# 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 $8\times8$ 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($1\leqslant i\leqslant 8$)，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：

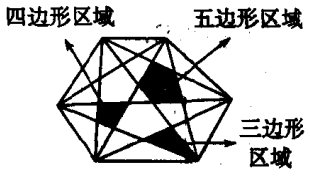



图 1-1

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

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

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。

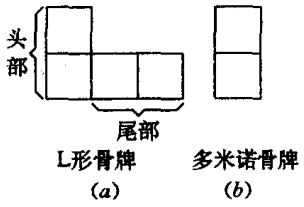


图 1-2

(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。

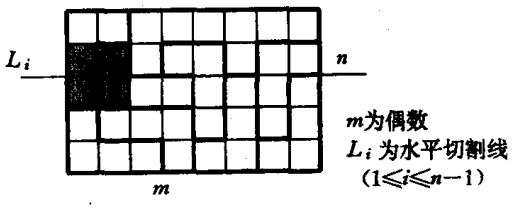


图 1-3

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($a_1$，$a_2$，

…,$a_n$)。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。

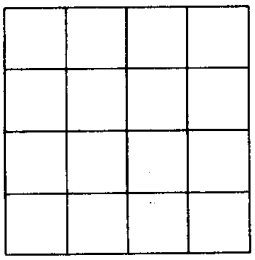


图 1-4

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。

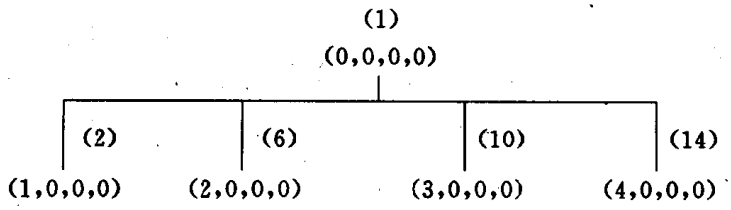



图 1-6

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。

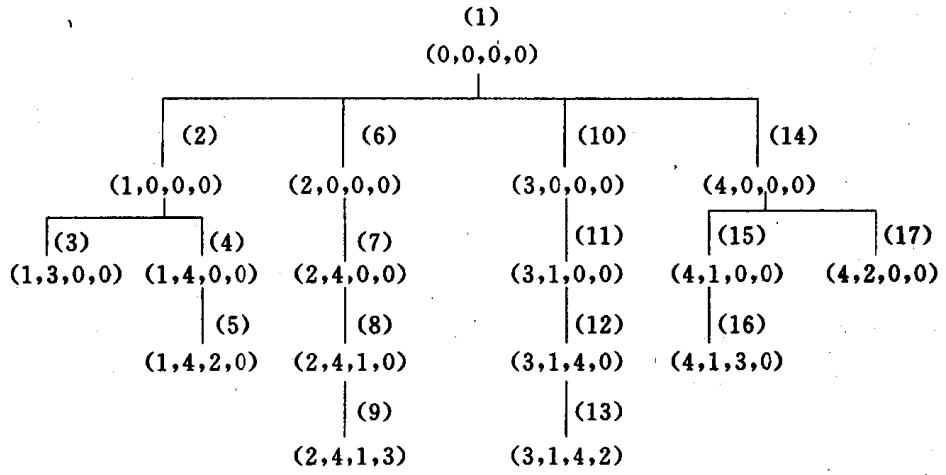



图 1-7

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。

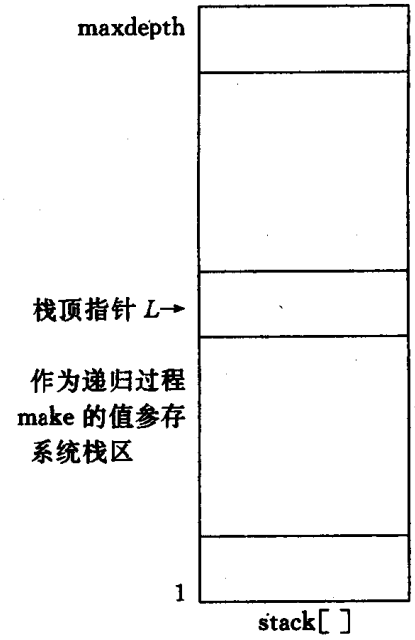
栈是一种线性表，所有进栈或出栈的数据都只能在表的同一端进行，就像堆盘子和拿盘子一样，都只能在顶端“堆上”或“取下”。这顶端叫“栈顶”，另一端叫“栈底”。Pascal 编译系统内部，保留一部分内存用作栈区，存放过程和函数的值参以及过程和函数内部所说明的局部变量。每当一个过程和函数被启用时，系统就在栈顶分配一组值参和局部变量(进栈)。而当该过程或函数退出时，这些局部变量或值参就被消除(退栈)。

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。

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[$L-1$]。



作为程序的全局变量存系统数据区

图 1-8

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}
```

```
stack:array [1..maxdepth] of node;
  ……
procedure make(L:integer);
  var
    算符的变量说明;
   ……
  begin
   ……
    make(L+1);
   ……
  end;{make}
begin
  ……
  make(1);
  ……
end.{main}
```

由于递归程序 make 中栈指针 L 和算符以值参和局部变量形式存在栈区，$L$ 直接指明当前路径序列中所有结点间的逻辑关系，比较其它搜索算法中设置指向父辈结点的指针场的做法，其存储量小，访问速度快，结构简洁。特别是得到了一条到达树叶的树枝 $t$（即求得一条路径）后，按返回调用过程的形式，沿树枝 $t$ 的路径回溯到第一个尚未用过通向右边方向的分枝点。实际上，此时的栈中不再保留下一条路径不会再用的结点，栈中结点个数即为当前路径的实际长度，因此一般不会发生内存溢出。可以说，回溯法是求所有路径试题时可采用的算法中最好的一种。

下面给出用回溯法求所有路径的算法框架。

```
program 程序名;
const maxdepth=×××;
type statetype=…;      {状态类型定义}
     operatertype=…; {算符类型定义}
     node=record       {结点类型}
          state:statetype;        {状态域}
          operater:operatertype;        {算符域}
          end;
{注:结点的数据类型可以根据试题需要简化}
var
  stack:array [1..maxdepth] of node;     {存当前路径}
  total:integer;  {路径数}
procedure make(L:integer);
  var
    i:integer;  {子结点个数}
  begin
    if stack[L-1]是目标结点 then
      begin
          total:=total+1;  {路径数+1}
          打印当前路径[1..L-1];
          exit;  {回溯。若试题仅要求一条路径,则 exit 改为 halt 即可}
      end;{then}
```

```
    for I:=1 to 解答树次数 do
      begin
        生成 stack[L].operater;
        stack[L].operater 作用于 stack[L-1].state,
                                产生新状态 stack[L].state;
        if stack[L].state 满足约束条件 then make(k+1);
      {若不满足约束条件,则通过 for 循环换一个算符扩展}
      {递归返回该处时,系统自动恢复调用前的栈指针和算符,再通过}
  {for 循环换一个算符扩展}
      {注:若在扩展 stack[L].state 时曾使用过全局变量,则应插入若干语句,}
      {恢复全局变量在 stack[L-1].state 时的值。}
       end;{for}
      {再无算符可用,回溯}
  end;{make}
begin
  total:=0;    {路径数初始化为 0}
  初始化处理;
  make(1);
  打印路径数 total;
end.{main}
```

## 三、应用算法框架解题

```
program queen;
uses crt;    {调用屏幕单元}
var
  stack:array[1..20]of byte;{ stack[i]——第 i 行皇后的列位置 1≤i≤n }
  n,total:integer;          {棋盘规模和摆法数}
procedure make(l:integer);
{从 L 行出发,递归搜索所有摆法 }
var i,j:integer;{i——子结点个数,同指列位置算符;j——辅助变量}
    att:boolean;{攻击标志}
begin
  if l=n+1 then  {产生一种成功摆法}
   begin
    inc(total); writeln('No',total);{累计方案数并输出该方案}
    for j:=1 to n do writeln(' ':stack[j],'*');
    writeln;
    exit;  {回溯,求下一方案}
   end;{then}
  for i:=1 to n do
   begin
    att:=false; stack[l]:=i;{置非攻击标志,(l,i)试放一个皇后}
    for j:=1 to l-1 do       {检查前 l-1 行,若产生攻击,置 att 为 true}
     if (abs(l-j)=abs(stack[j]-i))or(i=stack[j]) then
      begin
       att:=true; j:=l-1;
      end;{then}
    if not att then make(l+1); {若(l,i)置放皇后不产生攻击,则搜索 l+1 行}
```

```
    end;
  end;{make}
  begin
    clrscr;{清屏}
    total:=0;{方案数初始化}
    fillchar(stack,sizeof(stack),0);{栈初始化}
    write('n=');readln(n);{读入棋盘规模}
    make(1);{从第1行开始搜索所有摆法}
    writeln('Total=':9,total);{打印方案数}
  end.{main}
```

## 四、回溯法的深入

由上述解题过程可以看出，有了回溯法的算法框架，搜索一条路径或所有路径的程序就比较清晰了。但这仅是在一般情况下。若求解过程出现下述情况：

1. 求满足某个特定条件的一条最佳路径；
2. 扩展结点时参与运算的变量有多个，构成多元化的约束条件；
3. 需对当前状态分情形进行递归搜索。

就不能简单套用回溯法的算法框架了，必须具体问题具体分析。下面，我们再举一个例题。

[**例 2**] 对于任意一个 $m\times n$ 的矩阵，求 L 形骨牌覆盖后所剩方格数最少的一个方案。

算法分析：

我们曾在 1.2 节的例 5 中给出一个推论：

任意 m×n(m,n≥3)的矩阵，被 L 形骨牌覆盖后所剩的最少空格数

$$\text{Step}=\begin{cases}4 & ((m*n)\bmod 4=0)\text{and}((m*n)\bmod 8<>0);\\ (m*n)\bmod 4 & \text{其它。}\end{cases}$$

本题就是求一个剩余空格数为 Step 的覆盖方案。

我们定义结点的状态为骨牌覆盖后的矩阵状态。一个 $L$ 形骨牌覆盖某方格，该方格值为 1，否则为 0。经过旋转或翻转使之吻合的两种 L 形骨牌算是同一形状，共得 L 形骨牌的八种形状，见图 1-9。

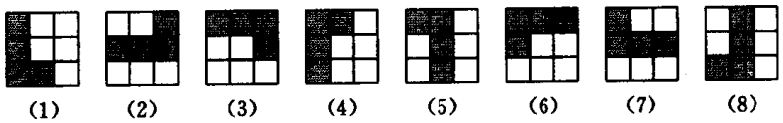



图 1-9

结点的算符为 L 形骨牌的形状序号。试放当前骨牌时，不是所有形状的骨牌都适用。若形状 $i(1\leqslant i\leqslant 8)$ 的骨牌放入以某空格为左下角的 3×3 的方阵后，伸出矩形之外或者与其它骨牌重叠，那么算符 $i$ 显然是不适用的，应当舍去。因此，不产生越界和重叠是算符采用的约束条件。

扩展结点时参与运算的变量主要有四个：

1. 搜索的方格坐标 $x$，$y$

我们按自上而下，由左至右的顺序逐格搜索。若某格$(x, y)$为空格，就在以其为左上角的3×3方阵内，逐一试放形状1..形状8的骨牌；

2. 当前骨牌覆盖后所剩的空格数Blank

搜索前，我们根据$m$，$n$值预先确定最少空格数Step，以此作为槛值。在搜索过程中，若当前的Blank超过这个边界值，则废弃最后一个L形骨牌的放法，回溯至上一格，重新试放下一形状的L形骨牌，直至满足空格数为Step的覆盖方案产生为止。

3. 已用的骨牌数$Lt$

由于是按顺序逐格搜索第$Lt$块骨牌的放入位置，因此，每搜索一格需递归调用一次。若当前格被覆盖，则递归搜索下一格$(x', y')$时，Blank和$Lt$不变；若任何形状的L形骨牌未能放入某空格对应的3×3方阵，则递归搜索下一格$(x', y')$时，Blank+1，$Lt$不变；若当前空格对应的3×3方阵放入某形状的骨牌，则递归搜索下一格$(x', y')$时，Blank不变，$Lt$+1。

显然：搜索过程与上述四个变量的当前值密切相关，因此我们将它们作为递归过程Run的四个值参。

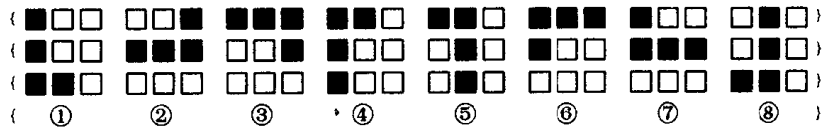
```
Run(x, y, Blank, Lt);
begin
  if Blank > Step then Exit;
  if 搜索了矩阵的所有格子 then 打印结果并退出程序;
  if (x, y)已覆盖
    then Run(x',y', Blank, Lt)
    else begin
      在(x, y)对应的3×3方阵中试放形状1..形状8的骨牌;
      if 未能放入所有形状的骨牌
        then Run(x',y', Blank+1, Lt)
        else begin
          在(x, y)对应的3×3的方阵内放入某形状的骨牌;
          Run(x',y', Blank, Lt+1);
          恢复第Lt+1块骨牌放入前的矩阵状态;
        end;
    end;
end;
```

另外，为了提高搜索效率，我们不仅在搜索前根据$m$和$n$值确定槛值$Step$，而且还先设定了矩阵的行数>列数，即$n>m$时$n$与$m$互换。在方案产生后再将覆盖方案旋转90°后输出，使得与输入保持一致。

下面，给出程序题解：

```
program L_gu_pai;

const
  Startxy : array [1..8] of 1..3
{Startxy[k]——第k种形状的L形骨牌所对应的3*3矩阵的第1行的列序号}
          =(1, 3, 1, 1, 1, 1, 1, 2);

 Ls : array [1..8, 1..3, 1..3] of 0..1
```

```
{Ls[k, i, j]——第 k 种形状的 L 形骨牌所对应的 3 * 3 矩阵中，(i,j)方格的值}
    =(((1, 0, 0), (1, 0, 0), (1, 1, 0)), ((0, 0, 1), (1, 1, 1), (0, 0, 0)),
      ((1, 1, 1), (0, 0, 1), (0, 0, 0)), ((1, 1, 0), (1, 0, 0), (1, 0, 0)),
      ((1, 1, 0), (0, 1, 0), (0, 1, 0)), ((1, 1, 1), (1, 0, 0), (0, 0, 0)),
      ((1, 0, 0), (1, 1, 1), (0, 0, 0)), ((0, 1, 0), (0, 1, 0), (1, 1, 0)));
    {Ls——L 形骨牌的八种形状：}
```




```
  Maxm=10;
  Maxn=10;

type
  SquareType = array [1..Maxm, 1..Maxn] of integer;{矩阵类型}

var
  m, n, i, j, Step : integer;     {M, N——矩阵的长和宽}
                                  {i, j——辅助变量}
                                  {Step——最佳解中的空格数}
  Change : boolean;               {交换标志}
  Square : SquareType;            {M * N 的矩阵,即覆盖方案}

procedure Init;
var i:integer;
begin
  Write('m, n =');
  repeat
    Readln(m, n);
  until (m>=1) and (m<=Maxm) and (n>=1) and (n<=Maxn);
  if (m < 2) or (n < 2) then begin
  {若 M 小于 2，或 N 小于 2，则该矩阵无法放入 L 形骨牌}
   Writeln('Blank =', m * n);  {矩阵为空}
   Writeln('Total =', 0);      {所用 L 形骨牌数为 0}
   for i:=1 to m do begin      {打印空矩阵}
    for j:=1 to n do Write(0:4);
    Writeln;
   end;
   Readln;
   Halt;{退出程序}
  end;
  {根据 M 和 N，设 Step 的值}
  if ((m * n) mod 4 = 0) and ((m * n) mod 8 <> 0)
    then Step := 4     {求出被 L 形骨牌覆盖后所剩的最少空格数}
    else Step := (m * n) mod 4;
  {交换 M 和 N 的值，使得 M >= N}
  if m < n then begin
    Change := true;{若交换了 M 和 N 的值，则置交换标志为 true}
    i := m; m := n; n := i;
```

```
    end
    else Change := false;
    FillChar(Square, Sizeof(Square), 0); {矩阵初始化为 0}
  end; {Init}

  procedure Run(Posx, Posy, Blank, Lt : integer);
  {递归搜索试放第 Lt 块 L 形骨牌的方案。(当前空格数为 Blank, 试放位}
  {置为(Posy, Posx)右下角的 3 * 3 矩阵,当前已用的 L 形骨牌数为 Lt) }

  var No : integer;

  function Can : boolean;
  {若当前位置(Posx, Posy)右下角的 3 * 3 矩阵能放入第 No 种形状的 L 形骨牌}
  {则返回 True,并生成覆盖方案 Square;否则返回 False}
  var
    sq:SquareType;
    row, col, dx, dy : integer;
  begin
    Can := false;
    dy := Posy-1;
    dx := Posx - Startxy[No];
    sq := Square;
    for row := dy + 1 to dy + 3 do
      for col := dx + 1 to dx + 3 do
        if Ls[No, row - dy, col - dx] = 1 then
          if ((row < 1) or (row > m) or (col < 1) or (col > n))
          or (sq[row, col] <> 0)
          then exit
          else sq[row, col] := Lt;
    Square := sq;
    Can := true;
  end; {Can}

  procedure Restore;
  {废弃当前 L 形骨牌的放法, 恢复 Square}
  var row, col, Max : integer;
  begin
    if Posy + 3 > m then Max := m else Max := Posy + 3;
    for row := Posy to Max do
      for col := 1 to n do
        if Square[row, col] = Lt then Square[row, col] := 0;
  end; {Restore}

  begin
    if Blank > Step then Exit; {空格数超过边界值,回溯}
    if Posy >= m + 1 then begin
    {所有行搜索完毕, 则打印空格数所用骨牌数以及覆盖方案}
      Writeln('Blank = ', Blank);
      Writeln('Total = ', Lt - 1);
      if Change
      then for j := 1 to n do begin
      {若 M 和 N 的值被交换过, 则把所得方案旋转 90 度后打印}
```

```
            for i := m downto 1 do Write(Square[i, j]: 4);
            Writeln;
          end
      else for i := 1 to m do begin
      {若 M 和 N 的值未被交换过，则把所得方案原样打印}
            for j := 1 to n do Write(Square[i, j]: 4);
            Writeln;
          end; {for}
      Readln;
      Halt; {退出程序}
    end; {then}
    if Square[Posy, Posx] <> 0
    {若当前格(Posy, Posx)已覆盖，则递归搜索下一格}
      then Run(Posx mod n + 1, Posy + Posx div n, Blank, Lt)
      else for No := 1 to 9 do
           {当前格(Posy, Posx)未覆盖，则递归试放形状 1—形状 8 的 L 形骨牌}
           if No = 9
        {所有形状的 L 形骨牌试放失败，则(Posy, Posx)为空格，递归搜索下一格}
              then Run(Posx mod n + 1, Posy + Posx div n, Blank + 1, Lt)
            else if Can then begin {若 L 形骨牌放入，则递归搜索下一格}
                   Run(Posx mod n + 1, Posy + Posx div n, Blank, Lt + 1);
                 Restore; {回溯，恢复 L 形骨牌放入前的状态}
                 end; {else}
end;

