出第一种摆法 V_1-V_2-V_3 后，由于皇后置放第三行任何列位置都会产生攻击，因此舍弃该摆法，开始寻求第二种摆法。从图 1-7 可看出，第二条路径为 V_1-V_2-V_4-V_5，V_3 在第二种摆法中不再用到，不必保留，应当退回到 V_2 状态，从那里选择尚未使用过的列位置 4，扩展出 V_4。一般来说，当求出一条路径后，必须从叶结点开始，沿所在路径回溯，回溯至第一个还剩有适用算符的分枝点(亦称为尚未使用过的通向右边方向的结点)，从那里换上一个新算符，继续求下一条路径。

按上述规定，对照图 1-7，我们来具体分析 4 个皇后的置放过程。初始状态(0,0,0,0)作为根结点 V_1，由此出发，置第 1 个皇后于第 1 行第 1 列位置。从(1,0,0,0)开始，第 2 个皇后相继选择了第 2 行的 1,2 列位置，由于会产生攻击，因此选择该行的列位置 3 放入，产生状态(1,3,0,0)。但是第 3 个皇后无论放入第 3 行哪列位置都难逃攻击，因此只得沿第一条路径回溯至第一个尚未用过的通向右边方向的分枝点 V_2，以寻求第二种摆法。从(1,0,0,0)状态换上新的列位置 4，产生(1,4,0,0)。从(1,4,0,0)选择列位置 2(由于列位置 1 产生攻击)，产生(1,4,2,0)。由于第 4 个皇后无论置放第 4 行哪列位置都会产生攻击，第二种摆法失败，同样再从 V_5 开始，沿第二条路径回溯。由于 V_2，V_4 都没有未使用的满足约束条件的算符(列位置)了，因此第一个分枝点是 V_1，从 V_1 的(0,0,0,0)换上位置 2，产生 V_6 的(2,0, 0,0)。这样依次使用满足约束条件的算符扩展下去，又得出第三条路径 V_1-V_6-V_7-V_8-V_9。可见，V_9 的(2,4,1,3)是一种成功的摆法。

按上述规律不断回溯检查，直至得出第六条路径 V_1-V_{14}-V_{17}。沿路径从 V_{17} 回溯，由于 V_{14} 选择尚未用过的列位置 3,4 都会产生攻击，因此不再剩有适用的列位置了，只得回溯至 V_1。又因为 V_1 已经选择了列位置 4 而无法再扩展，至此，求出了 4 皇后的所有可能摆法。

二、回溯法的算法分析和程序框架

如上节所述，从左至右逐条树枝地构造和检查查找解答树，已处理过的结点若以后不会再使用则不必保留(一般说来，检查长度为 n 的树枝，只要保留 n 个结点就够了)。若按这种方式得到一条到达树叶的树枝 t，实际上就得到了一条路径。然后沿树枝 t 回溯到第一个尚未使用过通往右边路径方向上的分枝点，并由此分枝点向右走一步，然后再从左至

右地逐个进行构造和检查，直至达到叶子为止，这时又得到一条路径。按这种方法搜索下去，直至求出所有路径。显然用这种方法检查，在树枝左边的一切结点都已检查过，树枝右边的一切结点尚未产生出来。我们把这种不断“回溯”查找解答树中目标结点的方法，称作“回溯法”。

由上述算法思想，我们很容易想到，应选择怎样一种数据结构来存放当前路径上各结点的状态和算符？它应具有“后进先出”的特征，就像食堂里的一叠盘子，每次只许一个一个地往顶上堆，一个一个地从顶上往下取。这就是我们通常所说的栈。

栈是一种线性表，所有进栈或出栈的数据都只能在表的同一端进行，就像堆盘子和拿盘子一样，都只能在顶端“堆上”或“取下”。这顶端叫“栈顶”，另一端叫“栈底”。Pascal 编译系统内部，保留一部分内存用作栈区，存放过程和函数的值参以及过程和函数内部所说明的局部变量。每当一个过程和函数被启用时，系统就在栈顶分配一组值参和局部变量(进栈)。而当该过程或函数退出时，这些局部变量或值参就被消除(退栈)。

我们为回溯法设计的一个递归过程 make(L)，就是利用系统的这一特性(见图 1-8)。

1. 在全局变量说明中，分配一个连续的数组区域 stack[1..maxdepth]，顺序存放栈中表目，即当前路径的结点。栈元素为结点，一般用路径最大长度 maxdepth 作为栈容量，直接用当前数组下标 L 指向栈顶，但 L 不允许是全局变量。

2. 栈顶指针 L 作递归程序 make 的值参数，指出待扩展结点在当前路径序列 stack 中的顺序，算符作 make 过程的局部变量。

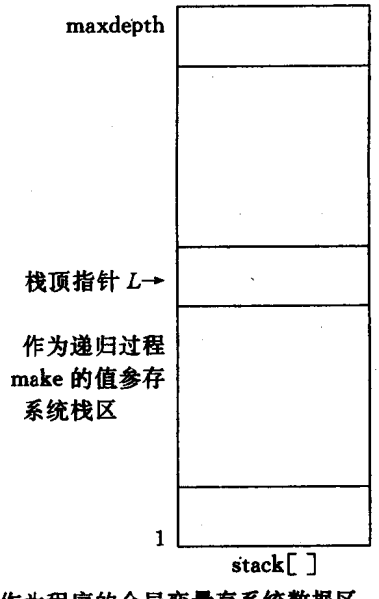

图 1-8

调用 make(L)后，栈顶指针 L 和当前结点的算符进入系统栈区，算符作用于 stack[$L-1$]结点状态，产生当前路径序列的第 L 个结点 stack[L]；退出 make(L)后，L 指针和 stack[L]结点的算符出系统栈区，回溯至调用前的父结点 stack[$L-1$]。

3. 递归过程 make(L)的边界条件是

(1) 若 stack[L]是目标结点；

(2) 若 stack[L]再无可使用的算符，即无法再扩展。

4. 只要当前结点能扩展，则递归调用 make($L+1$)，沿所在路径扩展子结点 stack[$L+1$]，直至到达边界条件为止。然后通过返回调用过程的形式回溯(恢复调用前的指针 L 和算符)，以寻求下一条路径。

5. 主程序中调用 make(1)，从初始结点出发回溯搜索。递归结束返回主程序时(此时 L 恢复为 1)表明所有路径搜索完毕。

```
program 程序名;
  ……
var{全局变量说明}
```