begin
  Init; {初始化}
  Run(1, 1, 0, 1);
  {从位置(1, 1)试放第 1 块骨牌开始，递归搜索最佳覆盖方案}
end. {main}
```

由于上述程序一开始就根据 $m$ 和 $n$ 值计算出最少空格数，并以此作为搜索边界，因此大大减少了搜索空间，提高了效率。如果没有这个边界值，盲目搜索所有放置方案，其笨拙程度是可以想象的。

程序设计的核心是算法分析。为了帮助读者能自如地运用组合问题的基本解题方法构思程序算法，我们将在以后各章节中给出较多的例题，以期更深刻地揭示这些方法的要领和使用技巧。同时还给出一些程序示例，介绍一些如何应用组合知识进行程序设计的经验，为读者构筑一座由组合理论通向编程实际的桥梁。

## 习 题 一

1. 一个售货亭前排着 $2N$ 个人的队伍等候购物，假定他们都购买价值 5 元的同一货物。其中 $N$ 个人持 5 元货币，$N$ 个人持 10 元货币，而售货员开始发售货物时没有零钱。问有多少种排队方式可使得售货员能依次顺利出售货物，而不出现找不出钱的局面。

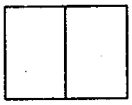
2. 现有一个 $N\times N$ 的棋盘，按由左至右、由上而下的顺序编号(如题图 1-1)。

| | | | |
|---|---|---|---|
| 1 | 2 | .... | $n$ |
| $n+1$ | $n+2$ | .... | $2n$ |
| | | | |
| | | | $n^2$ |

题图 1-1

其中棋盘上的若干格被挖空。

假设一块多米诺骨牌



恰覆盖棋盘两格，问是否可用这种骨牌互不重选地将这张残缺的棋盘完全覆盖(没有一格未被覆盖，也没有骨牌伸出棋盘或盖住挖空处)。若不能完全覆盖，则给出一个未覆盖的棋格数最少的方案。

输入：

棋盘规模 $N$；

挖去的格子数 $M$；

以下 $M$ 行为 $M$ 个被挖空的格子的编号。

输出：

使用的骨牌数 $X$；

以下 $X$ 行为 $X$ 个骨牌所覆盖的棋格位置的编号。

3. 输入 $M\times N$ 个士兵的身高，输出一个满足下列性质的 $M$ 行 $N$ 列的行军队列：

每一行从左到右身高递增；

每一列从前到后身高递增。

4. 现有 $N$ 个黑棋和 $N$ 个白棋，将这 $2N$ 个棋子放入一个 $N\times N$ 棋盘，使每行每列都有一个黑棋和一个白棋。求满足上述要求的所有布棋方案。

5. 有一座大楼，大楼有 $N$ 层，共居住 $M$ 个人。已知每个人的出发层和目的层

$A_1,B_1$

. .

. .

. .

$A_m,B_m$

$A_i$——第 $i$ 个人的出发层；

$B_i$——第 $i$ 个人的目的层。

现有最多可乘 $K$ 人的电梯 1 部，电梯从 1 楼出发，求一个最佳方案，在这个方案下，$M$ 个人顺利到达目的地且电梯经过的路程数(以一层为一路程单位)最少。

6. 在 $N\times M$ 方格棋盘中，有某一方格不布棋子，其它方格都布上棋子。跳棋规则如下：

(1) 每枚棋子在跳动时，其相邻方格(有公共边的方格)必须有一枚棋子为垫子才能

起跳；

(2) 棋子只能沿水平或垂直方向跳动一格；

(3) 棋子跳过垫子进入垫子相邻方格同一方向的空格，并把垫子取出棋盘。

编程实现

(1) 由键盘输入一个空格位置；

(2) 棋子按规则跳动，使棋子余下最少；

(3) 输出方式直观，易于查对。

7. 对于题 6 的棋盘

编程实现

(1) 求出一个空方格最少的初始状态，能使棋局依跳棋规则跳动时，棋盘棋子只余下一枚，并演示跳棋过程(说明存在一个空格位置按规则余下一枚棋子的初始状态，但不允许任意指定一个空格位置而达到目标状态)

(2) 由键盘输入：

① 2 个空格及其相应位置的初始状态；

② 棋盘余下棋子数为 1(或 2 或 3)。

输出按规则的跳子方法。

8. 设有一个布满棋子的 $N\times N$ 棋盘，跳子规则仿题 6。

编程完成任务：

(1) 把 $N\times N$ 方阵棋盘向四周扩展成 $M\times M$，试求出最小的 $M$，能使棋子依规则跳动时，棋盘内最后只余一枚棋子。并演示其跳法。

(2) 输出棋局应直观，易于查对。

# 第二章　从鸽笼原理到 Ramsey 理论

本章介绍组合数学里最简单也是最基本的原理——鸽笼原理，由此引出现今仍深刻且活跃的 Ramsey 理论，介绍它们在一些存在性问题中的应用。

## 2.1　鸽笼原理

所谓鸽笼原理是：

$n+1$ 个鸽子飞回 $n$ 个鸽笼至少有一个鸽笼含有不少于 2 只的鸽子。

也有人将鸽笼原理称为抽屉原理。

这个原理的证明谁都能完成，现抽去具体的“鸽笼”、“抽屉”等物理属性，从数学上看，就是把 $m$ 个元素分放到 $n$ 个盒子中去，当不能均匀分放时，总有一个盒子内元素数目要超过“平均数”，由此得出鸽笼原理的基本描述：

$m(m\geqslant 1)$个元素分成 $n$ 个组，那么总有一个组至少含有元素个数为

$$[m/n]$$

注：[　]为“上整数”记号，其中当 $m$ 表示为 $m=nq+\gamma(0\leqslant\gamma\leqslant n-1)$时

$$[m/n]=\begin{cases}q+1 & \text{当 } \gamma\neq 0;\\ q & \text{当 } \gamma=0。\end{cases}$$

下面先举几个浅显的例子说明利用鸽笼原理的一般步骤，从不同的灵活性中总结出带规律性的东西来。

[例 1]　13 人的小组至少有[13/12]=2 人生日在同一个月。

[例 2]　抽屉里有 10 双手套，从中取 11 只出来，其中至少有[20/11]=2 只完全配对。

[例 3]　34 位内宾参加国宴，至少有[34/33]=2 人来自同一行政区（全国目前有 33 个省、直辖市行政区）。

[例 4]　设 $q_i$ 是正整数$(i=1,2,\cdots,n)$，$q\geqslant q_1+q_2+\cdots+q_n-n+1$。如果把 $q$ 个物体放入 $n$ 个盒子中去，则存在一个 $i$，使得第 $i$ 个盒子中至少有 $q_i$ 个物体。

证明：用反证法。假设结论不成立，每个盒子都装不到应装的数。即对每一个 $i$，第 $i$ 个盒子至多放有 $n_i$ 个物体，$n_i\leqslant q_i-1$，从而这 $n$ 个盒子放入的物体总数为

$$q=\sum_{i=1}^{n}n_i\leqslant\sum_{i=1}^{n}(q_i-1)=\sum_{i=1}^{n}q_i-n<q_1+q_2+\cdots+q_n-n+1$$

这与 $q\geqslant q_1+q_2+\cdots+q_n-n+1$ 矛盾，从而结论成立。

显然鸽笼原理是 $q_i=2$ $(1\leqslant i\leqslant n)$时的情形，而例 4 是鸽笼原理的一般形式。

下面，逐渐介绍鸽笼原理较复杂的一些应用。从这一看似显而易见的原理出发，可以导出许多组合数学中并不那么显而易见的有趣结论。

[例 5]　边长为 1 的正方形内部任置 5 个点，则其中必有 2 个点，它们之间的距离小于或

等于 $\sqrt{2}/2$。

证明：过正方形中心作两条与边平行的对称轴，将正方形分成 4 个全等的小正方形，见图 2-1。

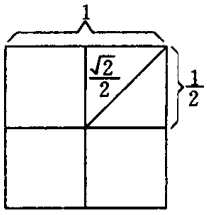


图 2-1

这样，总有一个小正方形内至少含$\lceil 5/4 \rceil = 2$ 个点，它们之间相距小于或等于 $\sqrt{2}/2$。证毕。

[**例 6**] 设 $m$ 是取定的一个自然数。求证，任取 $m+1$ 个整数，则其中至少有 2 个整数，其差是 $m$ 的倍数。

证明：解题的思路是：差为 $m$ 的倍数的两个整数，除以 $m$，有相同的余数。设 $m$ 是给定的一个正整数，任意一个整数 $n$(也包括负整数)被 $m$ 除，得到一个商和余数。余数为 0 也就是通常说的整除，否则余数取为 $1,2,\cdots,m-1$ 之间。总之可写成

$$n = q \cdot m + \gamma \qquad (0 \leqslant \gamma \leqslant m-1)$$

这样，全体整数按余数相同的归入一类，共划分成 $m$ 个类，余数为 $i$ 个类记为 $I_i$，这 $m$ 个剩余类记成

$$I_0, I_1, \cdots, I_{m-1}$$

把模 $m$ 的 $m$ 个剩余类取名为“鸽笼”，$m+1$ 个数放入 $m$ 个鸽笼，至少有 2 个数在同一个剩余类内，故这 2 个数之差为 $m$ 的倍数。

[**例 7**] $n^2+1$ 个正整数序列$\{a_1, a_2, \cdots, a_{n^2+1}\}$，此处对一切 $i, j, a_i \neq a_j$，那么必存在项数为 $n+1$ 的单调上升或单调下降的子序列。

证明：设 $L_i$ 为从 $a_i$ 开始最长的上升子序列的项数。

若某个 $L_i \geqslant n+1$，则结论成立；

若全部 $L_i < n+1 (i=1,2,\cdots,n^2+1)$，则 $n^2+1$ 个 $L_i$ 分属 $1,2,\cdots,n$。由鸽笼原理，至少有$\lceil (n^2+1)/n \rceil = n+1$ 个 $L_i$ 是相等的。不妨说 $a_{i1}, a_{i2}, \cdots, a_{in+1}$有相同的 $L$ 值，则必有 $a_{i1} > a_{i2} > \cdots > a_{in+1}$。若不然，譬如 $a_{i1} < a_{i2}$，则以 $a_{i2}$开始的上升子序列至少有 $L_{i2}$项，再在前面接上 $a_{i1}$，则 $L_{i2} = L_{i1}+1$，与 $a_{i1}, a_{i2}$有相同的 $L$ 值矛盾。证毕。

本结论中 $n^2+1$ 已不能再降、达到临界值了。例如 $n=4, 4^2=16$ 项整数可以不含长为 5 的子序列。$\{a_1, a_2, \cdots, a_{16}\}$可以取为$\{\underline{4、3、2、1}, \underline{8、7、6、5}, \underline{12、11、10、9}, \underline{16、15、14、13}\}$。其它 $n$ 值也可用同样的思路构造。

[**例 8**] 完全图 $k_6$(6 个顶点且每两个点都连接有一条边的图)，对它的边用红、蓝两种颜色任意涂色，则存在同色边的三角形。如果顶点代表一个人，任两人如认识就用红边连接，否则用蓝边连接。上述问题即可转化为另一个问题。

“任意 6 个人聚会，必发生或有 3 人互相认识，或有 3 人互不认识”。

证明：设 $k_6$ 的顶点为 $V_1, V_2, \cdots, V_6$，从 $V_1$ 引出的 5 条边中，由鸽笼原理知其中至少有

$$\lceil 5/2 \rceil = 3$$

条同色边。不妨设$(V_1, V_2), (V_1, V_3), (V_1, V_4)$这三条边涂上红色。再看$\triangle V_2V_3V_4$，如它的 3 条边涂红色，则结论成立；如不然，仍由鸽笼原理得出至少有 1 条红边，这红边两端再配上 $V_1$ 点，必组成一个 3 边红色三角形。证毕。

## 2.2 Ramsey 问题和 Ramsey 数

用红蓝两种颜色去涂 $n$ 个顶点完全图的边，每边涂且仅涂一种颜色，得到的图叫做 2 色完全图，仍记为 $k_n$。当然由于边的涂色方式不同，得到的 2 色完全图也不同。图 2-2 给出两个不同的 2 色完全图 $k_5$（图中实线表示红边，虚线表示蓝边）。

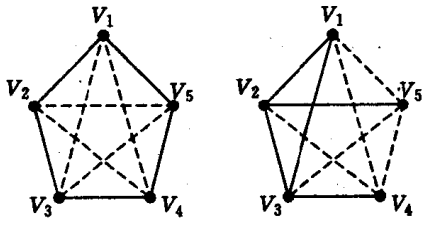



图 2-2

图 2-2 说明用红蓝两色去涂完全图 $k_5$ 的边，不能保证涂出来的 2 色完全图 $k_5$ 一定含有单色三角形。而 2.1 节的例 8 说明，任何一个 2 色完全图 $k_6$ 都会含有单色三角形。

现在，我们可以把上述结论加以推广：

用 $\gamma(p,g)$ 表示这样的正整数，即当 $n\geqslant\gamma(p,g)$ 时，任何一个 2 色完全图 $k_n$，或者含有红色完全图 $k_p$（即 $k_p$ 每条边都是红的），或者含有蓝色完全图 $k_g$，两者必居一；而当 $n<\gamma(p,g)$ 时，存在 2 色完全图 $k_n$，它不含红色完全子图 $k_p$ 和蓝色完全图 $kg$。数 $\gamma(p,g)$ 叫做 Ramsey 数。

我们还可以换一种更一般化的说法：一对常数 $p$ 和 $g$ 对应一个常数 $n$，使得 $n$ 个人中或有 $p$ 个互相认识，或有 $g$ 个互不认识，这个 $n$ 的最小值用 $\gamma(p,g)$ 表示。

显然 $\gamma(1,g)=1,\gamma(2,g)=g,\gamma(p,g)=\gamma(g,p)$。

**[例 1]** 某俱乐部有个约定，4 人围坐在圆桌旁一起打桥牌，必须每人和两旁的人曾合作过，或者都不曾合作过。证明只要俱乐部里有 6 名成员，就一定能凑集 4 人在一起打桥牌，如果俱乐部只有 5 名成员，结论成立吗？

**证明**：我们将俱乐部成员作为图的顶点，若两人曾合作（或不曾合作），则在相应两顶点之间连一条边。根据每人和两旁的人曾合作（或不曾合作）的规则，4 人打桥牌可组成两种四边形 $C_4$，见图 2-3。

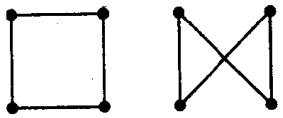


图 2-3

欲证 $\gamma(C_4,C_4)=6$。由于 $\gamma(3,3)=6$，不妨设 $k_6$ 含有红三角形 $\triangle V_1V_2V_3$。于是由图 2-4 可知 $k_6$ 含单色四边形，因此 $\gamma(C_4,C_4)\leqslant 6$。而图 2-5 表明 $\gamma(C_4,C_4)\geqslant 6$

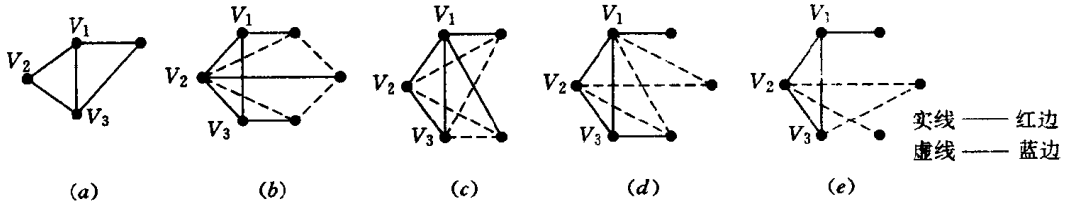



图 2-4

由此可见，俱乐部起码有 6 名成员才一定能按约定凑集 4 人打牌，而 5 名成员是不能担保打牌人数的。

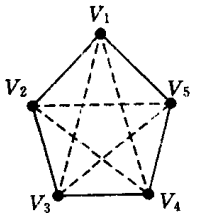



图 2-5

下面将 $\gamma(p,g)$ 推广至 $k$ 种颜色的情形。用 $k$ 种颜色 $C_1$，$C_2,\cdots,C_k$ 去涂完全图 $k_n$ 的边，每边涂且只涂一种颜色，得到的图叫做 $k$ 色完全图，仍记为 $k_n$。用 $\gamma(n_1,n_2,\cdots,n_k)$ 表示这样的正整数，即当 $n\geqslant\gamma(n_1,n_2,\cdots,n_k)$ 时，任何一个 $k$ 色完全图含有 $C_1$ 色完全子图 $k_{n1}$，或者 $C_2$ 色完全子图 $k_{n2}$，或者 $C_k$ 色完全子图 $k_{nk}$；而当 $n<\gamma(n_1,n_2,\cdots,n_k)$ 时，存在 $k$ 色完全图 $k_n$，它不含 $C_i$ 色完全子图 $k_{ni}$，$i=1,2,\cdots,k$。数 $\gamma(n_1,n_2,\cdots,n_k)$ 仍为 Ramsey 数。不难证明 $\gamma(n_1,n_2,\cdots,n_k)$ 存在。

当 $n_1=n_2=\cdots=n_k=3$ 时，记 $\gamma(\underbrace{3,3,\cdots,3}_{k\text{个}})=\gamma_k$，数 $\gamma_k$ 叫做经典 Ramsey 数。显然 $\gamma_1=3,\gamma_2=\gamma(3,3)=6$。

**[例 2]** 17 名科学家，每名科学家通信时只讨论三个题目。证明其中至少三名科学家，他们相互通讯时讨论的是同一题目。

用顶点表示科学家，用颜色 $C_1,C_2,C_3$ 表示三个题目 $A_1,A_2,A_3$，并且当两名科学家讨论的是 $A_i(i=1,2,3)$ 时，相应两顶点过一条 $C_i$ 色边，则这道题就要证明一个 3 色完全图 $k_{17}$ 一定含有单色三角形，即欲证 $\gamma_3\leqslant 17$。

证明：取完全图 $k_{17}$ 的顶点 $V$，它连有 16 条边，共 3 种颜色，因此至少有 6 条边同色，不妨设 $VU_1,VU_2,\cdots,VU_6$ 是红边。$k_{17}$ 中以 $U_1,U_2,\cdots,U_6$ 为顶点的完全子图 $k_6$ 中，如果含有红边 $U_iU_j(1\leqslant i\neq j\leqslant 6)$，则 $k_{17}$ 含有红三角形 $\triangle VU_iU_j$。因此设 $k_6$ 不含红边，$k_6$ 是 2 色完全图。由“任何一个 2 色完全图 $k_6$ 都含单色三角形”的定理可知，$k_6$ 含有蓝三角形，或者黄三角形，从而 $k_{17}$ 含有单色三角形，于是 $\gamma_3\leqslant 17$。证毕。

由于经典 Ramsey 数 $\gamma_k$ 的定义，可以引出一个有趣的数学结论：

**[例 3]** 把自然数集合 $N=\{1,2,\cdots,n\}$ 分成 $k$ 个两两不交的子集 $S_1,S_2,\cdots,S_k$。则当 $n$ 适当大时，一定存在某个子集 $S_i,1\leqslant i\leqslant k$，它同时含有两个正整数 $x,y$ 及它们的和 $x+y$。

证明：用 $n+1$ 个顶点表示自然数 $0,1,\cdots,n$，用 $k$ 种颜色 $C_1,C_2,\cdots,C_k$ 去涂以 $0,1,\cdots,n$ 为顶点的完全图 $k_{n+1}$。当正整数 $x,y$ 满足 $x-y$ 或 $y-x\in S_i$ 时，边 $xy$ 为 $C_i$ 色。由 Ramsey 数 $\gamma_k$ 的意义，当 $n+1\geqslant\gamma_k$，即 $n\geqslant\gamma_k-1$ 时，$k$ 色完全图 $k_{n+1}$ 含有单色三角形。设 $\triangle xyz$ 是 $k_{n+1}$ 中 $C_i$ 色三角形，其中 $x>y>z$，则正整数 $a=x-y,b=y-z$ 与 $c=x-z$ 都属

于 $S_i$。显然 $c=a+b$。因此 $S_i$ 同时含有正整数 $a$、$b$ 与 $a+b$。证毕。

确定 Ramsey 数是很难的问题。到目前为止，主要还是研究 $\gamma(p,g)$，精确求得的数值为数甚少。下面，我们给出两个 $\gamma(p,g)$ 的上界估计式：

(1) $R(p,g)\leqslant R(p,g-1)+R(p-1,g)$

证：完全图 $k_n$，$n=\gamma(p,g-1)+\gamma(p-1,g)$，对边作任意的红蓝 2 涂色。任取 $k_n$ 的一个顶点 $U$，$U$ 的用红色边连接的顶点全体记为 $R_u$，$U$ 的用蓝色边连接的顶点全体记为 $B_u$，显然 $|R_u|+|B_u|=N-1=R(p,g-1)+R(p-1,g)-1$。根据鸽笼原理的推广形式(2.1 节中的例 4)可推出：

或者有 $|B_u|\geqslant R(p,g-1)$ 或者 $|R_u|\geqslant R(p-1,g)$。

由对称性，不妨考虑 $|R_u|\geqslant R(p-1,g)$ 的情况，由 Ramsey 的定义，在 $R_u$ 中全部顶点构成的完全子图 $k_{|R_u|}$ 中，保持原来的边的红蓝 2 涂色，则或者有 $p$ 个顶点的红色边完全子图，或者有 $g$ 个顶点的蓝色边的完全子图。当发生前者时，只要把 $U$ 点加上 $R_u$ 中这 $p-1$ 个顶点，就成了 $k_n$ 中有 $p$ 个顶点的红色边完全子图。证毕。

(2) 若在上述估计式中，$R(p,g-1)$ 和 $R(p-1,g)$ 均为偶数，则 $R(p,g)$ 的上界估计式还可改进为

$$R(p,g)\leqslant R(p,g-1)+R(p-1,g)-1$$

证：记 $N_1=R(p,g-1)+R(p-1,g)-1$，对 $k_{n1}$ 的边红蓝 2 涂色，则必存在一个顶点 $U$，它只关联偶数条同色边。因为 $N_1$ 为奇数，根据图论中关于“奇度数的顶点总共有偶数个”的结论，必得出这样的结果。

由对称性，不妨设 $U$ 顶点关联红色边偶数条。$k_{n1}$ 中的顶点划分为 $\{U\}$、$\{R_u\}$、$\{B_u\}$，显然有

$$|R_u|+|B_u|=N_1-1=R(p-1,g)+R(p,g-1)-2$$

则有鸽笼原理，或有 $|B_u|\geqslant R(p,g-1)$

或有 $|R_u|\geqslant R(p-1,g)-1$

但是 $|R_u|=U$ 顶点关联的红边数＝偶数，$R(p-1,g)-1$＝奇数，所以上式可以改进成

$$|R_u|\geqslant R(p-1,g)$$

从而获得更好一些的估计式。其余推论完全类似上界估计式 1 的证明。证毕。

有了上述两个递归的上界估界式，加上递归边界 $\gamma(1,g)=1$，$\gamma(2,g)=g$，$\gamma(p,g)=\gamma(g,p)$，我们就可以编写估计 $\gamma(p,g)$ 上界的程序了。但必须指出，程序只能对 $\gamma(p,g)$ 数进行上界估计，一些运行结果与精确值有一定误差。要准确计算 $\gamma(p,g)$ 数，只要将程序返回的整数值逐一递减($n\leftarrow n-1$)，直至递减后的 $n$ 值所对应的 $k_n$ 图中出现了不含红色完全子图 $k_p$ 或蓝色完全子图 $k_g$ 的情形，则 $n+1$ 就是精确的 $\gamma(p,g)$ 值了。当然，这种搜索效率是极其费时间的，仅对较小的 $p$ 和 $g$ 值才有可能实现。

```
program ramsey;
  uses
    crt;
  const
```

```
  maxn              =50;
type
  rtype          =array[1..maxn,1..maxn] of integer;
var
  r                 : rtype;          {ramsey 数组}
  a,b               : integer;        {ramsey 数的两个参数}
procedure init;     {输入 ramsey 数的两个参数}
  begin
    clrscr;
    repeat write('a=');
           readln(a);
    until (a>1) and (a<=maxn);
    repeat write('b=');
           readln(b);
    until (b>1) and (b<=maxn);
  end;
procedure main;
  var i,j : integer;
  begin
    for i:=2 to a do r[i,2]:=i;  {建立递归边界}
    for i:=2 to b do r[2,i]:=i;
    for i:=3 to a do
      for j:=3 to b do
        if (odd(r[i-1,j]) or (odd(r[i,j-1])
          then r[i,j]:=r[i-1,j]+r[i,j-1]
          else r[i,j]:=r[i-1,j]+r[i,j-1]-1;
      writeln('R(',a,',',b,')=',r[a,b]);
    end;
begin
  init;  {输入参数 a,b}
  main;  {计算和输出 ramsey 数 r(a,b) }
end.
```

## 习 题 二

1. 已知 $N$ 个正整数 $a_1,a_2,\cdots,a_n$，证明，这 $N$ 个数中总是可以选择两个数使得这两个数的和或差能被 $N$ 整除。

2. 设 $a_1,a_2,\cdots,a_n$，是 $1,2,\cdots,N$ 的一个排列，$N$ 是奇数。证明$(a_1-1)(a_2-2)\cdots(a_n-n)$是一个偶数。

3. 证明：若将 $k_7$ 的边着红色和蓝色，则至少有 3 个同色三角形

4. 证明：若将顶点数为 $2N$ 的完全图的边着红色和蓝色，则同色三角形的数量至少是 $2C(N,3)$

# 第三章　排列组合及其计数问题

所谓计数问题，就是计算具有某种特性的对象有多少。长期以来，数学家们为各种计数问题设计了准确的数学公式，而今人们大量使用计算机从事计数运算，其意义在于：

1. 对于运算量或数据量较大的计数，计算机编程运算显然比手算要精确和简捷得多；

2. 计算机既能够依据公式计算具有某种特性的对象有多少，又能够把它们完全列举出来。若用手算枚举，则是一件极困难的事。

计数的基本原理是加法原理和乘法原理。而第四章的容斥(即包含排斥)原理是加法原理的推广。排列组合是计数中最常见和最简单的基本问题。

## 3.1　两个基本计数原理

### 一、加法原理

如果完成一件事情的 $n$ 类进行方式 $A_1, A_2, \cdots, A_n$，每一类进行方式 $A_i$ 中有 $m_i$ 种方法($1 \leqslant i \leqslant n$)，而任何两类进行方式 $A_i$ 与 $A_j$($1 \leqslant i, j \leqslant n, i \neq j$)都互不相同(见图 3-1)，且不论采用这些方法中的任何一种都能单独地完成这件事情，那么要完成这件事共有

$$N = m_1 + m_2 + \cdots + m_n$$

种方法。

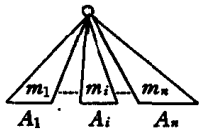



图　3-1

提醒读者注意的是，当 $A_i$ 与 $A_j$ 有重叠(方法相同)时，需要更深一层的原理——容斥原理来计数。

### 二、乘法原理

如果完成一件事情要分几个步骤 $B_1, B_2, \cdots, B_N$，而每一个步骤 $B_i$ 有 $m_i$ 种方法($1 \leqslant n$)(见图 3-2)，那么完成这件事共有

$$N = m_1 \cdot m_2 \cdot \cdots \cdot m_i \cdot \cdots \cdot m_n$$

种方法。

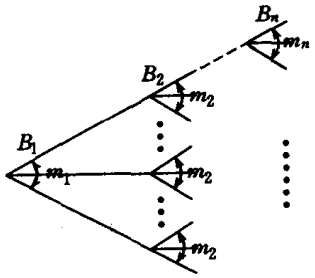
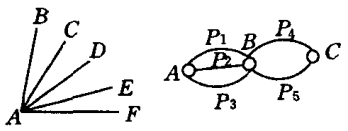



图　3-2

[例 1] 求图 3-3($a$)中的道路数和图 3-3($b$)中从 $A$ 经 $B$ 到 $C$ 的道路数。




($a$) ($b$)

图 3-3

**解**：前者用加法原理，立即得 5 条道路，后者用乘法原理，共 6 条道路，即

$<P_1,P_4>,<P_1,P_5>$

$<P_2,P_4>,<P_2,P_5>$

$<P_3,P_4>,<P_3,P_5>$

[例 2] 国际会议洽谈贸易，有 5 家英国公司，6 家日本公司，8 家中国公司，彼此都希望与异国的每个公司单独洽谈一次，问要安排多少个会谈场次？

**解**：每两国会议次数用乘法原理

中英会谈场次＝5×8＝40

英日会谈场次＝5×6＝30

中日会谈场次＝6×8＝48

由于上述三类会谈互不相交，求安排的会谈总场次数用加法原理：

40＋30＋48＝118 个场次

## 3.2 排　　列

研究排列问题的主要目的是求出根据已知条件所能作出的不同排列的种数。排列按照元素的排列方式可分为三种排列

1. 线排列；
2. 圆排列；
3. 重排列。

### 一、线排列

先考察一个简单的问题，有红、蓝、白三只球，要放到编号为 1，2，…，10 的十个盒子中，如果每个盒子只能装一只球，我们想知道把球放到盒子中的不同放法的种数。

让我们来放这些球，每次放一只球，依次顺序为：放红球→放蓝球→放白球。

放红球的方法为 10 种　　（红球可放到 10 个盒子的任一个中去）

放蓝球的方法为 9 种　　（蓝球可放到剩下的 9 个盒子的任一个中去）

放白球的方法为 8 种　　（白球可放到剩下的 8 个盒子的任一个中去）

根据乘法原理，这些球的不同放法总共有 10×9×8＝720 种

从这个例子的结果可以推广到一般，得出线排列的定义：

从 $n$ 个不同的元素中，取 $r$ 个按次序排列，称为从 $n$ 中取 $r$ 个排列，其排列数记作 $P(n,r)$。

这里值得注意的是，所谓按次序排列无非就是提醒我们，如果两个排列的元素相同而且排列次序也完全相同，就是两个相同的排列，只能算作一种排列。换句话说，如果两个排列所包含的元素及排列的次序，只要两者有一个不相同，它们就是两种不同的排列。而 $P(n,r)$就是求从 $n$ 个元素中取元素个数为 $r$，但排列次序不同的排列数。

从 $n$ 中取 $r$ 个排列的典型模型是把 $r$ 个不同颜色的球放到 $n$ 个编号不同的盒子中去，而且每个盒子只能放一只球。很显然，这些球的不同放法总数是

$$P(n,r) = n(n-1)(n-2)\cdots(n-r+1)$$

亦可写成

$$P(n,r) = n!/(n-r)!$$

当 $r=0$ 时，一个元素也不取，算作是取 0 个元素的一种排列，即 $P(n,0)=1$；当 $r=n$ 时，有 $P(n,n)=n!$ 称作全排列；而把 $0<r<n$ 的情况称作选排列。

从下面几个例子中我们可以看到，把球放到盒中去这个问题的讨论不是毫无意义的。

[例 1] 在五天之内安排三次考试，且不允许一天内有两次考试，那末一共有多少种安排法？

**解**：假定把三次考试看作三只颜色不同的球，五天看作五个编号不同的盒子，那末我们得到的结果是

$$P(5,3) = 5 \times 4 \times 3 = 60 \text{ 种}$$

[例 2] 确定各位数中不重复的四位十进制数的个数

**解**：从 0,1,…,9 的十个数中选四个数的排列数可以看作是把四个颜色不同的球放入十个标号盒的不同放法数：

$$P(10,4) = 5040(\text{个})$$

但这些数中的以 0 开头的数为

$$9 \times 8 \times 7 = 504(\text{个})$$

其中，9、8、7 为千位数为 0 的情况下，百位数、十位数、个位数上可选数字数。

因此，不是以 0 开头的四位数有

$$5040 - 504 = 4536$$

## 二、圆排列

从集合 $S=\{a_1,a_2,\cdots,a_n\}$的 $n$ 个不同元素中，取出 $r$ 个元素按照某种次序(如逆时针)排成一个圆圈，称这样的排列为圆排列。

需要注意的是一个圆排列旋转可得另一个圆排列，这两个圆排列是相同的，例如取出 $r$ 个元素 $a_1,a_2,\cdots,a_r$ 的圆排列，可以旋转得出 $r$ 个线排列，即

$$a_1a_2\cdots a_r, a_2\cdots a_ra_1, a_3\cdots a_ra_1a_2, \cdots, a_ra_1a_2\cdots a_{r-1}$$

这 $r$ 个线排列在圆排列中只能算同一个。一个圆排列可以产生 $r$ 个线排列，而总共有 $P(n,r)$个线排列，因此圆排列的个数为

$$P(n,r)/r = n!/(r(n-r)!)$$

[例 3] 有 8 人围圆桌就餐，问有多少种就座方式？如果有两人不愿坐在一起，又有多少种就座方式？

**解**：$n=8, r=8$，因此 8 人围圆桌就餐的就座方式有

$$\frac{P(8,8)}{8}=\frac{8!}{8}=7! \quad \text{种就座方式}$$

设不愿坐在一起的两人为甲和乙。当甲乙坐在一起时，相当于 7 人围桌而坐，其就坐方式为$\frac{7!}{7}(n=7,r=7)$。而甲乙坐在一起时，只有两种情况，或甲坐乙右边，或甲坐乙左边。这样一来，甲和乙坐在一起时共有 $2\times\frac{7!}{7}=2\times6!$ 就坐方式。因此，甲和乙不坐在一起时共有就座方式的种数为

$$7!-2\times6!=3600 \quad \text{种方式}$$

[例 4] 4 男 4 女围圆桌交替就坐有多少种方式？

**解**：显然，这是一个圆排列问题。先让四男围桌而坐，共有 4! /4 种就座方式。然后加入一个女的进去就座就有 4 种方式，加入第二个女的又有 3 种方式，加入第三个女的又有 2 种方式，加入第四个女的只有 1 种方式。由乘法原理知，四男四女交替就坐的方式数为

$$\frac{4!}{4}\times4\times3\times2\times1=144 \text{ 种就座方式}$$

## 三、重排列

上面我们讨论了从 $n$ 个互不相同的元素组成的集合 $S=\{a_1,a_2,\cdots,a_n\}$ 中选 $r$ 个元素进行排列、在每种排列中每个元素至多只出现一次的情况。现在考虑允许重复出现的情况，即考虑在重集 $S=\{k_1\cdot a_1,k_2\cdot a_2,\cdots,k_n\cdot a_n\}$ 中选 $r$ 个元素进行排列。根据重复数 $k_1k_2\cdots k_n$ 是否为 $\infty$，重排列又可分为无限重排列和有限重排列两种。

### 1. 无限重排列

现在，我们再回到把三个不同颜色的球放到十个不同的盒子中去，假定一个盒子能容纳球的只数不限。因为蓝球和白球也能像红球一样放入十个盒子中的任一个，所以放置的方法总数是

$$10\times10\times10=1000$$

一般地，从 $n$ 个不同元素中取 $r$ 个按次序排列。若每个元素无限次重复（即 $k_1=k_2=\cdots=k_n=\infty$）则称排列为无限次排列，其排列数等于 $n^r$。

这是因为把每次取出的 $r$ 个元素分别排在 $n$ 个位置上时，在第一个位置上，可以从 $n$ 个元素里任选一个来排，有 $n$ 种方法；在第二个位置上，由于允许重复选取，仍然可以从这 $n$ 个元素里任选一个来排，也有 $n$ 种方法；同理，在其余位置上也都有 $n$ 种方法。根据乘法原理，排列总数是：

$$\underbrace{n\cdot n\cdots n}_{r\text{个}}=n^r$$

它的典型模型是把 $r$ 只不同颜色的球，放到编号不同的盒子中去，如果一个盒子能放

球的只数不限，那么将 $r$ 只不同颜色的球放到 $n$ 个编号不同的盒子中的方法有 $n^r$ 种。

[例 5] 无线电收发报机有‘·’‘—’(短、长)两种信号，用四个信号代表一个阿拉伯数码，问可以表示多少个不同码？

通用明码用 0～9 十位数码中的四个数码表示一个汉字，可以表示多少个不同汉字？

**解**：对无线电收发报机来说，有两种信号，取四个表示一个数码，这是一种重复排列，在四个位置上的每个位置都有两种排法，可以看作是在两个不同的元素中，每次取出四个元素的有重复的排列，因此总共可以表示不同的数码个数是 $2^4=16$。

由于十进位记数法中只有十个不同的数码，因此这样就完全够用了。

同理，通用明码用四个数码表示一个汉字，可以表示出的汉字个数为 $10^4=10000$。若不允许重复，四个十进制码仅能表示 $P(10,4)=5040$ 个汉字，对于普通用语则嫌不足，对有些用于姓名或地名的特殊汉字来说，就更不够了。

### 2. 有限重排列

若允许元素有限次重复，那 $S=\{k_1a_1,k_2a_2,\cdots,k_na_n\}(k_i\neq\infty)$，则从所有元素中取 $r$ 个有次序排列的种数又如何计算呢？我们还是先从一个例题开始讲起：

[例 6] 把两只红球、一只蓝球和一只白球放到编号不同的十个盒子中去的方法总数是多少？

**解**：将两只红球涂成深红与浅红，使之变成可区别，于是将这四只不同颜色的球放到十个盒子中去的方法种数是 $P(10,4)=5040$，在这 5040 种放法中，我们是按图 3-4 的两种方法进行：

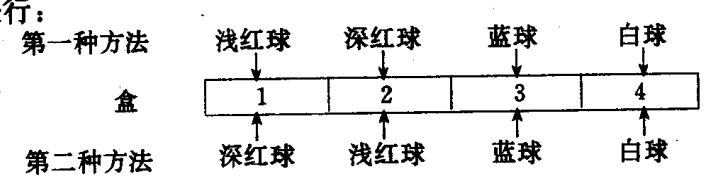



图 3-4

如果不区分红的深浅，那么两种方法变成一种方法，因此把两只红球、一只蓝球、一只白球放入十个编号不同的盒子中去的方法种数有 $5040/2=2520$。

一般地，把 $r$ 只彩色球放到 $n$ 个编号不同的盒子中去的方法种数是

$$\frac{P(n,r)}{r_1!r_2!\cdots r_t!}$$

其中 $r_i$ 表示第 $i$ 种彩球有 $r_i$ 只，$i=1,2,\cdots,t$，$r=r_1+r_2+\cdots+r_t$ 当 $r=n$ 时，有

$$\frac{n!}{n_1!n_2!\cdots n_t!}$$

其中 $n_1$ 表示 $n_1$ 个相同元素，$\cdots$，$n_t$ 表示 $n_t$ 个相同元素且 $n_1+n_2+\cdots+n_t=n$

[例 7] 某市区中的一处棋盘形街道，有南北方

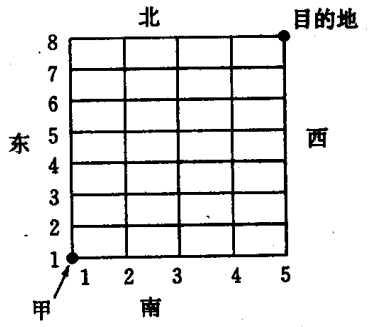



图 3-5

向街道(纵街)5 条,东西方向街道(横街)8 条,见图 3-5,某甲从东南角走到西北角,要按最短路径走(就是只能向西或向北走),共有几种走法?

**解**:某甲要按最短路径由东南角走至西北角,只要走过 7 段纵街和 4 条横街即可,而先走横街还是先走纵街的次序可以不同,即路线可以不同。因此,我们只要研究这 7 段纵街和 4 段横街能排成多少种不同的次序,即有几种不同的走法。在这 11 个元素中有 7 个相同元素(都是纵街)另外又有 4 个相同元素(都是横街),所以按有限重复的排列次数公式,可以求得不同走法的种数是

$$N = \frac{11!}{7!4!} = 330(\text{种})$$

## 3.3 组　　合

研究组合的主要目的之一是求出根据已知条件所能作出的不同组合的种数。组合按照元素的组合方式可分为二种组合

1. 非重组合;
2. 重组合。

### 一、非重组合

所谓非重组合,就是组合中的每一个元素最多只能允许出现一次。那么什么是组合呢?我们还是先从一个例题谈起:

[**例 1**] 有三只全是红色的球放到十个编号不同的盒子中去,如果每个盒子只能放一只球,求把球放到这种盒子中去共有多少种放法?

**解**:我们可以把上述问题看作是 10 个元素中有 3 个相同元素,则从 10 个元素中取 3 个元素的排列数为

$$P(10,3)/3! = 120(\text{种})$$

从这个数值例子的结果可以推广到一般,引出组合的定义:

从 $n$ 个不同元素中,取 $r$ 个而不考虑次序时,称为从 $n$ 中取 $r$ 个组合,其组合数记作 $C(n,r)$。

值得注意的是,所谓不考虑次序无非是提醒我们,如果两个组合中的元素相同,不管元素的次序如何,都是相同的组合,即算作一种组合。只有当两个组合中元素不完全相同时,才是不同的组合。而 $C(n,r)$就是求所有元素个数为 $r$ 且各组合中的元素不尽相同的组合数。

从 $n$ 中取 $r$ 个组合的模型是把 $r$ 只相同颜色的球放到 $n$ 只编号不同的盒子中去,其方法种数是

$$C(n,r) = \frac{n(n-1)\cdots(n-r+1)}{r!} = \frac{n!}{(n-r)!r!} = \frac{P(n,r)}{r!}$$

即从 $n$ 个不同元素中任选 $r$ 个元素的组合数就等于从 $n$ 个不同元素中任选 $r$ 个元素的排列数除以 $r$ 个元素的全排列。

这里，有两种特殊情况：

(1) 一个元素也不取的组合也算作是一个组合，即 $C(n,0)=1$；

(2) 取全部元素的组合也算作是一个组合，即 $C(n,n)=1$；

[例 2] 在 15 个学生中间选一个 4 人代表队参加国际奥林匹克信息学竞赛，使得学生 $A$ 和学生 $B$ 至少有一个必须在 4 个成员的代表队内，共有多少种选法？

**解**：在 15 个学生中选一个 4 人代表队的数目是 $C(15,4)$；

把 $A$ 和 $B$ 都排除在外的代表队数目是 $C(13,4)$；

因此选法的总数是

$$C(15,4) - C(13,4) = 650(\text{种})$$

[例 3] 从 1～300 之间任选 3 个不同的数，使得这 3 个数的和正好被 3 除尽，问共有几种方案？

**解**：将 1～300 的 300 个数分成 3 类：

(1) 被 3 除的余数为 1 的数集 $A=\{1,4,7,\cdots,298\}$，$|A|=100$；

(2) 被 3 除的余数为 2 的数集 $B=\{2,5,8,\cdots,299\}$，$|B|=100$；

(3) 被 3 除的余数为 0 的数集 $C=\{3,6,9,\cdots,300\}$，$|C|=100$。

任取三个数，其和正好被 3 除尽的有如下两种情况：

(1) 三个数或同属 $A$ 或同属 $B$ 或同属 $C$，应有

$$C(100,3) + C(100,3) + C(100,3) = 3C(100,3) \text{ 种；}$$

(2) 三个数分别属于集合 $A,B,C$，根据乘法原理，应有

$$100 \times 100 \times 100 = 100^3 \text{ 种；}$$

综上所述，根据加法原理，任选三个不同的数，它们的和正好被 3 除尽的方案种数为

$$N = 3C(100,3) + 100^3 = 1485100$$

下面，我们通过计算机编程，解决一道组合例题的计数和枚举问题：

[例 4] 某机要部门安装了电子锁。$M$ 个工作人员每人发一张磁卡，卡上有开锁的密码特征。为了确保安全，规定至少要有 $N$ 个人同时使用各自的磁卡才能将锁打开。现在需要你计算一下，电子锁上至少要有多少种特征，每个人的磁卡上至少有几个特征。如果特征的编号以小写英文字符表示，将每个人的磁卡的特征编号打印出来。要求输出的电子锁的总特征数最少。

为了使问题简单，$M$ 与 $N$ 的上下限为

$$3 \leqslant M \leqslant 7, \qquad 1 \leqslant N \leqslant 4$$

$M$ 与 $N$ 由键盘输入，工作人员的编号用 1#，2#，…等。

例如 $M=3,N=2$，则电子锁上要有三种特征，个人的磁卡上要有两种特征。

算法分析：

首先，我们从组合数学的角度分析该题。

题意告诉我们“至少要有 $N$ 个人同时使用各自磁卡才能将锁打开”。换言之，任意 $N-1$ 个人在一起，至少缺少一种开锁的密码特征，故不能打开电子锁，剩下的 $M-(N-1)$ 个人中的任意一个人到场，就一定能将锁打开。由 $M$ 个人中的 $M-N+1$ 的组合数为 $C(m,m-n+1)$，故电子锁至少应有 $C(m,m-n+1)$ 种特征。这样使得 $N-1$ 人在场时至

少缺少一个特征而打不开锁。对于任何一个工作人员来说，其余 $M-1$ 个人中任意 $N-1$ 个人在场，至少缺少一个这个工作人员磁卡上具有的特征而无法打开锁。所以每个人至少要有 $C(m-1,n-1)$ 种特征，如表 3-1 所示。

表 3-1

| 工作人员人数 $M$ | 打开锁所需的在场人数 $N$ | 电子锁的最少总特征数 $P$ $P=C(M,M-N+1)$ | 每个磁卡的最少特征数 $E$ $E=C(M-1,N-1)$ |
|---|---|---|---|
| 3 | 1 | 1 | 1 |
| 4 | 2 | 4 | 3 |
| 5 | 4 | 10 | 4 |
| 7 | 4 | 35 | 20 |

虽然通过组合数学知识是能够求出电子锁的最少总特征数和每人磁卡的最少特征数。但问题是，题目不仅要求计数，而且还要求枚举，即要求用 $C(M-1,N-1)$ 个小写英文字母表示出电子锁的所有特征，并输出 $M$ 张磁卡，每张磁卡具有电子锁的 $C(M-1,N-1)$ 个特征，这些特征也要用相应的小写英文字母表示出来。那么，如何枚举呢？

首先，我们用 #1，#2，…，#$M$ 表示 $M$ 个工作人员的编号，用 $C(M,M-N+1)$ 个小写英文字母 $a,b,\cdots$ 表示电子锁的所有特征编号。若电子锁的特征数超过 26 个，则用首字母为小写英文字母，第 2 字母为 $a$ 的双写英文字母 $aa,ba,ca,\cdots$ 表示。

然后我们枚举出 $M$ 中取 $M-N+1$ 的所有组合，这样的组合数一定有 $C(M,M-N+1)$ 个。例如 $M=4,N=2$，从 4 个工作人员中取 3 人的组合形式为[#1，#2，#3]，[#1，#2，#4]，[#1，#3，#4]，[#2，#3，#4]。电子锁的最少特征数为 $C(4,1)=4$，我们分别编号为 $a,b,c,d$。

下面，我们在表 3-2 至表 3-6 中依次往这 4 个组合中的每个工作人员的磁卡加入 $a$，$b,c,d$ 四种特征：

表 3-2

| 特征 / 编号 | $a$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|
| #1 | | | | |
| #2 | | | | |
| #3 | | | | |
| #4 | | | | |

初始时，每人的磁卡为空。

⇒

表 3-3

| 特征 / 编号 | $a$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|
| #1 | $a$ | | | |
| #2 | $a$ | | | |
| #3 | $a$ | | | |
| #4 | | | | |

往第一种组合的 #1，#2，#3 的磁卡，加入电子锁的 $a$ 特征。

表 3-4

| 编号＼特征 | a | b | c | d |
|---|---|---|---|---|
| #1 | a | b | | |
| #2 | a | b | | |
| #3 | a | | | |
| #4 | | b | | |

往第二种组合的#1,#2,#4的磁卡,加入电子锁的 b 特征。

⇒

表 3-5

| 编号＼特征 | a | b | c | d |
|---|---|---|---|---|
| #1 | a | b | c | |
| #2 | a | b | | |
| #3 | a | | c | |
| #4 | | b | c | |

往第三种组合的#1,#3,#4的磁卡,加入电子锁的 c 特征。

表 3-6

| 编号＼特征 | a | b | c | d |
|---|---|---|---|---|
| #1 | a | b | c | |
| #2 | a | b | | d |
| #3 | a | | c | d |
| #4 | | b | c | d |

往第四种组合的#2,#3,#4的磁卡,加入电子锁的 d 特征。

虽然,由表 3-6 可以看出,每个人磁卡上的特征数为 $C(3,1)=3$ 个,各个人的磁卡特征不尽相同。

第一个工作人员的磁卡具有 a,b,c 特征;

第二个工作人员的磁卡具有 a,b,d 特征;

第三个工作人员的磁卡具有 a,c,d 特征;

第四个工作人员的磁卡具有 b,c,d 特征;

任两张磁卡组合起来,一定会具备电子锁的所有特征 a,b,c,d,满足两人同时使用各自磁卡就能开锁的要求。

我们应用回溯法的算法框架来实现上述求解步骤。组合中当前位置的人员编号作为算符,磁卡的特征序列作为状态,每生成一个组合,当前组合的序号转换为特征字进入组合中每个工作人员的磁卡。所有组合生成后,每个工作人员磁卡上的 $C(m-1,n-1)$ 个特征字即为题解:

```
procedure make(l,last);
{l—组合的当前位置;last—组合 l 位置的工作人员的可能编号为#last+1…#m。
  组合中各位置的编号以及各组合 l 位置上的编号按递增顺序生成}
begin
  if l=m-n+2 then        {当前组合已生成}
    begin
      取电子锁的下一特征;
      往组合 1…(m-n+1)位置上的工作人员的磁卡加入该特征;
      退出 make 过程;
    end;
  for i:=last+1 to m do
```

```
    begin
      置当前组合第l位置的编号#i;
      递归调用 make(l+1,i),求在 m 人中取 l+1 人的组合;
    end;{for}
  end;{make}
```

例如:现在 $M=4,N=2$。我们通过调用 make(1,0)求每个人磁卡的特征,见图 3-6。具体递归过程如下,我们将组合的形成过程表征为结点,make 上方的数字表示递归顺序。

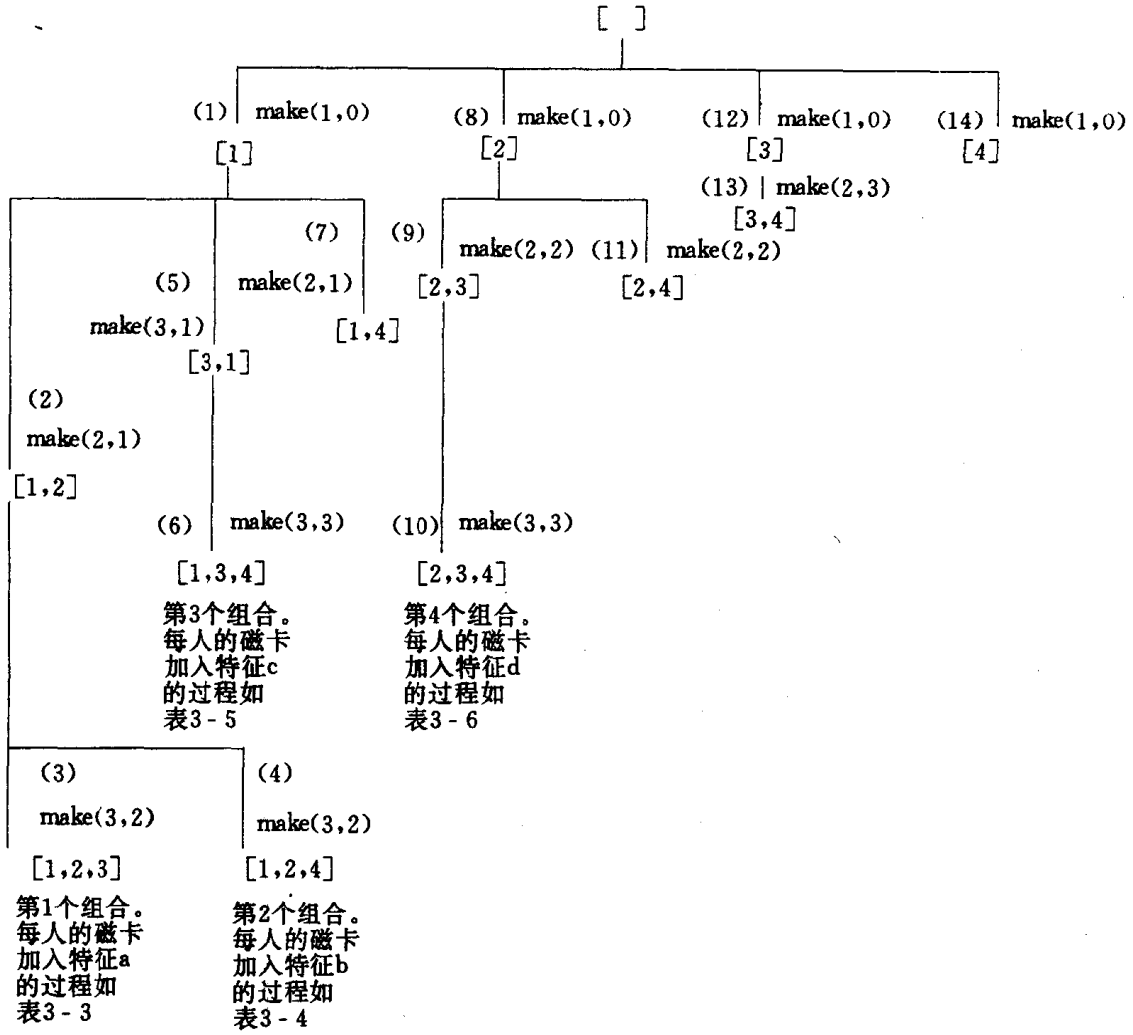



图 3-6

由此可见,上述递归定义是正确的。因为 $M$ 和 $N$ 是预先给定的,即 $M$ 中取的 $M-N+1$ 是一个确定值。每递归一次,当前组合的一个位置上的编号被确定。从根结点开始,递归 $M-N+1$ 次,则到达递归边界,求出一个组合,并往这个组合的工作人员的磁卡上加一个特征,然后在回溯过程中继续递归,求下一个组合。这样递归若干次后,一定会求出 $M$ 中取 $M-N+1$ 的所有组合,由最后到达的递归边界明确每个人磁卡的特征。

最后,我们给出门锁问题的程序题解。

```
const
```

```
  record:array [#1..#35] of string
         = ('a','b',…,'z','aa','ba','ca',…,'ia');
  {record[#i]——电子锁的第 i 种特征编码,(1≤i≤35)}
type
  listtype=array [1..7] of byte;{组合的数据类型}
  node     =string[35];          {特征序列的数据类型。node[j]字符一
                                  般取#1...#35(1≤j≤35)}
var
  stack       : listtype;  {当前组合,stack[i]—当前组合第 i 个位置
                            的工作人员的磁卡编号}
  way         : array [1..7] of node;
  {磁卡。way[i,j]编号为#i 的工作人员所持磁卡上的第 j 个特征字符,可通过 record[way[i,j]])
转换成相应的特征编号}
  min,    {每张磁卡的最少特征数}
  man,    {工作人员总数,要求输入}
  together,  {打开锁所需的最少在场人数,要求输入}
  total : integer;{电子锁的最少总特征数}

procedure init;
var i : integer;
 begin
   total := 0;  {电子锁的最少总特征数初始化为 0}
   for i:=1 to 7 do way[i]:=' ';{所有磁卡初始化为空}
   repeat          {读入工作人员总数 M}
     write('M =');readln(man);
   until (man>=1) and (man<=7);
   repeat       {读入打开锁所需的最少在场人数 N}
     write('N =');readln(together);
   until (together<=4) and (together>=1);
 end;{init}

procedure make(l: integer; last: integer);
 var i: integer;
 begin
   if l=man-together+2 {从 man 中取 man-together+1 的组合生成}
     then begin
            total:=total+1;{总特征数+1}
            for i:=1 to man-together+1
              do way[stack[i]]:=way[stack[i]]+chr(total);
     {总特征数作为特征字符加入当前组合的 man-together 个工作人员的磁卡}
            exit;{退出过程}
          end;{then}
    for i:=last+1 to man do {当前组合 l 位置的人员编号顺序取 last+1…man}
      begin
       stack[l]:=i;  {i 编号的工作人员放入当前组合的 l 位置}
       make(l+1, i);{求当前组合第 l+1 位置的人员编号,编号范围在 i+1…man 之间}
      end;{for}
   end;{make}

 procedure print;
  var i, j : integer;
```

```
begin
  输出电子锁的总特征数 total;
  min := Length(way[1]);
  输出每人磁卡的最少特征数 min;
  for i:=1 to man do
    begin
      write(i,'#');
      for j:=1 to min do 输出 i#磁卡上的第 j 个特征编号 record[way[i,j]];
      writeln;
    end; {for}
end; {print}

begin
  init;          {输入工作人员数和打开锁所需的最少在场人数}
  make(1,0);     {递归搜索每个工作人员磁卡上的特征}
  print;         {输出结果}
end. {main}
```

## 二、重组合

上面讨论的是不同元素中不允许重复选取的组合。更一般地，也有从不同元素中允许重复选取的组合问题。例如多项式展开的各项中，系数大多是重复的，整数的质因数分解式中，质因数也常出现重复的情况，这些属于不同元素中允许重复选取的组合问题。

从 $n$ 个不同元素中，取 $r$ 个允许重复的元素而不考虑其次序时，称为从 $n$ 个中取 $r$ 个允许重复的组合，简称重组合。允许重复的组合数记作 $H(n,r)$ 或 $C(n+r-1,r)$。

允许重复的组合的典型模型是把 $r$ 只相同颜色的球放到 $n$ 个编号不同的盒子中去，而且每个盒子的放球数不加限制，其方法种数是

$$C(n+r-1,r)=\frac{(n+r-1)!}{r!(n-1)!}$$

得出这个结果的一种简便方法是考察 $n+1$ 个 1 和 $r$ 个 0 的排列问题，并且每一个排列是由 1 开始和 1 结尾的。如果我们将 1 当作分划出放东西的盒子，而将 0 作为球，那末每一个这样的排列就对应于放 $r$ 只相同颜色的球到 $n$ 个编号不同的盒子中去的一种方法。例如设 $n=5,r=4$，则序列

$$10\boxed{1}100\boxed{1}10\boxed{1}$$

看作四只球放到五个盒子中去的一种方法。这种放法就是在第一个盒子中有一只球，第二个盒子中没有球，在第三个盒子中有两只球，第四个盒子没有球，在第五个盒子中有一个球。根据有限重复排列的计算公式，将 $n+1$ 个 1 和 $r$ 个 0 排列，使每个排列的两端都是 1 的方法总数是

$$\frac{P(n+1+r-2,n+1+r-2)}{r!(n+1-2)!}=\frac{(n+r-1)!}{r!(n-1)!}=C(n+r-1,r)$$

因为从 $n$ 个不同对象中选取 $r$ 个对象并允许重复选取的问题，可以看作用 $r$ 个相同的标记去标明这 $n$ 个不同的对象，而每一个对象可以被标上任意多个标记，因此在允许重复选取的情况下，从 $n$ 个不同对象中选取 $r$ 个对象的方法的数目是 $C(n+r-1,r)$。

[例 5] 试问$(X+Y+Z)^{1986}$有多少项?

**解**:$(X+Y+Z)^{1986}$的展开式每一项都是 1986 次方的,相当于 1986 个无区别的球放到三个编号不同的盒子去,每盒放入的球数不加限制。例如:

$X^{1986}$相当于 1986 个球都放入 $X$ 盒、$Y$ 和 $Z$ 盒空盒;

$X^{1984}YZ$ 相当于 1984 个球放入 $X$ 盒、$Y$ 和 $Z$ 盒各放一球;

所以 $R=1986, N=3$ 时,$C(1986+3-1,1986)=C(1988,1986)=1975078$(项)

现在,我们将重复组合的应用范围再拓宽一下。设不定方程为

$$x_1 + x_2 + \cdots + x_n = r$$

如果 $x_i$ 代表第 $i$ 个盒子所放的球的个数,$r$ 代表相同颜色的球数,那末这 $r$ 只球放入 $n$ 个编号不同的 $x_1, x_2 \cdots, x_n$ 盒子中,每盒放球只数不限的一种放法就是上述不定方程的一个解。因此所求的方法种数就是方程所有非负整数解的组数。这是因为对方程的一组固定的非负整数解 $x_1, x_2 \cdots, x_n$,

令

$$\begin{aligned} y_1 &= x_1 + 1 \\ y_2 &= x_1 + x_2 + 2 \\ &\cdots\cdots \\ y_n &= x_1 + x_2 + \cdots + x_n + n \end{aligned}$$

则有

$$1 \leqslant y_1 < y_2 \cdots < y_n = r + n$$

即 $y_1, y_2, \cdots, y_{n-1}$是从 $1, 2, \cdots, (r+n-1)$中取出的(按由小到大次序排列)$n-1$ 个数,因此方程解数便是从 $r+n-1$ 个数中取出 $n-1$ 个数的组合数,即

$$C(r+n-1, n-1) = C(r+n-1, r)$$

注:$C(p,q) = C(p, p-q)$

[例 6] $2x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8+x_9+x_{10}=3$ 的非负整数解共有多少组?

**解**:由于 $x_1, x_2, \cdots, x_{10}$的解为非负整数且 $x_1$ 的系数为 2,因此 $x_1$ 的值为 0 或 1。

当$x_1=0$ 时,原方程为 $x_2+x_3 \cdots +x_{10}=3$

非负整数解的组数是 $C(9+3-1,1)=C(11,3)$;

当$x_1=1$ 时,原方程为 $x_2+x_3+\cdots+x_{10}=1$

非负整数解的组数是 $C(9+1-1,1)=C(9,1)$

因此,原方程的非负整数解的组数是

$$C(11,3) + C(9,1) = 165 + 9 = 174$$

## 3.4 排列组合问题的一个实验程序

试题:

一个具有 $n$ 个不同元素的集合 $S=\{a_1, a_2, \cdots, a_n\}$,从 $S$ 中任选 $r(r \leqslant n)$个元素进行排列或组合,要求枚举出所有可能方案。

输入格式如下:

1. 计算机提示'$N,R=$',然后由操作者输入集合 $S$ 的元素个数及任取元素的个数;

2. 顺序输入集合 $S$ 的每个元素;

3. 计算机提示'Pailie?'。若操作者输入'$y$',则选择排列,若输入'$n$',则选择组合;

4. 计算机提示'Chongfu?'。若操作者输入'$n$',表明元素不允许重复。若输入'$y$',则表明元素允许重复。接着计算机提示'1——youxian,2——wuxian'。若操作者输入'1',则选定元素为有限次重复,然后依次输入每个元素的重复次数。若操作者输入'2',则选定元素为无限次重复(实际上允许每个元素可重复一次);

5. 计算机提示'Yuan pai lie?'。若输入'$y$',则选择圆排列,若输入'$n$',则选择线排列。

计算机经计算后显示所有排列(组合)的方案以及方案总数。

算法分析:

求满足指定条件(任选的元素的个数、排列还是组合、元素可重复与否)的方案数有多少,这是数学试题要求解答的问题,有明确的计数公式。而计算机解题的目标是枚举,即把各种方案完全列举出来,在枚举过程中累计方案总数。这里,计数公式仅不过是作验证结果是否正确的测试工具罢了。那么如何枚举排列组合方案呢?

首先我们必须明确,对任何元素形式的排列组合都可以归结为结元素序号 1 至 $n$ 的排列组合。

例如

$n=3, r=3, S$ 集合中的元素值为

$A[1]=$ '$A$'; $A[2]=$ '$B$'; $A[3]=$ '$C$'

Pailie? $y$

chongfu? $n$

即求 $a_1, a_2, a_3$ 的全排列方案。

设 $st$ 数组存储 $A$ 数组下标值的排列。

下标排列形式

st[1]=1; st[2]=2; st[3]=3

st[1]=1; st[2]=3; st[3]=2

st[1]=2; st[2]=1; st[3]=3

st[1]=2; st[2]=3; st[3]=1

st[1]=3; st[2]=1; st[3]=2

st[1]=3; st[2]=2; st[3]=1

排列后的下标对应 A 数组的元素值

A[st[1]]='A'; A[st[2]]='B'; A[st[3]]='C'

A[st[1]]='A'; A[st[2]]='C'; A[st[3]]='B'

A[st[1]]='B'; A[st[2]]='A'; A[st[3]]='C'

A[st[1]]='B'; A[st[2]]='C'; A[st[3]]='A'

A[st[1]]='C'; A[st[2]]='A'; A[st[3]]='B'

A[st[1]]='C'; A[st[2]]='B'; A[st[3]]='A'

显然,对 S 元素全排列的方案有如下 6 种形式:

'A', 'B', 'C'
'A', 'C', 'B'
'B', 'A', 'C'
'B', 'C', 'A'
'C', 'A', 'B'
'C', 'B', 'A'

至于允许元素重复的排列组合仅是指在每种方案中重复元素可使用有限次。无限次重复实际上也只不过重复 $r$ 次，即一种方案的所有元素都为该重复元素。在当前方案中每枚举重复元素一次，该元素的重复次数减 1，直至递减为 0，该元素不再参与挑选为止。下面我们就来讨论对最简形式 $S=\{1,2,\cdots,n\}$ 的排列组合问题。

我们采用回溯法搜索每个线排列(圆排列)或组合方案中各元素的值。在求线排列问题时，参与挑选的元素值为 $1,2,\cdots,N$；在求圆排列时，为了避免首尾元素相接后的重复排列，我们规定首元素为一个范围在 $1,2,\cdots,N-R+1$ 之间的指定值 $X$，其它 $N-1$ 个元素的选值范围为 $X+1,X+2,\cdots,N$，在这个约束条件下枚举圆排列方案。每规定一个首元素值回溯搜索一次。这样调用 $N-R+1$ 次回溯法，便可搜索出所有圆排列方案。而在求组合问题时，由于各元素次序不同时看作是同一种方案，因此对这种无序配置定义一个规定——对于 $a_1$，参与挑选的元素值为 1 至 $n$，而其它元素 $a_i(2\leqslant i\leqslant n)$，参与挑选的元素值为 $a_{i-1}$ 至 $n$，即每个方案的各元素值按递增顺序排列。无论是排列还是组合，挑选算符的顺序都由小至大。但是否所有该范围的算符都可无条件选中呢？不是的。若不允许元素重复时，已处理的元素中枚举过该整数值；或者允许元素重复时，已处理的元素中选用该整数值的次数递减为 0，则该算符显然是不适用的；应当舍去。因此，元素值的可取数非零是排列组合的又一个约束条件。

现在让我们先来观察一个简单的排列组合问题。

设 $N,R=3,3$

123

Pailie? $y$

Chongfu? $n$

即求 $S=(1,2,3)$ 的无重复元素的全排列。

初始状态显然是一个空集(　　)。

此时第一个排列方案开始枚举 1 作为第 1 个元素。为了说明这个事实，我们引进了 3 个结点，每个结点表征相应的状态信息，见图 3-7：

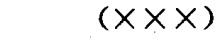



图　3-7

每个结点共有 3 个数据。第 $i(1\leqslant i\leqslant 3)$ 个数据指明当前方案中第 $i$ 个元素值。若该数据为 0，表明第 $i$ 个元素值尚未枚举出来。从空集出发，第 1 个元素值可以分别挑选 1,2,3，扩展出 3 个子结点。

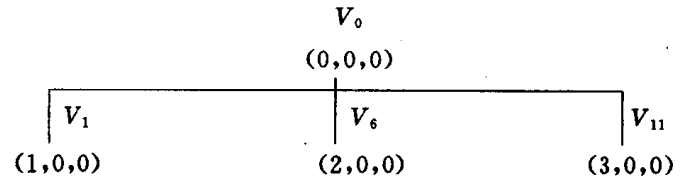



图 3-8

图 3-8 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 2 个元素的可能值，如此反复进行，一旦出现新结点的 3 个数据全非零，那就寻到了一种全排列方案。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 3-9 的图形。

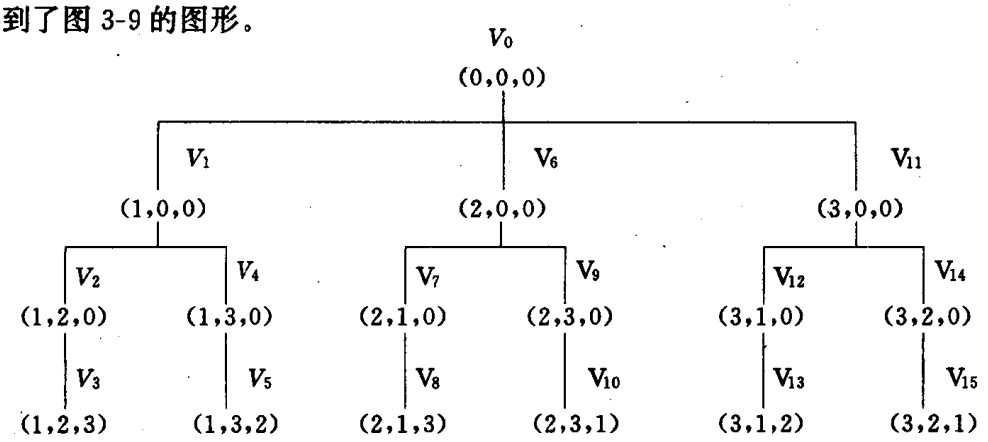



图 3-9

该图形像一棵倒悬的树。树中的每一个结点都是当前方案中满足约束条件的元素状态。其初始结点 $V_0$ 叫根结点，而最下端的结点 $V_3$，$V_5$，$V_8$，$V_{10}$，$V_{13}$，$V_{15}$称为叶结点，这些叶结点的 3 个数据全非零，亦即本题的目标结点。由根结点到每一个目标结点之间，揭示了一种全排列的形成过程。显然，1，2，3 的全排列存在由这六个叶结点表示的六种方案。

回溯搜索的过程如下：

初始状态(0，0，0)作为根结点 $V_0$，由此出发，第 1 个元素值置 1。从(1，0，0)出发枚举第 2 个元素值。选 1 会重复，因此选 2，产生状态(1，2，0)。但是第 3 个元素值再不能挑选已枚举过的 1，2，唯剩下选 3 了。至此，第一种全排列方案(1，2，3)产生。沿第一条路径回溯至 $V_2$，3 恢复为可选值。但从 $V_2$ 扩展只能重蹈旧方案，因此再往上回溯，选择 $V_1$作第 $i$ 个尚未用过的通向右边方向的分枝点，此时 3，2 相继恢复为可选值。但由于第 1 个方案已枚举 2 作为第 2 个元素值且第 2 个方案的第 1 个元素已枚举值 1，因此从 $V_1$往下扩展只能枚举 3，扩展出 $V_4$状态(1，3，0)。继续扩展，第 3 个元素当然是选择尚未枚举的唯一值 2 了，子结点 $V_5$的状态(1，3，2)表明第 2 个全排列方案产生。

按上述规律不断回溯检查，直至得出第六条路径 $V_0$—$V_{11}$—$V_{14}$—$V_{15}$。沿路径从 $V_{15}$回溯，由于 $V_{14}$，$V_{15}$无论怎样枚举当前元素值都会导致重复方案，因此回溯至 $V_0$。又因为 $V_0$ 已经相继枚举了值 1，2，3 作为第 1 个元素值而无法再扩展，至此，求出了所有的全排列方案。

```
Program Pall;
```

```
Uses crt;
Const
  Maxn                    = 100;
{n的最大值}
Type
  ttype                   = array [1..maxn] of integer;
{list 的类形说明}
Var
  list,stack              : ttype;
{list[i]：值 i 的可取数；stack[i]：方案中第 i 个元素的值}
  p,d,c                   : boolean;
{  true 排列      true 重复      true 圆排列  }
{p                ; d             ; c          }
{  false 组合     false 不重复   false 非圆排列}
  n,r,tot,x               : integer;
{n 个元素中取 r 个元素的排列或组合；tot——方案总数}
  a                       : array [1..maxn] of char;
{待排列或组合的集合元素}
Procedure read_boolean(var b:boolean);
{输入'Y'则返回 true，输入'N'则返回 false}
Var
  ch:char;
begin
  repeat
    ch:=upcase(readkey);              {输入一个字符}
  until (ch='Y') or (ch='N');
  writeln(ch);
  if ch='Y' then b:=true              {如果字符是'Y',则返回 true}
            else b:=false;            {否则返回 false}
end; { read_boolean}
procedure init;
var
  i:integer;
  ch:char;
begin
  write('N R =');readln(n,r);         {输入 N 和 R}
  for i:=1 to n do read(A[i]);        {输入待排列或组合的集合元素}
  write('Pailie ?'); read_boolean(p);  {输入是排列还是组合}
  write('Chongfu ?'); read_boolean(d); {输入是否允许重复}
  if d=true {如果允许重复,则询问是有限重复还是无限重复}
     then begin
          write('1——Youxian, 2——Wuxian '); readln(i);
          case i of
            1:for i:=1 to n do
              {若有限次重复，则依次输入每个无素的重复次数}
                begin
```

```
                write('T[i] = '); readln(list[i])
              end;
          2:{若无限次重复,则设每个元素可重复 r 次}
            for i:=1 to n do list[i] := r
      end
    end
  else {如果不允许重复,则设每个元素可重复 1 次}
       for i:=1 to n do list[i] := 1;
  write('Yuan Pai Lei ?');     {输入是否圆排列信息}
  read_boolean(c);
  tot:= 0                          {总数初始化为 0}
end;

procedure print;                   {打印一个方案}
var
  i :integer;
begin
  inc(tot);write(tot:5, ') ');
  for i:=1 to r do write(a[stack[i]]);
  writeln
end;

procedure make(s:integer);
  {递归搜索当前方案的第 s 个元素值}
var
  i,a : integer;
begin
  if s= r+1       {当已经取出 r 个元素, 则输入出一个方案}
     then print
     else begin
            if c = true {若是圆排列,则 x+1..n 为 s 位置的待选元素}
               then a:=x+1
               else if p=true then a:=1
                    {如果是排列,则 1..n 为 s 位置的待选元素}
                       else if s=1 then a:=1
                                   else a:=stack[s-1];
                    { 如果是组合,则          ┌1..n          s=1}
                    {   第 s 个元素的值范围=┤                  }
                    {                       └stack[s-1]..n s≠1}
               for i:=a to n do
                 if list[i] <>0          {值 i 的可取数非零}
                    then begin
                           dec(list[i]);   {值 i 的可取数减 1}
                           stack[s]:=i;    {方案中第 s 个元素的值为 i}
                           make(s+1);      {递归搜索第 s+1 个元素的值}
                           inc(list[i])    {恢复值 i 的可取数}
                         end
          end
end;
```

```
begin
  init;        {初始化}
  if not c
     then make(1)     {从第1个元素开始搜索所有方案}
     else begin
            for x:=1 to n do list[x]:=1; {设所在元素的可选数为1}
            for x:=1 to n-r+1 do
              begin
                {设排列的首元素为x, 从第2个元素位置开始搜索当前圆排列}
                {位置2..位置n的待选元素范围为x+1..n}
                stack[1]:=x;
                make(2)
              end;
          end;
  writeln('TOT = ',tot); readln {输入出方案数}
end.
```

## 习 题 三

1. 试枚举出满足下列条件的所有 $N$ 位十进制正整数：

(1) 各位数字不尽相同；

(2) 不允许出现数集[0,1,…,9]中的 $m$ 个数字。

2. 有 $N$ 位先生和 $N$ 位女士围圆桌而坐，如果要求男女交替安排座位，试问有多少种可能的座位？

又若上述 $2N$ 个人围圆桌而坐，如果其中有 $M$ 对人不愿坐在相邻位子上，有多少种不同的坐法。

3. 从{1,2,…,$n$}中选出3个数组成一个三重组$(x,y,z)$，使得 $z>x$ 且 $z>y$。

(1) 若 $z=k+1$ $(0\leqslant k\leqslant n-1)$，则这样的三重组有多少个？

(2) 若这样的三重组分为三种类型 $x=y$，$x<y$，$x>y$，则每种类型的三重组各有多少？

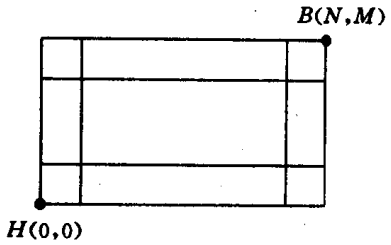
4. 在一个 $N\times N$ 的国际象棋盘上放着 $N$ 个车，这 $N$ 个车分为 $M$ 个组$(1\leqslant M\leqslant N)$，每组注以同一标记。试问有多少种方法放置这 $N$ 个车，使得没有一个车可以攻击另一个车(若同组内的两个车处在同一行或同一列便可互相攻击)。

5. 一位秘书在某大厦工作(题图3-1)。该大厦($B$ 点)在她家($H$ 点)东边 $N$ 个街段，北边 $M$ 个街段(图中的线条表示街道)，假定她每天从家里到大厦去上班都走某条递增的路线(即只能向东或向北走)。问她可以走多少种不同的路径。又若图中若干条街段上积满了水使她无法通过，这时她走哪条路线最短？

6. 从 $S=\{1,2,\cdots,n\}$ 中选取 $K$ 个数，使之没有两个数相邻，求不同的选取数。

7. 输入三角形周长 $N$，输出所有边长为整数的三角形的三边长。

8. 现有 $N$ 种不同颜色的珠子，每种个数不限，假定一个项链至少要有 $k(0\leqslant k\leqslant N)$ 种不同颜色的珠子做成，且不同颜色的珠子交替出现，那么用这 $N$ 种不同颜色的珠子可




题图 3-1

穿制多少种由 $P$ 颗珠子组成的项链(其中 $P$ 是一个质数)?

# 第四章　容斥原理

我们曾在第三章 3.1 节中讲到，如果完成一件事情有 $n$ 类进行方式 $A_1, A_2, \cdots, A_n$，每一类进行方式 $A_i$ 中有 $M_i$ 种方法($1 \leqslant i \leqslant n$)。当各类进行方式中的方法有重迭现象时，完成这件事的方法计数就不能简单采用加法原理，而是需要更深一层的原理——容斥原理来计数。

## 4.1　容斥原理的两种形式

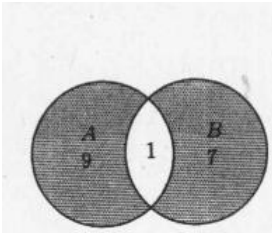
什么是容斥原理呢？我们先从下面二个简单问题谈起：

[**例 1**]　某班有 9 个学生有哥哥，7 个学生有姐姐，有哥哥又有姐姐的学生只有 1 个，那么全班有哥哥或有姐姐的学生共有多少个？

**解**：设集合 $A=\{a \mid a \in$ 某班，$a$ 有哥哥$\}$

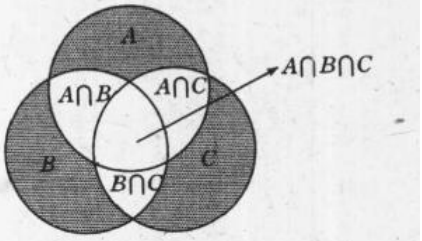
$$B=\{b \mid b \in \text{某班}, b \text{ 有姐姐}\}$$

由题设可得 $|A|=9$，$|B|=7$。又由交集合和并集合的定义可知，$A \cap B$ 恰好表示有哥哥又有姐姐的学生的集合，$A \cup B$ 恰好表示有哥哥或有姐姐的学生的集合。显然题设 $|A \cap B|=1$，要求解出并集 $A \cup B$ 中的元素个数 $|A \cup B|$。




(阴影部分是 $A \cup B$)

图　4-1




(阴影部分是 $A \cup B \cup C$)

图　4-2

由图 4-1 可以得到求两个集合的并集的元素个数的公式：

$$|A \cup B| = |A| + |B| - |A \cap B|$$

属于集合 $A$ 或属于集合 $B$ 的元素个数等于集合 $A$ 与集合 $B$ 的元素个数的和，减去同时属于这两个集合的元素个数。这里之所以要减去 $|A \cap B|$，原因是 $|A|$ 与 $|B|$ 中都计算了 $|A \cap B|$ 的数字，这样重复算了一次，当然必须减去一次。于是得出：

$|A \cup B|=9+7-1=15$，即某班有哥哥或有姐姐的学生共有 15 个。

从例 1 的求解过程中可以看到 $|A \cup B|$ 的计数公式是关键，那么对于三个有限集合 $A, B, C$ 的情形，是否也有相应的公式呢？答案是肯定的，这就是

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |C \cap A|$$

$$+|A\cap B\cap C|$$

即属于 $A$ 或属于 $B$ 或属于 $C$ 的元素个数等于这三个集合的元素个数的和，减去同时属于集合 $A$ 和集合 $B$ 的元素个数，减去同时属于集合 $B$ 和集合 $C$ 的元素个数，减去同时属于集合 $C$ 和集合 $A$ 的元素个数，再加上同时属于 $A,B,C$ 三个集合的元素个数。

这个公式可由图 4-2 上直接得到说明。要计算 $|A\cup B\cup C|$，需要先计算 $|A|+|B|+|C|$，其中已把 $|A\cap B|$，$|B\cap C|$，$|A\cap C|$ 都重复计算了一次，因此必须减去，于是得到 $|A|+|B|+|C|-|A\cap B|-|B\cap C|-|A\cap C|$，这样一来，多减了一次 $|A\cap B\cap C|$，所以必须加上，最终得到上述公式。

[**例 2**] 一所学校只设数学、物理、化学三门课程。已知修这三门课的学生分别是 150，120 和 100 人。同时修数学、物理两门课的学生 21 人；同时修数学、化学的 16 人；同时修物理、化学的 8 人；同时修三门的学生 5 人。问这所学校共有多少学生？

**解**： 设 $M$ ——修数学课的学生的集合；

$P$-修物理课的学生的集合；

$C$-修化学课的学生的集合。

由题设得 $|M|=150,|P|=120,|C|=100,|M\cap P|=21,|M\cap C|=16,|P\cap C|=8,|M\cap P\cap C|=5$。

$$|M\cup P\cup C|=|M|+|P|+|C|-|M\cap P|-|M\cap C|-|P\cap C|+|M\cap P\cap C|=330$$

即学校的学生数为 330 人。

利用求 $|A\cup B|$ 和 $|A\cup B\cup C|$ 的公式解决了例 1、例 2 的问题，那么这两个公式能否推广到四个甚至更多的集合上去呢？答案是肯定的。即：

令 $S_i(i=1,2,\cdots,m)\subseteq S$，且 $S_i$ 是 $S$ 中具有性质 $P_i$ 的元素所组成的子集合，则 $\bigcap_{i=1}^{m}S_i$ 是 $S$ 中同时具有性质 $P_1,P_2,\cdots,P_m$ 的元素子集合；$\bigcap_{i=1}^{m}\overline{S}_i$ 是 $S$ 中既不具有性质 $P1$，又不具有性质 $P_2,\cdots$，更不具有性质 $P_m$ 的元素子集合。

于是我们引出下面两种形式的容斥原理：

容斥原理

$S$ 中不具有性质 $P_1,P_2,\cdots,P_m$ 的元素个数为

$$|\overline{S}_1\cap\overline{S}_2\cdots\cap\overline{S}_m|=|S|-\sum S_i+\sum|S_i\cap S_j|-\sum|S_i\cap S_j\cap S_k|+(-1)^m|S_1\cap S_2\cdots\cap S_m|$$

式中第一个和式取遍集合 $\{S_i|,i=1,2,\cdots,n\}$，共进行 $C(n,1)=n$ 次和运算；

第二个和式取遍集合 $\{(S_i,S_j)|i,j=1,2,\cdots,n,\ i\neq j\}$，共进行 $C(n,2)$ 次和运算；

第三个和式取遍集合 $\{(S_i,S_j,S_k)|i,j,k=1,2,\cdots,n,\ i\neq j\neq k\}$，共进行 $C(n,3)$ 次和运算；

……

第 $m$ 个和式取遍集合 $\{(S_{i1},S_{i2},\cdots,S_{im})|i_1,i_2,\cdots,i_m=1,2,\cdots,n,i_1\neq\cdots\neq i_m\}$，共进行 $C(n,m)$ 次和运算；

证明：

等式左端是计算 $S$ 中不具有 $M$ 个性质 $P_1,P_2,\cdots,P_m$ 中任何一个性质的元素的个数。要证明等式成立，无非要证明等式右端中的两种情形成立：

1. 不具有 $M$ 个性质 $P_1,P_2,\cdots,P_m$ 中任何一个性质的一个元素被计算的次数净值为 1；

2. 至少具有这 $M$ 个性质 $P_1,P_2,\cdots,P_m$ 中之一的元素被计算的次数的净值为 0。

证情形 1 成立：

设 $S$ 中不具有 $M$ 个性质 $P_1,P_2,\cdots,P_m$ 中任何一个性质的元素 $X$，它在 $S$ 中，但不在 $S_i(i=1,2,\cdots,m)$ 中，于是在等式右边计算的次数的净值为 $1-0+0-0+\cdots+(-1)^m0=1$

证情形 2 成立：

设 $S$ 中恰好具有这 $M$ 个性质中 $k$ 个性质$(1\leqslant k\leqslant m)$的一个元素 $y$。由于它在 $S$ 中，故它在 $S$ 中被计算的次数为 $C(k,0)=1$；又由于 $y$ 恰好具有 $k$ 个性质，所以它是集合 $S_1,S_2,\cdots,S_m$ 中的 $k$ 个集合的元素，因而它在 $\sum|S_i|$ 中被计算的次数是 $C(k,1)=k$；又因为在 $k$ 个性质中取出一对性质的方法有 $C(k,2)$ 个，故 $y$ 是 $C(k,2)$ 个集合 $S_i\cap S_j$ 中的一个元素，所以它在 $\sum|S_i\cap S_j|$ 中被计算的次数是 $C(k,2)$；同理，它在 $\sum|S_i\cap S_j\cap S_k|$ 中被计算的次数是 $C(k,3)\cdots$，因此 $y$ 在等式右边被计算的次数的净值为：

$$C(k,0)-C(k,1)+C(k,2)\cdots+(-1)^mC(k,m)$$

由于 $m>k$ 时 $C(k,m)=0$，因此

$$\begin{aligned}&C(k,0)-C(k,1)+C(k,2)-\cdots+(-1)^mC(k,m)\\&=C(k,0)-C(k,1)+C(k,2)-\cdots+(-1)^kC(k,k)\\&=0\end{aligned}$$

（由二项式定理 $(1+x)^k=\sum_{i=0}^{k}C(k,i)x^i$ 中，令 $x=-1$ 时成立。）

等式右端中的两种情形成立，故容斥原理一可证。

容斥原理二

在集合 $S$ 中至少具有 $P_1,P_2,\cdots,P_m$ 中的一个性质的元素个数是

$$|S_1\cup S_2\cup S_3\cdots\cup S_m|=\sum|S_i|-\sum|S_i\cap S_j|+\cdots+(-1)^{m+1}|S_1\cap S_2\cap\cdots\cap S_m|$$

证：由于集合 $S_1\cup S_2\cdots\cup S_m$ 是 $S$ 中至少具有 $M$ 个性质 $P_1,P_2,\cdots,P_m$ 中的一个性质的元素所组成的子集合，所以有

$$|S_1\cup S_2\cup\cdots\cup S_m|=|S|-|\overline{S}_1\cap\overline{S}_2\cap\cdots\cap\overline{S}_m|$$

将容斥原理一代入上式，即可证明容斥原理二。证毕。

[例 3] 分别计算不超过 120 的合数和素数的个数。

解：因 $11^2=121>120$，所以不超过 120 的合数必定是 2,3,5,7 的倍数。

设 $S_i$ 为不超过 120 的数 $i$ 的倍数集$(i=2,3,5,7)$

$$|S_2|=\left[\frac{120}{2}\right]=60,\quad |S_3|=\left[\frac{120}{3}\right]=40,$$

$$|S_5| = \left[\frac{120}{5}\right] = 24, \quad |S_7| = \left[\frac{120}{7}\right] = 17$$

$S_2$、$S_3$、$S_5$、$S_7$ 的元素个数的总和为

$$|S_2| + |S_3| + |S_5| + |S_7| = 141$$

从 $S_2$、$S_3$、$S_5$、$S_7$ 中任取 2 个集合，进行 $C(4,2)=6$ 次交运算，累计其结果，使得同属于这四个集合中任两个集合的元素个数的总和为

$$\begin{aligned}&|S_2 \cap S_3| + |S_2 \cap S_5| + |S_2 \cap S_7| \\ &+ |S_3 \cap S_5| + |S_3 \cap S_7| + |S_5 \cap S_7| \\ &= \left[\frac{120}{2\times 3}\right] + \left[\frac{120}{2\times 5}\right] + \left[\frac{120}{2\times 7}\right] + \left[\frac{120}{3\times 5}\right] + \left[\frac{120}{3\times 7}\right] + \left[\frac{120}{5\times 7}\right] \\ &= 20 + 12 + 8 + 8 + 5 + 3 = 56\end{aligned}$$

从 $S_2, S_3, S_5, S_7$ 中任取 3 个集合，进行 $C(4,3)=4$ 次交运算，累计其结果，使得同属于这四个集合中任三个集合的元素个数的总和为

$$\begin{aligned}&|S_2 \cap S_3 \cap S_5| + |S_2 \cap S_3 \cap S_7| + |S_2 \cap S_5 \cap S_7| + |S_3 \cap S_5 \cap S_7| \\ &= \left[\frac{120}{2\times 3\times 5}\right] + \left[\frac{120}{2\times 3\times 7}\right] + \left[\frac{120}{2\times 5\times 7}\right] + \left[\frac{120}{3\times 5\times 7}\right] \\ &= 4 + 2 + 1 + 1 = 8\end{aligned}$$

同属于 $S_2, S_3, S_5, S_7$ 的元素个数的总和为 $|S_2 \cap S_3 \cap S_5 \cap S_7| = \left[\frac{120}{2\times 3\times 5\times 7}\right] = \left[\frac{120}{210}\right] = 0$

由容斥原理二得

$$|S_2 \cup S_3 \cup S_5 \cup S_7| = 141 - 56 + 8 - 0 = 93$$

这就是说，在不超过 120 正整数中或 2 的倍数、或 3 的倍数、或 5 的倍数、或 7 的倍数的个数共有 93。但是必须注意 93 个数中所含的 2,3,5,7 不是合数，必须舍去。所以不超过 120 的合数的个数应是 93−4=89 个。于是不超过 120 的非合数个数是 120−89=31。除去这 31 个数中所含的非素数 1，所以不超过 120 的素数个数是 30 个。

[例 4] $A,B,C,D,E,F$ 六个字母的全排列不允许出现 '$ACE$' 和 '$DF$' 字串的排列数

解：设 $i=\{A,B,\cdots,F$ 的全排列$\}$ $\quad |i|=6!$；

$A_1=\{ACE$ 作为一个元素出现的排列$\}$ $\quad |A1|=4!$

$A_2=\{DF$ 作为一个元素出现的排列$\}$ $\quad |A2|=5!$

显然 $A_1 \cap A_2$ 为同时出现 '$ACE$' 和 '$DF$' 的排列数 $|A_1 \cap A_2|=3!$

根据容斥原理一，不允许出现 '$ACE$' 和 '$DF$' 的排列数为

$|\overline{A}_1 \cap \overline{A}_2| = 6! - (4! + 5!) + 3! = 582$

## 4.2 容斥原理的一般形式

容斥原理一得出了从集合全体 $I$ 中同时不属于各子集 $S_1, S_2, \cdots, S_n$ 的元素个数的总和。这一节，我们将容斥原理推广到一般场合，即给出一个整数 $k(0 \leqslant k \leqslant n)$，要求得出 $i$ 中

恰好属于 $k$ 个子集的元素个数。例如 4.1 节的例 2 中，单修一门英语的学生有多少？只修一门课程的学生又有多少？只修两门课程的学生又有多少？

显然：

单修一门数学的学生数目应有 $|M\cap\overline{P}\cap\overline{C}|$

修一门课程的学生数目应有 $|M\cap\overline{P}\cap\overline{C}|+|\overline{M}\cap P\cap\overline{C}|+|\overline{M}\cap\overline{P}\cap C|$

修二门课程的学生数目应有 $|M\cap P\cap\overline{C}|+|M\cap\overline{P}\cap C|+|\overline{M}\cap P\cap C|$

一般地，已知 $S_1,S_2,\cdots,S_n$ 是有限集 $I$ 的 $n$ 个子集，设

$P_k$ 为 $S_1,S_2,\cdots,S_n$ 中同属 $k$ 个子集的元素个数的总和，即

$$P_0=|I|,\qquad P_1=\sum_{1\leqslant i\leqslant n}|S_i|,$$

$$P_2=\sum_{i\neq j}|S_i\cap S_j|,\cdots,\qquad P_n=|S_1\cap S_2\cap\cdots\cap S_n|$$

$Q_k$ 为 $I$ 中恰好属于 $k$ 个子集的元素的个数，即

$$Q_0=|\overline{S_1}\cap\overline{S_2}\cap\cdots\cap\overline{S_n}|$$

$$Q_1=\sum_{1\leqslant i\leqslant n}|\overline{S}_1\cap\cdots\cap\overline{S}_{i-1}\cap\underline{S_i}\cap\overline{S}_{i+1}\cap\cdots\cap\overline{S}_n$$

$$Q_2=\sum_{1\leqslant i<j\leqslant n}|\overline{S_1}1\cap\cdots\cap\overline{S}_{i-1}\cap\underline{S_i}\cap\overline{S}_{i+1}\cdots\cap\overline{S}_{j-1}\cap\underline{S_j}\overline{S}_{j+1}\cap\cdots\cap\overline{S}_n|$$

$$\cdots\cdots\cdots\cdots$$

$$Q_{n-1}=\sum_{1\leqslant i\leqslant n}|\underline{S_1\cap S_2\cdots\cap S_{i-1}}\cap\overline{S}_i\cap\underline{S_{i+1}\cap\cdots\cap S_n}|$$

$$Q_n=|\underline{S_1\cap S_2\cdots\cap S_n}|$$

那么，这些 $Q_k(1\leqslant k\leqslant n)$ 与 $P_1,P_2,\cdots,P_n$ 之间有什么关系呢？

我们可由图 4-3 得出 $M\cap\overline{P}\cap\overline{C}$（阴影部分表示）的元素个数等于 $|M|-|M\cap P|-|M\cap C|+|M\cap P\cap C|$。

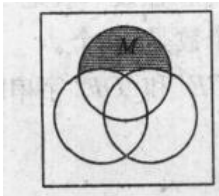



图 4-3

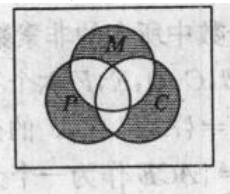



图 4-4

由图 4-4 不难得出：

$$\begin{aligned}
&|M\cap\overline{P}\cap\overline{C}|+|\overline{M}\cap P\cap\overline{C}|+|\overline{M}\cap\overline{P}\cap C|\\
=&|M|-|M\cap P|-|M\cap C|+|M\cap P\cap C|+|P|-|M\cap P|\\
&-|P\cap C|+|M\cap P\cap C|+|C|-|C\cap M|-|C\cap P|+|M\cap P\cap C|\\
=&|M|+|P|+|C|-2(|M\cap P|+|M\cap C|+|P\cap C|)+3|M\cap P\cap C|\\
=&P_1-2P_2+3P_3\\
=&Q_1
\end{aligned}$$

其值为图 4-4 中阴影部分的元素个数。

同理，从图 4-5 中可以得出：

$$
\begin{aligned}
&|M \cap P \cap \overline{C}| + |M \cap \overline{P} \cap C| + |\overline{M} \cap P \cap C| \\
&= |M \cap P| - |M \cap P \cap C| + |M \cap C| \\
&\quad - |M \cap P \cap C| + |P \cap C| - |M \cap P \cap C| \\
&= |M \cap P| + |M \cap \overline{C}| + |P \cap C| - 3|M \cap P \cap C| \\
&= P_2 - 3P_3 = Q_2
\end{aligned}
$$

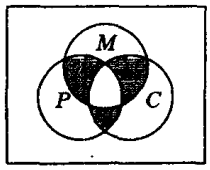



图 4-5

其值为图 4-5 中阴影部分的元素个数。

上题可以得出集合个数 $n=3, k=1,2$ 时的结果。如果我们将 $n$ 逐一增大，可以得出计算 $Q_k$ 的一般规律：

$$Q_k = P_k - C(k+1,1)P_{k+1} + C(k+2,2)P_{k+2} - \cdots \pm C(n,n-k)P_n$$

$$k=0,1,\cdots,n$$

显然 $k=0$ 时，上式公式就变成容斥原理形式一，可见它是容斥原理的推广。请读者自行推导 $Q_k$ 的计算公式。

**[例 1]** 某学校有 12 位教师，已知教数学课的教师有 8 位，教物理的教师有 6 位，教化学的教师有 5 位，其中 5 位既教数学又教物理，有 4 位兼教数学和化学，兼教物理和化学的有 3 位，有 3 位教师教三门课，试问教数、理、化以外的课的教师有几位，只教一门的教师有几位？正好教两门的教师有几位？

**解**：设

$I$—— 所有教师的集合　　$P_0 = |I| = 12$

$A_1$—— 教数学的教师集合　　$|A_1| = 8$

$A_2$—— 教物理的教师集合　　$|A_2| = 6$

$A_3$—— 教化学的教师集合　　$|A_3| = 5$

由题设：$|A_1 \cap A_2| = 5, |A_1 \cap A_3| = 4, |A_2 \cap A_3| = 3, |A_1 \cap A_2 \cap A_3| = 3$

$$P_1 = |A_1| + |A_2| + |A_3| = 8 + 5 + 6 = 19$$

$$P_2 = |A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3| = 5 + 4 + 3 = 12$$

$$P_3 = |A_1 \cap A_2 \cap A_3| = 3$$

教数、理、化以外课的教师有

$$
\begin{aligned}
Q_0 &= P_0 - P_1 + P_2 - P_3 \\
&= 12 - 19 + 12 - 3 \\
&= 2(\text{位})
\end{aligned}
$$

只教一门课的教师有

$$
\begin{aligned}
Q_1 &= P_1 - 2P_2 + 3P_3 \\
&= 19 - 2 \times 12 + 3 \times 3 \\
&= 4(\text{位})
\end{aligned}
$$

正好教二门的教师有

$$
\begin{aligned}
Q_2 &= P_2 - 3P_3 \\
&= 12 - 9 \\
&= 3(\text{位})
\end{aligned}
$$

## 4.3 容斥原理的应用

下面，我们通过几个有普遍意义的经典例题，看一看容斥原理是如何利用集合的基本运算来解决实际生活中的计数问题的，并作出程序解题的示范。

### 一、错排问题

$n$ 个有序的元素应有 $n!$ 种不同的排列。如若一个排列使得所有的元素都不在原来位置上，则称这个排列为错排。任给一个 $n$，求出 $1,2,\cdots,n$ 的错排个数 $D_n$ 共有多少个，并编程给出所有错排方案。

解：

设 $I=\{1,2,\cdots,n$ 的所有全排列$\}$　　显然 $|I|=n!$；

$I$ 的子集 $S_i=\{$数 $i$ 排在 $i$ 位置上的全排列$\}$。因数字 $i$ 不动，所以 $|S_i|=(n-1)!$ $(i=1,2,\cdots,n)$；

同理，$S_{i1}\cap S_{i2}\cap\cdots\cap S_{ik}$ 表示 $i_1,i_2,\cdots,i_k$ 位置上的全排列的集合，这就是说，在 $1,2,\cdots,n$ 中除了 $i_1,i_2,\cdots,i_k$ 这 $k$ 个数被固定外，其余 $n-k$ 个数可以任意排列，所以 $|S_{i1}\cap S_{i2}\cap\cdots\cap S_{ik}|$ 就相当于 $n-k$ 个数全排列的个数，即

$$|S_{i1}\cap S_{i2}\cap\cdots\cap S_{ik}|=(n-k)!$$
$$(1\leqslant i_1\leqslant i_2\cdots\leqslant i_k\leqslant n,\quad k=1,2,\cdots,n)$$

同时满足题设条件的排列个数 $D_n$ 应等于 $|\overline{S_1}\cap\overline{S_2}\cap\cdots\cap\overline{S_n}|$，由容斥原理一可得

$$\begin{aligned}D_n&=|I|-\sum_{1\leqslant i\leqslant n}|S_i|+\cdots+(-1)^k\sum_{i_1\neq\cdots\neq i_k}|S_{i1}\cap S_{i2}\cap\cdots\cap S_{ik}|+\cdots\\&\quad+(-1)^{n-1}|S_1\cap S_2\cdots\cap S_n|\\&=n!-C(n,1)(n-1)!+C(n,2)(n-2)!-\cdots\\&\quad+(-1)^kC(n-k)(n-k)!+\cdots+(-1)^{n-1}C(n,n)!\\&=n!\left(1-\frac{1}{1!}+\frac{1}{2!}+\cdots\cdots\pm\frac{1}{n!}\right)\end{aligned}$$

显然上述公式只能给出错排计数，无法枚举出错排方案，因此，我们不得不寻找另外一种产生错排 $D_n$ 的方法：

以 $1,2,\cdots,4$ 四个数的错排为例，分析其结构，找出规律性的东西来。

1,2 的错排唯一，即 2,1；

1,2,3 的错排为 3,1,2 和 2,3,1。这两者可视作是 1,2 错排，3 分别与 1,2 换位而得。见图 4-6。

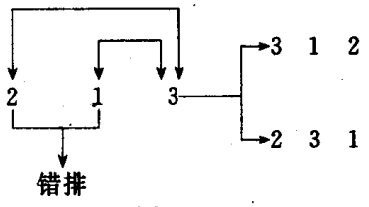



图 4-6

1 2 3 4 的错排有

| | |
|---|---|
| 4 3 2 1 | 4 1 2 3 |
| 3 4 1 2 | 3 4 2 1 |
| 2 1 4 3 | 3 1 4 2 |
| 4 分别与 1,2,3 换位，其余两个元素错排。 | 4 和 3,1,2(1,2,3 的一个错排)的每一个数换位。 |

4 3 1 2
2 4 1 3
2 3 4 1

4 和 2,3,1(1,2,3 的另一个错排)的每一个数换位。

由上面分析得出产生错排的第二种方法：

从 1,2,…,$n$ 中任取一数 $i$，数 $i$ 分别与其它的 $n-1$ 个数之一互换，其余 $n-2$ 个数进行错排，共得 $(n-1)D_{n-2}$ 个错排。另一部分为数 $i$ 以外的 $n-1$ 个数进行错排，然后 $i$ 与其中每个数互换得 $(n-1)D_{n-1}$ 个错排。然后由加法原理得出递归关系式：

$$D_n = (n-1)(D_{n-1} + D_{n-2}) \qquad D_1 = 0, \qquad D_2 = 1$$

我们使用这个递归公式编程，可使程序变得十分简练清晰。

我们将 1,2,…,$k$ 的错排方案存储在以 $P[k]$ 为首结点的单链表内，链结点数为 $D_k$，该链表是由 $P[k-2]$ 和 $P[k-1]$ 扩展生成的 $(2 \leqslant k \leqslant N)$。

由于 1,2 的错排方案仅有一个 2,1，因此初始时按照图 4-7 设

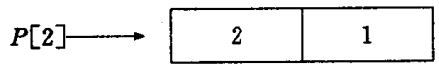



图 4-7

然后依次生成 $P[3],P[4],\cdots,P[k]$，生成的方法如下：

根据错排公式的递归定义，先求 $(k-1)D_{k-2}$ 个错排方案，即 $k$ 与 $P[k-2]$ 链上每个结点中的 1 互换，生成 $D_{k-2}$ 个 1,2,…,$k$ 的错排方案，然后 $k$ 与 $P[k-2]$ 链上每个结点中的 2 互换，生成 $D_{k-2}$ 个 1,2,…,$k$ 的错排方案，...直至 $k$ 与 $P[k-2]$ 链上每个结点中的 $k-1$ 互换，一共生成 $(k-1)D_{k-2}$ 个 1,2,…,$k$ 错排方案。

为了使互换元素的 $k-2$ 个元素保持错排，设当前与 $k$ 互换的元素为 $I(1 \leqslant I \leqslant k-1)$；$P[k-2]$ 当前链结点的第 1 至第 $I-1$ 个元素间，若元素值小于 $I$ 则不变，否则元素值加 1，第 $I$ 个元素置 $k$ 值；第 $I$ 至第 $k-2$ 个元素间，若元素值小于 $I$，则值不变并向后移 1 个位置，否则元素值加 1 后向后移 1 个位置；第 $k$ 个元素置 $I$ 值。这样便生成一个 1 至 $k$ 的错排方案，链入 $P[k]$。

例如，图 4-8 表示为：

接下来，根据错排公式的递归定义，再求 $(k-1)D_{k-1}$ 个错排方案，即对 $P[k-1]$ 链的每一个错排方案，$k$ 分别与 1,2,…,$k-1$ 互换位置。每次互换生成一个 1,2,…,$k$ 的错排方案，共可生成 $k-1$ 个错排，而 $P[k-1]$ 链共有 $D_{k-1}$ 个链结点，因此又有 $(k-1)D_{k-1}$ 个

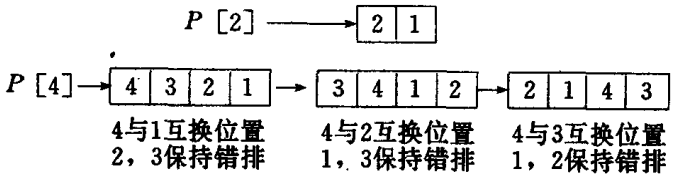
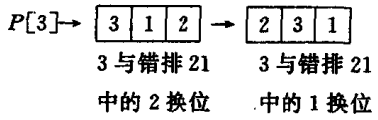



图 4-8

1,2,…,$k$ 错排方案,链入 $P[k]$。

例如,图 4-9 表示为:



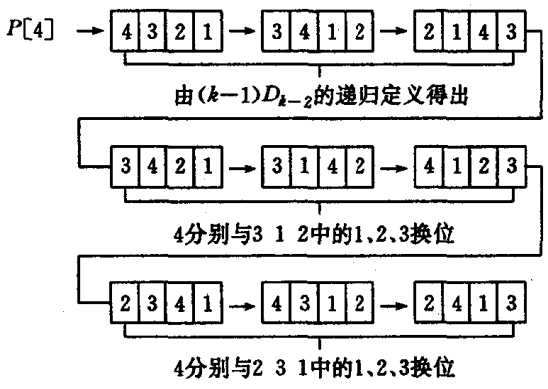




图 4-9

下面给出程序题解

```
program cou_pai;
  uses
    crt;{ 使用屏幕单元 }
  const
    maxn              = 30;
  type
    listtype          = array [1..maxn] of integer;  { 一种错排方案 }
    ptr               = ^ node;  { 所有错排方案 }
    node              = record
                          list : listtype;
                          next : ptr;
                        end;
  var
    pai_lie           : array [2.. maxn] of ptr;
    { pai_lie[i]为 i 个元素的所有错排方案 }
    n,tot             : Integer;
  procedure Init;
```

```
begin
  clrscr;
  repeat write ('n='); { 输入元素数 }
       readln ( n );
  until (n>1) and (n <= maxn);
  new(pai_lie[2]); { 确定 12 的错排 21 }
  pai_lie[2] ^ .list[1] := 2;
  pai_lie[2] ^ .list[2] := 1;
  pai_lie[2] ^ .next    := nil;
end;
procedure clear (p:ptr);  { 释放指针 }
  begin
    If p = nil then exit;
    clear(p ^ .next); { 递归释放 P 以后的方案 }
    dispose(p);
  end;
procedure make(lev :integer);  { 生成 Lev 个元素的错排方案 }
  var
    head,p,q  :ptr;
    i,j       :integer;
  begin
    new(head);
    p := head;
    if lev > 3 then
      for i := 1 to lev-1 do  { 先求(Lev-1)D_{Lev-2}个错排方案 }
        begin
          q := pai_lie[lev-2];
          while q <> nil do
            begin
              new(p ^ .next);
              p := p ^ .next;
              for j := 1 to i-1 do
                if q ^ .List[j] < i
                  then p ^ .list[j] := q ^ .list[j]
                  else p ^ .list[j] := q ^ .list[j] + 1;
              p ^ .list[i] := lev;
              for j := i+1 to lev-1 do
                if q ^ .list[j-1] < i
                  then p ^ .list[j] := q ^ .list[j-1]
                  else p ^ .list[j] := q ^ .list[j-1] + 1;
              p ^ .list[lev] := i;
              q:=q ^ .next;
            end;
        end;
    if lev>3
      then clear(pai_lie[lev-2]); { 再加上(Lev-1)D_{Lev-1}个错排方案 }
```

```
      q := pai_lie[lev-1];
      while q <> nil do
       begin
        for i := 1 to lev-1 do
          begin
            new(p^.next);
            p := p^.next;
            p^.next := nil;
            for j := 1 to i-1 do
              p^.list[j] := q^.list[j];
            p^.list[i] := lev;
            for j := i+1 to lev -1 do
              p^.list[j] := q^.list[j];
            p^.list[lev] := q^.list[i];
          end;
        q:=q^.next;
       end;
      pai_lie[lev] := head^.next;
    end;
procedure show (p:ptr);  { 显示方案 }
 var i     :integer;
 begin
   if p = nil
     then exit;
     inc (tot);       { 错排数+1 }
     write(tot,' :');
     for i :=1 to n do  { 显示当前错排 }
       write (p^.list[i]:3);
     writeln;
     show(p^.next); { 递归显示下一错排 }
   end;
procedure main ;
   var
     i          :integer;
   begin
     for i := 3 to n do { 顺序搜索 3 个元素、4 个元素、…、n 个元素错排的所有方案 }
       make (i);
       tot :=0;          { 方案数初始化 }
       show(Pai_lie[n]); { 显示 n 个元素错排的所有方案 }
   end;
begin
   init;  { 输入元素数，确定初始错排 }
   main;  { 计算和输出 n 个元素的错排方案 }
   writeln ('Total number of ways : ',tot); { 输出错排方案数 }
   readln;
end.
```

## 二、布棋问题

在 $M\times N$ 的方格中任意指定 $X$ 个格子构成一个棋盘，在任一个这样的棋盘上放置棋子，要求任意两个棋子不得位于同一行或同一列上。在棋盘上放置 $K$ 个棋子并满足上述要求的一种方法称为一个方案。

问题 1：

编程要求：

1. 对给定的一个棋盘，求出该棋盘可放置的最多的棋子数 $P$。

2. 记 $D_i$ 为该棋盘上放置 $i$ 个棋子时的方案总数($1\leqslant i\leqslant P$)，其中经旋转和镜面反射而得的方案记为不同的方案，对每一个 $i$，求出相应的 $D_i$。

3. 程序应能够连续处理多个棋盘，对每一个棋盘，输出 $P$ 和 $D_1,D_2,\cdots,D_p$，只需输出数字，不必输出具体的棋盘方案。在输出至屏幕的同时也将求得的结果存入文件中，每一个棋盘的所有结果存入一个文件，第一个棋盘的结果放在文件 1.txt 中，第二个棋盘的结果放在文件 2.txt 中，依次类推。

需要程序处理的所有棋盘由一个仅含有数字的文本文件提供，该文件的第一行是两个数字，代表第一个棋盘的 $M$ 和 $N$，其下 $N$ 行为一个仅由 0,1 组成的 $M\times N$ 的矩阵，某一位置值为 1 表示相应的格子在这个棋盘上，为 0 表示相应的格子不在棋盘上；再下面一行为第二个棋盘的 $M$ 和 $N$，其下又是一个矩阵，表示第二个棋盘的布局信息；以此类推，$M$ 和 $N$ 均为 0 时表示所有棋盘已处理完毕，程序可以结束。

在程序连续处理多个棋盘时，注意你的程序中不应该有任何等待输入的语句，以便统计程序处理全部棋盘所用的时间。所用总时间是评分的一个依据。

应注意棋盘是稀疏的，即 $X<M\times N/2$。

$1<M,N<10$。

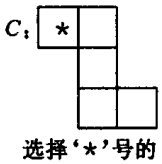
解：

算法分析：

根据试题给出的“任意两个棋子不得位于同一行或同一列”的置棋规则，对任意一个 $m\times n$ 的棋盘，可放置的最多棋子数 $p\leqslant\min(m,n)$。

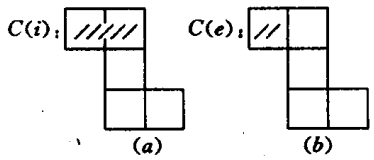
令 $I_k(C)$ 表示 $k$ 只棋子按上述规则布到棋盘 $C$ 的不同方案数。

在布棋过程中选择某一格，无非有两种可能：一是对该格布下了棋子；一是不布棋(见图 4-10)。因此棋盘 $C$ 可分为 $C(i)$ 和 $C(e)$ 两个部分。




选择‘*’号的格子布棋

图 4-10




注：虚线表示从原图中去掉的部分。

图 4-11

$C(i)$为棋盘 $C$ 的某一格子所在的行和列被排掉以后所余的部分(见图 4-11($a$))。$C(e)$是从 $C$ 中去掉该格后的棋盘(见图 4-11($b$))。

显然有 $I_k(C)=I_{k-1}(C(i))+I_k(C(e))$;

$I_0(C)=1$;

$I_{k-1}(C(i))$——表示对某格下了一个棋子后,剩下的 $k-1$ 棋子布到 $C(i)$棋盘上的方案数;

$I_k(C(e))$——表示对某格不布棋子,则 $k$ 个棋子布到 $C(e)$上的方案数。

上述公式是一个简明的递归式。我们可编出对应的递归程序,按上述规律逐一对每个空格试放棋子,以求得可放置的最多棋子数 $p$ 和放置 1,2,…,$p$ 棋子时的方案数 $D_1,D_2,\cdots,D_p$。

```
program qi_zi_duo_xiang_shi;
  uses
    crt;
  const
    maxn          = 10;     { 棋盘尺寸 }
  type
    boardtype     = array[1..maxn,1..maxn] of integer;  { 棋盘类型 }
    ltype         = array[0..maxn] of integer;  { 多项式类型 }
  var
    b             : boardtype;     { 棋盘 }
    l             : ltype;         { l[i](1≤i≤p)——放 i 个棋子的方案数
}
    num,m,n,s     : integer;       { num:数据组序号; m,n:棋盘长宽 }
    name1,name2   : string;        { 输入输出文件名 }
    f1,f2         : text;          { 输入输出文件变量 }
procedure init_b;  { 从输入文件中读入棋盘 }
  var i,j : integer;
    begin
      for i:=1 to n do
        for j:=1 to m do read(f1,b[i,j]);
    end;
procedure show;    { 显示方案数并存入文件 }
  var p,i : integer;
    begin
      p:=0;  { 求第 num 个棋盘可放置的最多棋子数 p }
      repeat inc(p);
      until (p>m) or (p>n) or (l[p]=0);
      dec(p); { 向 num.txt 文件和屏幕输出 p,D1..Dp }
      str(num,name2); assign(f2,name2+'.txt'); rewrite(f2);
      writeln(f2,p); writeln('Number ',num); writeln('P=',p);
      for i:=1 to p do writeln(f2,l[i]);
      for i:=1 to p do writeln('D',i,'=',l[i]);
      writeln;
      close(f2);
```

```
    end;
procedure make_l(lev:byte);     { 建立棋多项式 }
  var i,j,t : integer;
      ll : ltype;
    begin
      for i:=1 to n do
        for j:=1 to m do
          if b[i,j]=1 then
            begin
              { 第 lev 个棋子布入(i,j)后,排除 i 行和 j 列}
              for t:=1 to n do if b[t,j]=1 then b[t,j]:=-lev;
              for t:=1 to m do if b[i,t]=1 then b[i,t]:=-lev;
              make_l(lev+1); { 递归,对上述棋盘 C(i)布第 lev+1 个棋子 }
              ll:=l;
            { 恢复第 lev 个棋子所在行列的格子,该棋所在的(i,j)不准布棋 }
              for t:=1 to n do if b[t,j]=-lev then b[t,j]:=1;
              for t:=1 to m do if b[i,t]=-lev then b[i,t]:=1;
              b[i,j]:=-lev;
              make_l(lev+1);  { 递归,对上述棋盘 C(e)布第 lev+1 个棋子 }
              b[i,j]:=1;      { 恢复格子(i,j) }
              for t:=s downto 1 do l[t]:=l[t]+ll[t-1];
            { 求 Ik(C)=Ik(C(e))+Ik-1(C(i))}
          { 1≤k≤s }
              exit;
            end;
      fillchar(l,sizeof(l),0); l[0]:=1; { 递归边界:I0(C)=1 }
    end;
begin    { 主程序 }
  num:=0;  { 棋盘数初始化 }
  writeln('input file name='); readln(name1);
  assign(f1,name1); reset(f1);
  { 输入文件名串,并与文件变量 f1 连接起来;文件读准备 }
  repeat inc(num)            { 棋盘数加 1 }
         readln(f1,m,n);  { 读入棋盘规模 }
         if m<n then s:=m else s:=n; { 求行列数的小者 s }
         if m>0 then
           begin
             init_b;       { 输入 m*n 的 0.1 矩阵 }
             make_l(1);    { 求 p 和 D1..Dp }
             show;         { 输出结果 }
           end;
  until m=0;
  close(f1);
end.
```

## 三、有禁区的排列

下面我们将布棋问题再引申一步。在 $N\times N$ 的方格上设 $X$ 个格子构成一个禁区,禁

区范围外的方格可以布棋。若按任意两个棋子不得位于同一行或同一列的规则，问布 $n$ 个棋子无一落入禁区的方案数有多少？

上述问题在实际生活中有实用价值，例如有 $G,L,W,Y$ 四名工作人员，$A,B,C,D$ 为四项任务，但 $G$ 不能从事 $B$；$L$ 不能从事 $B,C$ 两项任务；$W$ 不能做 $C,D$ 工作，$Y$ 不能从事任务 $D$，若要求每人从事各自力所能及的一项工作，试问有多少种不同方案？

解：显然每一种分配方案相当于图 4-12 关于 $A,B,C,D$ 的有禁区的排列。

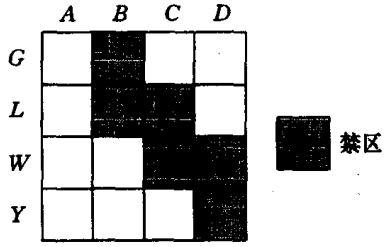



图 4-12

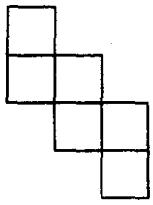


图 4-13

例如对于排列 $CDBA$，相当于 $G$ 从事任务 $C$，$L$ 从事任务 $D$，$W$ 从事 $B$，$Y$ 从事 $A$ 的任务安排。

问题相当于求图 4.8 所示的有禁区的棋盘用 4 个棋子进行布局的方案数。

设：$Y_i$——第 $i$ 个棋子布置到禁区部分的方案数；

$A_i$——第 $i$ 个棋子放入禁区的事件。

对于图 4-13 的棋盘，依据布棋方法，可求出 $Y_1=6,Y_2=10,Y_3=4$。

一个棋子落入禁区的方案数为 $Y_1$，剩下的 $n-1$ 个棋子可以任意排列，排列数为 $(n-1)!$；故至少有一个棋子进入禁区的方案数为 $\sum|A_i|=Y_1(n-1)!$。两个棋子落入禁区的方案数为 $Y_2$，而其余 $n-2$ 个棋子可以任意排列，排列数为 $(n-2)!$，故至少有两个格子进入禁区的方案数为 $\sum|A_i\cap A_j|=Y_2(n-2)!$……依此类推。

依据容斥原理一，布 $n$ 个棋子无一落入禁区的方案数应为

$$|\overline{A}_1\cap\overline{A}_2\cdots\cap\overline{A}_n|=n!-Y_1(n-1)!+Y_2(n-2)-\cdots+(-1)^nY_n$$

显然对于上例，所求的方案数为

$$\begin{aligned}n&=4!-6\times3!+10\times2!-4\\&=24-36+20-4\\&=4\end{aligned}$$

由有禁区排列问题的计算公式看出，它的程序可以在布棋问题程序的基础上稍作修改后得到。

但我们改用回溯法求解，以便从“一题多解，多向求解”的角度，帮助读者拓宽思路。

算法分析：

设 $N\times N$ 的矩阵中，元素 1 所在位置为禁区，元素 0 的位置可以任意布棋。由于任意两个棋子不得位于同一行或同一列，因此每个方案中充其量布 $N$ 枚棋子。

我们按由上至下，由左至右的顺序搜索每个布棋位置，若当前位置为禁区或与以前某

行中的棋子同位一列，则放弃该摆法，重新设定当前行的另一种摆法，若当前位置非禁区且该列上的各行未布棋，则布一枚棋子在该位置。……在搜索完 $N$ 行后，回溯至上一行，继续搜索下一排列。如此递归回溯，直至所有方案产生为止。

```
{ 有禁区的排列 }
program you_jin_qu_de_pai_lie;
  uses
    crt;
  const
    maxn            = 100;
  type
    ltype           = array[1..maxn] of integer;
    lltype          = array[1..maxn] of ltype;
  var
    n ,tot          : integer;    { 棋盘规模、方案数 }
    f               : text;       { 文件变量 }
                                  {                  (0 (i,j)可以布棋 }
    ll              : lltype;     { 棋盘。LL[i,j]= {                  }
                                  {                  (1 (i,j)为禁区   }
    l,w             : ltype;
{       (0  i列未布棋                                  }
{ L[i]={                    W[i]——i行棋子的列位置      }
{       (1  i列已布棋                                  }
  procedure init;
    var i,j : integer;
        str : string;
      begin
        clrscr;
        write('input file name : '); { 读入文件名串，并与文件变量连接 }
        readln(str);
        assign(f,str);
        reset(f);
        readln(f,n);
        for i:=1 to n do              { 读入棋盘 }
          for j:=1 to n do
            read(f,ll[i,j]);
        close(f);
      end;
  procedure search(lev:integer);
    var i : integer;
      begin
        if lev>n              { 若搜索完 n 行 }
          then begin
                 inc(tot);    { 则累计、输出方案数 }
                 write(tot:3,')  ');
                 for i:=1 to n do  { 输出各行棋子的列位置 }
                   write(w[i]:3);
```

```
            writeln;
          end
        else begin
          for i:=1 to n do { 搜索 1..n 列 }
          if (l[i]=0) and (ll[lev,i]=0) { i 列未布棋且当前行 i 列非禁区 }
          then begin
            l[i]:=1;              { 置 i 列布棋标志 }
            w[lev]:=i;            { 置当前行棋子的列位置 }
            search(lev+1);        { 递归搜索下一行 }
           l[i]:=0;               { 恢复 i 列未布棋标志 }
          end;
        end;
  end;
begin
  init;                           { 读入棋盘 }
  tot:=0;                         { 方案初始化 }
  fillchar(l,sizeof(l),0);        { 置所有列未布棋标志 }
  search(1);                      { 从第 1 行开始递归搜索所有排列 }
  writeln('tot = ',tot);          { 输出方案总数 }
end.
```

## 四、求最小棋盘

已知一个未知其规模的棋盘中有 $R$ 个格子，若任意两个格子处于同一行或同一列上，则称这两个格子为互不相容，否则称之为相容的。已知一个棋盘 $R$ 个格子两两是否相容或不相容的全部信息，编程求出一个相应的棋盘，并要求能包含该棋盘的矩形面积最小。

输入输出要求：

格子间是否相容的信息由一个文本文件提供，该文件第一行是一个数字，代表格子总数 $R$，其下 $R$ 行是一个仅由 0,1 组成的 $R\times R$ 的矩阵，若该矩阵第 $I$ 行第 $I$ 列为 1，表示格子 $I$ 与格子 $J$ 不相容，否则相容。

输出应显示求得的棋盘和包含该棋盘的最小矩形的长和宽。同时将所求得的棋盘存入一个名为 RESULT. TXT 的文件中，该文件共有 $R$ 行，其中第 $I(1\leqslant I\leqslant R)$ 行是第 $I$ 个格子的坐标，先是横坐标，后是纵坐标，其间以空格分隔。

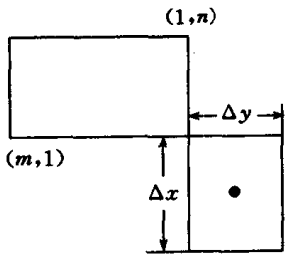



图 4-14

### 1. 算法分析

设 $m,n$——目前棋盘的行数和列数。初始时 $m,n=0$；

$\Delta x,\Delta y$——确定一个格子需扩充的行数和列数。显然最初应设 $\Delta x,\Delta y=1$。$\Delta x,\Delta y$ 组成一个独立棋盘，如图 4-14 所示。

根据 $R$ 个格子两两是否相容或不相容的全部信息，生成相应的最小棋盘的过程如

下：

```
m=0; n=0;
repeat
  Δx=1; Δy=1;
  if 存在一个未确定位置的格子 hh then
     begin
       格子 hh 放入 (m+1,n+1) 位置;
       repeat
         if 存在一个已确定位置但未检查的格子 h then
            begin
              置 h 格子检查标志;
              搜索所有与 h 不相容且未确定的格子 g:
            ① g 格与 h 格同行,则从 n+1 列开始逐列搜索:
              若 n..n+Δy 列中某列 n+i 的所有格子与 g 格互不相容,则 g 格放入 h 格所在行
              的 n+i 列;
              否则 Δy=Δy+1;g 格放入 h 格所在行的 n+Δy 列;
            ② g 格与 h 格同列,则从 m+1 行开始逐行搜索:
              若 m..m+Δx 行中某行 m+i 的所有格子与 g 格互不相容,则 g 格放入 h 格所在
              列的 m+i 行;
              否则 Δx=Δx+1;g 格放入 h 格所在列的 m+Δx 行;
            ③ 若 g 格与 h 格非同行且非同列,则失败退出;
          end; { then }
       until 目前已确定位置的格子都已检查;
       if Δx>Δy then Δx * Δy 组成的独立棋盘旋转 90°;
       m=m+Δx; n=n+Δy;
     end; { then }
until 所有格子的位置皆确定;
```

最后检查一下生成的棋盘。若同行同列的所有格子满足题意给出的相容或互不相容关系，则该棋盘一定被一个最小矩形面积 $m \times n$ 所含。否则无解。

例：有这样一个测试数据：

```
7
0 1 1 1 0 0 0
1 0 1 0 0 0 0
1 1 0 0 1 0 0
1 0 0 0 1 0 0
0 0 1 1 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 1 0
```

上述邻接矩阵对应一个不相容关系的无向图，如图 4-15 所示。

依据算法，格子 1 放入(1,1)位置，与格子 1 不相容的格子为{2,3,4}，下面逐一确定这些格子的位置：

格子 2 放入(1,2)。而格子 3 分别与第 1 行的格子 1,2 不相容，因此格子 3 放入(1,3)，而格子 4 不与格子 2，3 存在不相容关系，因此只能放入(2,1)，形成一个 2×3 的棋

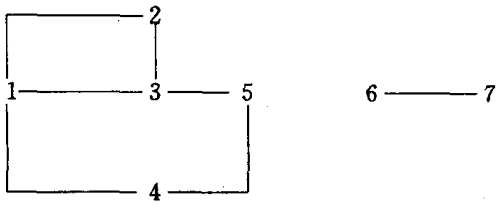



图 4-15

盘,如图 4-16 所示。

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | |
| 4 | | | |

图 4-16

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | |
| 4 | | 5 | |

图 4-17

置格子 1 检查标志。

然后检查已确定位置的格子 2。由于格子 2 不存在未确定位置的不相容格,因此检查格子 3。与格子 3 不相容且未确定位置的是格子 5,而格子 5 相容于与格子 3 同行的格子 1,2,而与格子 4 不相容,因此格子 5 存入(2,3),得出图 4-17 所示的位置。

置格子 2,3,4 检查标志。

接下来检查格子 5。由于与格子 5 间不存在未确定且不相容的格子,因此置格子 5 已检查。最后将格子 6 放入($m+1=3,n+1=4$)位置。与该格不相容且未确定的格子 7 放入(3,5),置格子 2,6,7 已检查标志,得出最终棋盘如图 4-18 所示。

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | / | / |
| 2 | 4 | / | 5 | / | / |
| 3 | / | / | / | 6 | 7 |

图 4-18

## 2. 程序分析

```
program ddd;
  uses
    crt;
  const
    maxr          = 200;     { 最多格子数(即棋盘最大尺寸) }
  type
    settype       = set of 1..maxr;     { 集合类型 }
    btype         = array[1..maxr,1..maxr] of byte;  { 棋盘类型 }
    lstype        = array[1..maxr] of settype;  { 以集合为元素的数组 }
    ltype         = array[1..maxr] of integer;
    postype       = array[1..maxr,1..2] of byte;
  var
    b             : btype;  { 棋盘 }
    done,put      : ltype;
```

```
    { dome[i] = { 1 格子 i 已检查 / 0 格子 i 未检查 }   put(i) = { 1 格子 i 已确定位置 / 0 格子 i 未确定位置 }
    xp,yp,a          : lstype;
    { xp[i](yp[i])——第 i 行(i 列)中已确定位置的格子集合 }
    { a[i]——与格子 i 不相容的格子集合 }
    pos              : postype; { 记录当前格子的横纵坐标 }
    r,n,m,x,y        : integer;
    { r——格子数 n,m——已确定的棋盘规模,x,y—独立棋盘的规模 }
    name             : string;    { 输入文件名 }
    f                : text;      { 输入文件变量 }
  procedure init;  { 从输入文件中读入格子间不相容的信息至 a 数组 }
    var i,j,k : integer;
    begin
      clrscr;
      write('input file name = '); readln(name);
      assign(f,name); reset(f);
      { 读入文件名串,并与文件变量 f 连接;f 文件读准备 }
      readln(f,r); { 读入格子数 }
      for i:=1 to r do { 依次处理每一个格子 }
        begin
          a[i]:=[];    { 求与格子 i 不相容的格子集合 a[i] }
          for j:=1 to r do
            begin
              read(f,k);
              if k=1 then a[i]:=a[i]+[j];
            end;
        end;
      close(f);    { 关闭文件 }
      fillchar(b,sizeof(b),0); { 棋盘初始化 }
    end;
function fit(ss:settype;g:integer):boolean;
  { 若集合 ss 中的所有元素与 g 存在不相容关系,则返回 true,否则返回 false }
  var i : integer;
    begin
      fit:=false;
      for i:=1 to r do
        if (i in ss) and not (g in a[i]) then exit;
      fit:=true;
    end;
procedure add_one(x,y,e:integer);  { 将格子 e 确定在(x,y)位置 }
  begin
    b[x,y]:=e; { 格子 e 位于(x,y) }
    xp[x]:=xp[x]+[e]; yp[y]:=yp[y]+[e];
    { e 进入 x 行和 y 列已确定的格子集合 }
    pos[e,1]:=x; pos[e,2]:=y; { 存储格子 e 的行列坐标 }
    put[e]:=1; { 置格子 e 已确定标志 }
  end;
procedure turn;  { 将行数大于列数的独立棋盘旋转 90 度 }
```

```
    var i,j,k : integer;
  begin
    for i:=1 to r do
      if (done[i]=1) and (pos[i,1]>m) then
        begin k:=pos[i,1]-m;
              pos[i,1]:=m+(y+1-pos[i,2]+n);
              pos[i,2]:=n+k;
        end;
    k:=x; x:=y; y:=k;
    for i:=1 to y do for j:=1 to y do b[m+i,n+j]:=0;
    for i:=1 to r do
      if (done[i]=1) and (pos[i,1]>m) then b[pos[i,1],pos[i,2]]:=i;
  end;
procedure main;
  var i,j,hh,h,g : integer;
    begin
      n:=0; m:=0;
      for i:=1 to r do
        begin done[i]:=0; put[i]:=0; xp[i]:=[]; yp[i]:=[]; end;
      repeat hh:=0; x:=1; y:=1;     { 寻求一个未确定位置的格子 hh }
             repeat inc(hh); until (hh>r) or (done[hh]=0);
             if hh<=r then  { 若存在这样的格子 }
              begin
                add_one(m+1,n+1,hh); { 将格子放在独立棋盘的左上角 }
                repeat h:=0;
                       repeat inc(h);  { 寻求一已确定但未检查过的格子 h }
                       until (h>r) or (put[h]=1) and (done[h]=0);
                       if h<=r then  { 若存在这样的 h }
                         begin
                           done[h]:=1;     { 标志 h 已检查过 }
                           for g:=1 to r do
                { 搜索所有和 h 有不相容关系且未确定位置的格子 g }
                             if (h in a[g]) and (put[g]=0) then
                               if fit(xp[pos[h,1]],g)
                { 从 n+1 列开始逐列搜索,直至某列(n+i)的所有格子与 g 格互}
              { 不相容且 h 格所在行的(n+i)列未有格子或者超出 n+y 列为止}
                                 then begin
                                        i:=0;
                                        repeat inc(i);
                                        until (i>y) or (fit(yp[n+i],g))
                                              and (b[pos[h,1],n+i]=0);
                                        if i>y then
                          { 若搜索超出 n+y 列,则 g 格放入 h 格所在行的 n+y+1 列 }
                                          begin
                                            inc(y);
                                            add_one(pos[h,1],n+y,g);
                                          end
                                        else add_one(pos[h,1],n+i,g);
                                        { 否则放入 h 格所在行的 n+i 列 }
```

```
                    end
                  else if fit(yp[pos[h,2]],g) then
                    { g 和 h 同列 }
                    begin
                      i:=0;
                { 从 m+1 行开始逐行搜索,直至某行(m+i)的所有格子与}
                { y 格互不相容且 h 格所在列的(m+i)行未有格子或者超}
                { 出 m+x 行为止 }
                      repeat inc(i);
                      until (i>x) or (fit(xp[m+i],g))
                        and (b[m+i,pos[h,2]]=0);
                      if i>x then { 确定 g 的行号 i }
                { 若搜索超出 m+x 行,则 y 格放入 h 格所在列的 m+x+1 行 }
                       begin
                         inc(x);
                          add_one(m+x,pos[h,2],g);
                        end
                      else add_one(m+i,pos[h,2],g);
                        { 否则放入 h 格所在列的 m+i 行 }
                    end
                  else begin
                      { 若 g 与 h 非同行和同列,则无解 }
                        writeln('No Solution!!!');
                        halt;
                        end;
              end;
          until h>r;  { 当前已确定的棋子都已检查过 }
          if x>y then turn;  { 独立棋盘进入总棋盘 }
             m:=m+x; n:=n+y;
        end;
    until hh>r;  { 所有格子位置皆确定 }
  end;
procedure show;  { 显示棋盘 }
  var i,j : integer;
    begin
      for i:=1 to m do
        begin
          for j:=1 to n do
            if b[i,j]=0
              then write('  .')
              else write(b[i,j]:3);
          writeln;
        end;
      writeln('size=',m,'*',n,'=',m*n);
    end;
function con(p,q:integer):boolean;
  { 检查格子 p 和格子 q 是否在棋盘的同一行或同一列 }
  begin
    if (pos[p,1]=pos[q,1]) or (pos[p,2]=pos[q,2])
```

```
        then con:=true
        else con:=false;
      end;
function check:boolean;
{ 两两检查所有格子,若j格与i格互不相容但不在同行或同列,或者j格与i格相容但在同行同列,
  则返回 false;否则返回 true }
  var i,j : integer;
    begin
      check:=false;
      for i:=1 to r-1 do
        for j:=i+1 to r do
          if (j in a[i]) and not (con(i,j)) or
              not (j in a[i]) and (con(i,j)) then exit;
      check:=true;
    end;
begin
  init; { 读入 R*R 矩阵 }
  main; { 求最小棋盘 }
  if check { 若棋盘上的所有格子满足给定的相容或不相容关系,则打印解 }
    then show
    else writeln('No Solution!!!'); { 否则打印无解 }
end.
```

## 习 题 四

1. 已知集合 $S$ 中有 $N$ 种不同元素以及每种元素的个数,试求 $S$ 中取 $R$ 个元素的组合。

2. 已知 $x_1+x_2+\cdots+x_n=k$ 和 $N$ 个变量的正整数区间($k\geqslant N$)

求上述方程的所有整数解。

3. 设 $S=\{a_1,a_2,\cdots,a_k\}$是一个字符集,由 $S$ 生成长度为 $N$ 的字符串($k\leqslant N$),串中的字符可以相同,但 $S$ 中的每个字符至少出现一次。求这样的串的个数。

4. 已知 $S=\{1,2,\cdots,n\}$,求

(1) 没有偶整数在它的自然位置($i$ 不排在第 $i$ 个位置上)的排列数;

(2) 恰有 $k$ 个整数在其自然位置的排列个数

5. 令 $A_n$ 表示 $S=\{1,2,\ldots,n\}$的圆排列中没有 12,23,$\cdots$,$(n-1)n$,$n1$ 出现的排列个数。求 $A_n$ 和 $A_n$ 个圆排列方案。

6. $N$ 对夫妇($N\geqslant 3$)围圆桌就坐且男女交替,有多少种就坐方法使得没有任何一对夫妇相邻。

7. 有 $N$ 个小朋友排成一列纵队散步。分别求满足下列要求的变换队形的方法数。

(1) 若有一半男的和一半女的,列队时排成男女男女……的队形。若变换队形时始终保持男女交替且第一名是男的,但每一个女孩前面不再是原来的男孩;

(2) 当初相邻的不再相邻。

8. $S=\{n_1\cdot a_1,n_2\cdot a_2,\cdots,n_n\cdot a_n\}$的排列中,若任何两个相同的字符都不相邻就叫做

“简单字”。例如 $S=\{2\cdot a_1,2\cdot a_2,\cdots,2\cdot a_n\}$ 排列 $a_2a_1a_3a_2a_1a_3$ 是一个 $N=3$ 的简单字。求简单字的字数。

9. 求下述两个剖分数。

(1) 把 $n$ 分成各部分不能被 $d$ 所整除的剖分数；

(2) 把 $n$ 划成每一部分不出现 $d$ 次或 $d$ 次以上的剖分数。

# 第五章 母 函 数

## 5.1 母函数的引出

我们从二项式定理开始展开讨论。

二项式系数取名于熟知的二项式展开式：

$$(x+y)^n=\sum_{k=0}^{n}c(n,k)x^ky^{n-k}$$

从组合数学的角度看，$(x+y)\cdots(x+y)$共 $n$ 个括号相乘相当于 $n$ 个无区别的球，放到 $x$、$y$ 两个编号不同的盒子中去，每个盒放入的球数不限。展开成多项式后，幂 $x^ky^{n-k}$应看作 $n$ 个括号内有 $k$ 个提供 $x$，$n-k$ 个括号提供 $y$，即 $k$ 个球放到 $x$ 盒子中，$n-k$ 个球放到 $y$ 盒。而这 $k$ 个球是在 $n$ 个球中任选的，共有 $c(n,k)$种选法，所以 $x^ky^{n-k}$的系数为 $c(n,k)$。

若 $y=1$，则得出著名的二项式定理：

$$(1+x)^n=\sum_{k=0}^{n}c(n,k)x^k=c(n,0)+c(n,1)x+c(n,2)x^2+\cdots+c(n,n)x^n$$

我们可以通过二项式定理，引出许多排列组合的等式。例如

(1) 令 $x=1$，可得出 $c(n,o)+c(n,1)+\cdots+c(n,n)=2^n$

(2) 令 $x=-1$，可得出 $c(n,o)-c(n,1)+\cdots+(-1)^nc(n,n)=0$

(3) 由$(1+x)^m(1+x)^n=(1+x)^{m+n}$可推出

$$(c(m,0)+c(m,1)x+\cdots+c(m,m)x^m)(c(n,o)+c(n,1)x+\cdots+c(n,n)x^n)$$
$$=c(m+n,0)+c(m+n,1)x+\cdots+c(m+n,m+n)x^{m+n}$$

展开左式后比较等号两端 $x$ 项对应的系数，引出了一个组合等式

$$c(m+n,r)=c(m,o)c(n,r)+c(m,1)c(n,r-1)+\cdots+c(m,r)c(n,0)$$

还可以类似地推出一些等式，但通过上述例子已可清晰地看出$(1+x)^n$ 在研究序列 $c(n,0),c(n,1)\cdots c(n,n)$的关系时所起的作用。从二项式出发，人们自然会想到研究多项展开式

$$(x_1+x_2+\cdots+x_k)^n=\underbrace{(x_1+x_2+\cdots+x_k)\cdots(x_1+x_2+\cdots+x_k)}_{n\text{个括号相乘}}$$

的系数规律及其组合意义。

$(x_1+x_2+\cdots+x_k)\cdots(x_1+x_2+\cdots+x_k)$共 $n$ 个括号相乘，相当于 $n$ 个无区别的球放到 $x_1,x_2,\cdots,x_k$ 的 $k$ 个编号不同的盒子中去，每个盒放入的球数不限。展开多项式后，幂 $x_1^{n_1}x_2^{n_2}\cdots x_k^{n_k}$ $(n=n_1+n_2+\cdots+n_k)$应看作 $n_1$ 个括号提供 $x_1$，形成 $x_1^{n_1}$，依此类推，$n_k$ 个括号提供 $x_k$，形成 $x_k^{n_k}$，即 $n_1$ 个球放入 $x_1$ 盒，$n_2$ 个球放入 $x_2$ 盒$\cdots$，$n_k$ 个球放入 $x_k$ 盒。而 $n_1$ 个球

是在 $n$ 个球中任选的，$n_2$ 个球是在$(n-n_1)$个球中任选的，…，$n_k$ 个球是在$(n-n_1-n_2-\cdots-n_{k-1})$个球中任选的，因此形成 $x_1^{n_1}x_2^{n_2}\cdots x_k^{n_k}$ 的不同方式数是

$$c(n,n_1)c(n-n_1,n_2)\cdots c(n-n_1-\cdots-n_{k-1},n_k)=n!/(n_1!n_2!\cdots n_k!)$$

它是 $x_1^{n_1}x_2^{n_2}\cdots x_k^{n_k}$ 的系数，称之为多项式的系数。

因此得出

$$(x_1+x_2+\cdots+x_k)^n=\sum(n!/(n_1!\cdots n_k!))x_1^{n_1}x_2^{n_2}\cdots x_k^{n_k}\qquad(n=n_1+n_2+\cdots+n_k)$$

展开多项式后，我们便可以得出如下结论：

1. 展开后不同的单项的数目对应于从 $n$ 个不同元素中取 $k$ 个允许重复的组合数，因此总共项数为 $c(n+k-1,n)$。例如展开$(x_1+x_2+x_3+x_4)^3$ 后的项数为 $c(3+4-1,3)=20$(项)，分别为

$x_1^3,x_2^3,x_3^3,x_4^3$　　($x_i^3$ 的系数为 3! /(3! 0! 0! 0!)=1)

$3x_1^2x_2,3x_1^2x_3,3x_1^2x_4,3x_2^2x_1,3x_2^2x_3,3x_2^2x_4,$

$3x_3^2x_1,3x_3^2x_2,3x_3^2x_4,3x_4^2x_1,3x_4^2x_2,3x_4^2x_3$

($x_i^2x_J$ 的系数为 3! /(2! 1! 0! 0!)=3)

$6x_1x_2x_3,6x_1x_3x_4,6x_2x_3x_4,6x_1x_2x_4$

($x_ix_Jx_k$ 的系数为 3! /(1! 1! 1! 0!)=6)

2. 多项式中所有系数的和为 $k^n$。这个结论是很明显的。只要含 $x_1=x_2=\cdots=x_k=1$，即可看出

$$k^n=\sum(n!/(n_1!\cdots n_k!))$$

例如上题，20 项的系数的和为 $4*1+3*12+6*4=64=4^3$

多项展开式中的系数总和相当于 $n$ 个标号球放入 $k$ 个标号盒且允许空盒的方式数。

由此可见，展开一个二项式或多项式后，其系数序列可以帮助我们讨论许多问题。由此引入母函数的概念：

对于序列 $a_0,a_1,a_2\cdots$构造函数

$$G(x)=a_0+a_1x+a_2x^2+\cdots$$

称函数 $G(x)$是序列 $a_0,a_1,a_2\cdots$的母函数。例如$(1+x)^n$ 是如下序列的母函数。

$$c(n,0),c(n,1),\cdots,c(n,n)$$

的母函数。

如若已知序列 $a_0,a_1,a_2\cdots$，则对应的母函数 $G(x)$便可根据定义给出。反之，如若已求得序列的母函数 $G(x)$，则该序列也随之确定。

母函数的类型较多，这里仅讨论最常见的两种类型的母函数：

(1) 普通母函数；

(2) 指数母函数。

## 5.2 普通母函数

对于有限(或无限)序列 $a_0,a_1,\cdots,a_i,\cdots$，把一个"有限次(或 $n\to\infty$无限次)"多项式

$$a_0 + a_1x + a_2x^2 + \cdots + a_ix^i + \cdots = \sum_{i=0}^{n} a_ix^i$$

称为序列$\{a_i\}$的普通母函数。这类母函数在求解各类组合问题，特别是整数拆分问题时，发挥了巨大作用。

## 一、整数拆分

所谓整数拆分，即把整数分解成若干整数的和，相当于把 $n$ 个无区别的球放到 $n$ 个无标志的盒子，盒子允许空着，也允许放多个球，整数拆分成若干整数的和，方法不一，不同的拆分法的总数叫拆分数。例如

整数 2 的拆分数为 2，即 2=2=1+1；

整数 3 的拆分数为 3，即 3=3=2+1=1+1+1；

整数 4 的拆分数为 5，即 4=4=3+1=2+2=2+1+1=1+1+1+1；

### 1. 天平称物问题

设有质量分别为 $n_1$ 克，$n_2$ 克，…，$n_k$ 克的整数值砝码，欲称 $i$ 克的物体。物体在左、砝码在右，共有多少种不同称法？

设有 $a_i$ 种方法称 $i$ 克物体，则$\{a_0, a_1, \cdots, a_i, \cdots\}$作系数序列的母函数是

$$(1 + x^{n_1})(1 + x^{n_2})\cdots(1 + x^{n_k}) = \sum a_ix^i$$

这是因为每个括号$(1+x_{n_j})$如提供 1，表示 $n_j$ 克砝码没有用上；如提供 $x^{n_j}$，表示 $n^j$ 砝码用上了。这样一来，右边多项展开式中的每一个 $x^i$ 表示可称出 $i$ 克物体，$x^i$ 的系数 $a_i$ 便是称出 $i$ 克物体的方案数。

[例 1] 共有 1 克，2 克，3 克，4 克的砝码各一枚，问能称出哪几种重量，有几种可能方案？

$$(1 + x)(1 + x^2)(1 + x^3)(1 + x^4)$$
$$= 1 + x + x^2 + 2x^3 + 3x^4 + 2x^5 + 2x^6 + 2x^7 + x^8 + x^9 + x^{10}$$

从右端的母函数知，可称出 1，2，…，10 克，系数便是方案数。例如右端有

$2x^5$ 项，即称出 5 克的方案有 2，即 5=3+2=1+4；

$2x^6$ 项，即称出 6 克的方案有 2，即 6=1+2+3=4+2；

$x^{10}$项，即称出 10 克的方案有 1，即 10=1+2+3+4。

### 2. 允许重复的组合问题

设有几种相异物体。当每种物体的可取数为 1，则对应天平称物问题。但如果是允许重复，即每种物体的可取数依次为 $\lambda_1, \lambda_2, \cdots, \lambda_n (1 \leqslant \lambda_i \leqslant \infty)$，则从中取 $\gamma$ 个物体的可重复的组合数 $a_\gamma$ 为多少？

设取 $\gamma$ 个物体的不同方式数为 $a_\gamma$，则 $a_\gamma$ 作系数序列的母函数是

$$(1 + x + x^2 + \cdots x^{\lambda_1})(1 + x + x^2 + \cdots + x^{\lambda_2})$$
$$\cdots(1 + x + x^2 + \cdots x^{\lambda_n}) = a_0 + a_1x + \cdots + a_\gamma x^\gamma + \cdots$$

我们将第一个括号看作第 1 种物体 $A_1$，第二个括号看作第 2 种物体 $A_2$，…，第 $n$ 个括

号看作第 $n$ 种物体 $A_n$。它的展开式中 $x^\gamma$ 幂来源为

$$x^{m_1}x^{m_2}\cdots x^{m_n}=x^\gamma \qquad m_1+m_2+\cdots+m_n=\gamma$$

其中 $m_i(1\leqslant i\leqslant n)$ 为第 $i$ 个括号提供的 $x$ 的次幂数。好比第 1 种物体提供 $m_1$ 个，第 2 种物体提供 $m_2$ 个，……，第 $n$ 种物体提供 $m_n$ 个。这样一来，右边展开式中 $x_\gamma$ 的系数 $a_\gamma$ 恰好是 $n$ 种物体 $A_1,A_2,\cdots,A_n$ 中允许重复取 $\gamma$ 个物体的组合数 $c(n+\gamma-1,\gamma)$。即

$$(1+x+x^2+\cdots+x^{\lambda_1})(\quad)\cdots(1+x^2+\cdots x^{\lambda_n})$$
$$=\sum_{\gamma=0}c(n+\gamma-1,\gamma)x^\gamma$$

[**例 2**] 一口袋中有 5 个红球，3 个黄球，绿、白、黑球可任意多的提供。每次从中取 3 个，问有多少种不同取法？

**解**：取 $\gamma$ 个球的不同放法数为 $a_\gamma$，则 $\{a_\gamma\}$ 的母函数为

$$(1+x+x^2+x^3+x^4+x^5)(1+x+x^2+x^3)(1+x+x^2\cdots)$$
$$(1+x+x^2\cdots)(1+x+x^2\cdots)$$
$$=\sum_{\gamma=0}c(n+\gamma-1,\gamma)x^\gamma$$

显然当 $n=5,\gamma=3$ 时

$$a_3=c(5+3-1,3)=35$$

### 3. 整数拆分

上面两个问题都从逻辑上将每个物体 $A_i$ 看作是一个整体。如果 $A_i$ 可由某数字 $k_i$ 表征的话，那么问题就转化成整数拆分问题。

整数 $\gamma$ 拆分成 $k_1,k_2,\cdots,k_n$ 的和，其中 $k_1$ 允许重复 $n_1$ 次，$k_2$ 允许重复 $n_2$ 次，…，$k_n$ 允许重复 $n_n$ 次。(若 $k_i$ 允许无限次重复，则 $n_i=\infty(1\leqslant i\leqslant n)$) 显然整数 $\gamma$ 的拆分数为不定方程

$k_1\lambda_1+k_2\lambda_2+\cdots+k_n\lambda_n=\gamma$(其中 $\lambda_i$ 是 $k_i$ 的重复数，或 0 或正整数)中的变量 $\lambda_1$ 至 $\lambda_n$ 的非负整数解的个数。

设整数 $\gamma$ 的拆分数为 $b_\gamma$。为推出 $\{b_\gamma\}$ 的母函数，研究函数

$$[1+x^{k_1}+(x^{k_1})^2+\cdots+(x^{k_1})^{\lambda_1}+\cdots+(x^{k_1})^{n_1}][1+x^{k}$$
$$+(x^{k_2})^2+\cdots+(x^{k_2})^{\lambda_2}+\cdots+(x^{k_2})^{n_2}][\quad]\cdots[1+x^{k_n}$$
$$+(x^{k_n})^2+\cdots+(x^{k_n})^{\lambda_n}+\cdots+(x^{k_n})^{nn}]$$
$$=\sum_{\gamma=0}b_\gamma x_\gamma$$

右边展开式 $x^\gamma$ 幂来源于左边各个方括号，如

$$x^\gamma=(x^{k_1})^{\lambda_1}(x^{k_2})^{\lambda_2}\cdots(x^{k_n})^{\lambda_n}$$
$$=x^{k_1\lambda_1+k_2\lambda_2+\cdots+k_n\lambda_n}$$

其中第 1 个方括号提供 $x^{k_1\lambda_1}$，…，第 $n$ 个方括号提供 $x^{k_n\lambda_n}$。($\lambda_i=0$ 或正整数$(1\leqslant i\leqslant n)$)，显然

$$k_1\lambda_1+k_2\lambda_2+\cdots+k_n\lambda_n=\gamma$$

即展开式中 $x^\gamma$ 的系数就是上式非负整数解的个数——整数 $\gamma$ 的拆分数。

[例 3] 求用 1 分,2 分,3 分的邮票贴出不同数值的方案数

**解**: $G(x)=(1+x+x^2+\cdots)(1+(x^2)+(x^2)^2$
$+(x^2)^3+\cdots)(1+(x^3)+(x^3)^2+(x^3)^3+\cdots)$
$=1+x+2x^2+3x^3+4x^4+5x^5+7x^6+\cdots$

以其中 $x^4$ 为例,其系数为 4,即 4 拆分成 1,2,3 之和的拆分数为 4:

$$4=1+1+1+1=1+1+2=2+2=1+3$$

[例 4] 若有 1 克的砝码 3 枚,2 克的 4 枚,4 克的 2 枚,问能称出哪些重量? 各有几种方案?

**解**: $G(x)=(1+x+x^2+x^3)(1+x^2+(x^2)^2+(x^2)^3$
$+(x^2)^4)(1+x^4+(x^4)^2)$
$=1+x+2x^2+2x^3+3x^4+3x^5+4x^6+4x^7$
$+5x^8+5x^9+5x^{10}+5x^{11}+4x^{12}+4x^{13}+3x^{14}+3x^{15}$
$+2x^{16}+2x^{17}+x^{18}+x^{19}$

由上式可知,可以称出 1～19 克重量,其中以 $x^{17}$ 为例,其系数为 2。即 17 拆分成 1,2,4(其中 1 允许出现 3 次,2 允许 4 次,4 允许 2 次)之和的拆分数为 2

$$17=4+4+2+2+2+2+1$$
$$17=4+4+2+2+2+1+1+1$$

换句话讲,$1\cdot\lambda_1+2\cdot\lambda_2+4\cdot\lambda_3=17(\lambda_1=0,1,2,3,\lambda_2=0,1,2,3,4,\lambda_3=0,1,2)$的非负整数解为 2 个。

注意:在整数 $\gamma$ 拆分成 $k_1,k_2,\cdots,k_n$ 的和的过程中,若其中 $k_i$ 至少出现一次,其母函数应为

$$G(x)=[1+x^{k_1}+(x^{k_1})^2+\cdots(x^{k_1})^{n_1}][\quad]\cdots$$
$$[x^{k_i}+(x^{k_i})^2+\cdots.\ (x^{k_i})^{n_i}]\cdots[1+x^{k_n}$$
$$+(x^{k_n})^2+\cdots+(x^{k_n})^{n_n}]$$

[例 5] 现有 10 只球,分给 $A$、$B$、$C$、$D$ 四个盒,$A$ 盒不得少于 4 个,$B$ 盒不得多于 5 个,$C$ 盒只准放偶数(包含 0),$D$ 盒无限制,问有几种分法?

**解**: 如有 $\gamma$ 个球,接上述条件分配的方式数为 $b_\gamma$,则$\{b_\gamma\}$的母函数 $G(x)$为

$$G(x)=(x^4+x^5+\cdots)(1+x^2+x^3+x^4+x^5)(1+x^2+(x^2)^2$$
$$+(x^2)^3\cdots)(1+x+x^2+x^3\cdots)$$

下面,我们给出一个枚举整数拆分方案的程序例解。

[例 6] 输入正整数 $M$ 和 $k$,输出将 $M$ 拆分成 $k$ 项整数和的所有方案。

由交换律产生的诸个方案算作同一方案,例如 $M=6,k=3$ 时

$$1+2+3=6$$
$$1+3+2=6$$
$$2+1+3=6$$
$$\cdots\cdots$$

算作 $1+2+3=6$

算法分析:

若求解拆分数，借助普通母函数是再合适不过了。但要枚举所有拆分方案，母函数恐怕就鞭长莫及了，只有通过搜索的方法解决。我们采用回溯法搜索其和为 $M$ 的 $k$ 项数值，搜索顺序由第 1 项至第 $k$ 项。为了避免由交换律产生的重复方案，我们分别设定 1，2，…，$M$ 作为第 1 项的值，并按数值递增的顺序依次搜索以后各项的值。具体地说，若当前拆分方案的首项值为 $J(1\leqslant J\leqslant M)$，前 STEP($1\leqslant$STEP$\leqslant k$)项的值已经选定，第 STEP 项的值为 INDEX，接下去，我们则在 INDEX…$[M/k]$中选择一个与前 STEP 项的数和相加后还小于等于 $M$ 的值 $I$，设定 STEP+1 项的值为 $I$。……依次类推，直至设定 $k$ 项值的数和为 $M$ 为止。这样回溯搜索一次，便枚举了首项值为 $J$ 的所有拆分方案；回溯搜索 $M$ 次，所有不重复的拆分方案便自然完全产生出来了。

```
Program Problem1;
Const maxm      = 100;
Var     k,M     : integer;          {项数,数和}
        Ans     : array[1..Maxm] of integer;     {存储方案}
        Total   : integer;      {拆分数}
procedure print;
  var i : integer;
  begin
    inc(total);     {累计拆分数}
    write('Ans No.',total:4,' :');
    for i:=1 to k-1 do write(ans[i],'+');        {打印拆分方案}
    writeln(ans[k],' = ',m)
  end;
procedure Solve(step,index,sum:integer);  {step——形成的项数;index——第
                                           step 项的值;sum——第 1..step 项的和}
  var i:integer;
  begin
    if step=k then     {若拆分成 k 项,且数和为 M,则打印拆分方案;若 k 项的数和
                                    不等于 M,则执行空语句}
       if sum=m then print else
    else for i:=index to m do  {否则还未拆分第 k 项.在 index..m 之间选择某值
         if sum+i<=m then      i,使得前 step 项的数和加上 i 后小于等于 M}
           begin
             ans[step+1]:=i;          {i 值作为第 step 项}
             solve(step +1,i,sum+i)     {递归搜索下一项的值}
           end
  end;
var i:integer;
begin
    repeat write('M = ');     {输入数和 M}
           read(m)
    until m in [1..maxm];
    repeat write('k = ');     {输入项数 k}
      read(k)
```

```
  until k in [1..m];
  total:=0;                          {拆分数初始化}
  for i:=1 to m div k do             {试选 1..M,分别作为第 1 项的值}
    begin
      ans[1]:=i;
      solve(1,i,i)                   {i 作为第 1 项的值,递归搜索首项为 i 的所有拆分方案}
    end;
  readln;
end.

  end.
```

## 二、求普通母函数系数序列的实验程序

由上可知,如若已求得普通母函数,可以通过展开多项式的办法确定其系数序列。这个系数序列对研究组合问题有相当重要的意义。

下面,我们给出一个实验程序。该程序从一个指定文件中读入普通母函数。文件格式为:

每行表示一个多项式,按幂次数递增的顺序输入各项。每项系数在前,指数在后,各项间以空格隔开,每行最后加 00,表示一个多项式输入结束。行间的回车表明多项式之间的连乘关系。

例如:$1+2x+3x^2$ 的文件内容为:

```
1_0_2_1_3_2_0_0
^z
```

$(1+2x+3x^2)(1+x)$的文件内容为:

```
1_0_2_1_3_2_0_0
1_0_1_1_0_0
^z
```

$(1+3x^2)(1+5x^3)$的文件内容为:

```
1_0_0_1_3_2_0_0
1_0_0_1_0_2_5_3_0_0
^z
```

程序将展开多项式,并输出各 $x^\gamma$ 的系数 $\alpha^\gamma(\gamma=0,1,2,3,\cdots$,以帮助读者进行组合计数的分析。在给出程序之前,我们先对其算法作一简要的分析:

用一个单链表 $P$ 存储普通型母函数的展开式,链结点存储当前项,结点形式为

| 指数域 | 系数域 | 指向下一项的指针 |
|---|---|---|

显然,初始时 $P$ 为

$P\rightarrow$ | 0 | 1 | ^ |

依次读入各个多项式。每读当前多项式的一项时,先建立一个空链,然后该项与 $P$ 链

上的每一项相乘(P 结点的系数域值×读入的系数值,次幂域值+读入的次幂值),形成积项,并按合并同类项的办法并入 γ 链:

若积项的次幂为目前 γ 链最大,则积项接 γ 链尾;

若积项的次幂等于 γ 链的某项,则积项的系数加上 γ 链项的系数。若和为零,则删去 γ 链上的该项;

若积项的次幂在 γ 链相邻的两项之间,则积项插入中间位置。

当前多项式的各项读完后,P 链释放。若还有相乘的多项式,则将 γ 辗转赋值给 P,以保留当前结果。然后再按上述规则读入和处理下一个多项式,直至展开所有相乘的多项式为止。

```
Program ZHL_2;

type
  link          =  ^ node;  { 指针 }
  node          =  record  { 项结点 }
                     time : integer;  { 次幂 }
                     a    :  real;{ 系数 }
                     next : link     { 指向下项的指针 }
                   end;

var
  p,r           :  link; { p——存储以前各多项式的展开式 }
                         { r——加入当前多项式后展开的结果 }
  f             :  text; { 输入文件 }

procedure init; { 文件读前准备 }
var
  str:string;
begin
  write('File Name = ');
  readln(str);
  assign(f,str);
  reset(f)
end;

procedure free(p:link); { 释放 p 链 }
begin
  if p<> nil
     then begin
            free(p^.next);
            dispose(p)
          end
end;

procedure add(time:integer; a :real; r:link);
{ 通过合并同类项,将 aX^time 链入 r 链 }
var
  t : link;
begin
```

```
  if r^.next = nil { 若次幂 time 最大,则 aX^time 加入 r 链尾 }
     then begin
          new(r^.next);
          r^.next^.time := time;
          r^.next^.a:=a; r^.next^.next := nil
        end
    else if r^.next^.time = time { 将 aX^time 并入 r 中次幂为 time 的项 }
           then begin
                  r^.next^.a := r^.next^.a+a;
                  if r^.next^.a= 0
                      { 删去 r 中系数为 0 的项 }
                      then begin
                             t:=r^.next;
                             r^.next :=r^.next^.next;
                             dispose(t)
                           end;
                end
           else if (r^.next^.time > time) and (time>r^.time)
                   { 若 time 次幂在 r 的相邻项之间,则在中间插入 aX^time }
                   then begin
                          new(t);
                          t^.time:=time; t^.a := a; t^.next:=r^.next;
                          r^.next := t
                        end
                   else add(time,a,r^.next) { 往下递归合并 }
  end;
procedure proceed; { 计算若干多项式之积 }
var
  time : integer;
  a : real;
  t : link;
begin
  new(p); new(p^.next); { 中间 p 链初始化 }
  p^.next^.time := 0; p^.next^.a:=1; p^.next^.next := nil;
  read(f,a,time); { 读入第 1 个多项式的首项 }
  while not eof(f) do
    begin
      new(r); r^.next:=nil; { 展开式初始化 }
      repeat { aX^time 乘以 p 链上的各项,并合并同类项,结果存入 r 链 }
        if a<>0
          then begin
                 t:=p^.next;
                 while t<>nil do
                   begin
                     add(time+t^.time,a*t^.a,r);
                     t:=t^.next
                   end
```

```
                end;
          read(f,a,time) { 读入当前多项式的下一项 }
        until (a=0) or eof(f);
        read(f,a,time); { 再读下一项 }
        if not eof(f)
        { 若当前项不是最后一项,则将当前展开的结果转赋给 p, }
        { 释放以前各多项式的展开式 }
          then begin
                 t:=p; p:=r; free(t)
               end;
    end
end;
procedure print(r:link); { 打印结果 }
  begin
    if r<>nil
        then begin
          write('a',r^.time,'_',r^.a:3:0,'  ');
          print(r^.next)
        end
  end;
begin
  init;              { 文件读前准备 }
  proceed;           { 展开多项式 }
  print(r^.next);    { 打印展开式各项系数 }
  writeln
end.
```

## 5.3 指数母函数

设有 $n$ 个元素，其中元素 $a_1$ 重复了 $n_1$ 次，元素 $a_2$ 重复了 $n_2$ 次，…，$a_k$ 重复了 $n_k$ 次，$n=n_1+n_2+\cdots+n_k$。从中取 $\gamma$ 个元素的排列数记为 $b_\gamma$。那么如何求 $b_\gamma$ 呢？

设 $m_1$ 个 $a_1$，$m_2$ 个 $a_2$，…，$m_k$ 个 $a_k$（$m_1+m_2+\cdots+m_k=\gamma$，$m_i\leqslant\gamma$），则可得出互不相同的排列数有

$$\frac{\gamma!}{m_1!m_2!\cdots m_k!}$$

但 $m_i(1\leqslant i\leqslant k)$有约束条件 $0\leqslant m_i\leqslant n_i$。在这种情况下，正整数的 $k$ 部分拆 $\gamma=m_1+m_2+\cdots+m_k$ 有不同的分拆方式，设为 $p$ 种，则排列数 $b_\gamma$ 应为

$$b_\gamma=\sum_{p\text{种}}\left(\frac{\gamma!}{m_1!m_2!\cdots m_k!}\right)$$

[例 1] 若有 8 个元素，其中 $a_1$ 重复了 3 次，$a_2$ 重复了 2 次，$a_3$ 重复了 3 次。从中取 4 个排列的排列总数？

设从中取 $\gamma$ 个组合的方式数为 $c_\gamma$，则序列$\{c_0,c_1,\cdots\}$的母函数为

$$G(x)=(1+x+x^2+x^3)(1+x+x^2)(1+x+x^2+x^3)$$

$$=1+3x+6x^2+9x^3+10x^4+9x^5+6x^6+3x^7+x^8$$

从 $x^4$ 的系数可知，这 8 个元素取 4 个的组合数为 10。这 10 个组合方案可以下式得到：

$$\begin{aligned}&(1+x_1+x_1^2+x_1^3)(1+x_2+x_2^2)(1+x_3+x_3^2+x_3^3)\\=&1+(x_1+x_2+x_3)+(x_1^2+x_1x_2+x_2^2+x_1x_3+x_2x_3+x_3^2)\\&+x_1^3+x_1^2x_2+x_1x_2^2+x_1^2x_3+x_1x_2x_3+x_2^2x_3+x_1x_3^2+x_2x_3^2\\&+x_3^2)+(x_1x_3^3+x_2x_3^3+x_1^2x_3^2+x_1x_2x_3^2+x_2^2x_3^2+x_1^3x_3\\&+x_1^2x_2x_3+x_1x_2^2x_3+x_1^3x_2+x_1^2x_2^2)+\cdots\end{aligned}$$

其中 4 次方的项表达了从 8 个元素(即 $a_1$、$a_3$ 各 3 个、$a_2$ 2 个)中取 4 个的组合，即 4 在满足约束条件的情况下的拆分方案 $p=10$：

$$\begin{aligned}4=&1+0+3=0+1+3=2+0+2=1+1+2=0+2+2\\=&3+0+1=2+1+1=1+2+1=3+1+0=2+2+0\end{aligned}$$

下面研究从中取 4 个排列的不同排列总数。以 $x_1^2x_2^2$ 对应的两个 $a_1$ 和 两个 $a_2$ 的不同排列为例，其不同排列数为 $\frac{4!}{2!\ 2!}=6$，即 $a_1a_1a_3a_3$，$a_3a_3a_1a_1$，$a_1a_3a_1a_3$，$a_3a_1a_3a_1$，$a_1a_3a_3a_1$，$a_3a_1a_1a_3$ 六种。同样 $x_1x_3^3$ 对应的 1 个 $a_1$、3 个 $a_3$ 的不同排列数为 $\frac{4!}{1!\ 3!}=4$，即 $a_1a_3a_3a_3$，$a_3a_1a_3a_3$，$a_3a_3a_1a_3$，$a_3a_3a_3a_1$。余此类推，故从上式中可得：

$$\begin{aligned}b_4=&\frac{4!}{1!3!}+\frac{4!}{1!3!}+\frac{4!}{2!2!}+\frac{4!}{1!1!2!}+\frac{4!}{2!2!}+\frac{4!}{3!1!}\\&+\frac{4!}{2!1!1!}+\frac{4!}{1!2!1!}+\frac{4!}{3!1!1!}+\frac{4!}{2!2!1!}\\=&70(\text{种})\end{aligned}$$

为了便于 $b_\gamma$ 的计算，利用上述特点，我们形式地引进函数

$$g(x)=\left(1+x+\frac{x^2}{2!}+\cdots+\frac{x^{\lambda_1}}{\lambda_1!}\right)\left(1+x+\frac{x^2}{2!}+\cdots+\frac{x^{\lambda_2}}{\lambda_2!}\right)$$

$$\cdots\cdots\left(1+x+\frac{x^2}{2!}+\cdots+\frac{x^{\lambda_k}}{\lambda_k!}\right)=\sum_{\gamma=0}b_\gamma\frac{x^\gamma}{\gamma!}$$

上述函数 $g(x)$ 称为序列 $\{b_r\}$ 的指数母函数。

分析一下，多项式展开后是如何构成单项中 $x^\gamma/\gamma!$ 的系数 $b_\gamma$ 的：

设第 1 个括号提供 $x^{i_1}/i_1!$，第 2 个括号提供 $x^{i_2}/i_2!$，…第 $k$ 个括号提供，$x^{i_k}/i_k!$，而且恰有 $i_1+i_2+\cdots+i_k=\gamma$，则它们的乘积形式：

$$\begin{aligned}&(x^{i_1}/i_1!)\cdot(x^{i_2}/i_2!)\cdots\cdots(x^{i_k}/i_k!)\\=&[\gamma!/(i_1!i_2!\cdots i_k!)]\cdot[(x^{i_1+i_2+\cdots\gamma_k})/\gamma!]\\=&[\gamma!/(i_1!i_2!\cdots i_k!)]\cdot[x^\gamma/\gamma!]\end{aligned}$$

而为了形成乘积中的 $x^\gamma/\gamma!$，各括号内提供的 $x_i/i!$ 还可在约束条件 $0\leqslant i_1\leqslant\lambda_1$，……，$0\leqslant i_k\leqslant\lambda_k$ 内变化，所有乘积 $g(x)$ 中 $x^\gamma/\gamma!$ 的系数，应遍历这些变化，不难看出，就遍历 $p$ 种。因此

$$b_\gamma=\sum_{n\text{种}}\frac{\gamma!}{m_1!\cdots m_k!}$$

针对[例 1],其 $b_\gamma$ 的母函数显然为

$$g(x)=\left(1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}\right)\left(1+\frac{x}{1!}+\frac{x^2}{2!}\right)\left(1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}\right)$$

将 $g(x)$展开成具有$\sum_{\gamma=0} b_\gamma \frac{x^\gamma}{\gamma!}$形成的多项式后,取 $\gamma$ 个数的排列数便一目了然。

$$g(x)=1+3\frac{x}{1!}+9\frac{x^2}{2!}+28\frac{x^3}{3!}+70\frac{x^4}{4!}+170\frac{x^5}{5!}$$
$$+360\frac{x^6}{6!}+560\frac{x^7}{7!}+560\frac{x^8}{8!}$$

即取 1 个数的排列数为 3,取 2 个的排列数为 9,取 3 个的排列数为 28,取 4 个的排列数为 70,如此等等。

[**例 2**] 由 1,2,3,4 四个数字组成的五倍数中,要求数 1 出现次数不超过 2 次,但不能不出现;2 出现次数不超过 1 次,3 出现次数可达 3 次,也可以不出现;4 出现为偶数,求满足上述条件的数的个数。

**解**: 1 出现 1 次或 2 次,母函数有因子$\frac{x}{1!}+\frac{x^2}{2!}$;

2 出现不超过 1 次,母函数有因子 $1+x$;

3 出现次数可达 3 次,也可不出现,母函数有因子 $1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}$;

4 出现次数为偶数,因此在 5 位数中的出现次数为 0 或 2 或 4,母函数有因子 $1+\frac{x^2}{2!}+\frac{x^4}{4!}$。

若设 $\gamma$ 位数的个数为 $b_\gamma$,则根据上述条件,序列 $b_1,b_2,\cdots,b_\gamma$ 的指数母函数为

$$g(x)=\left(\frac{x}{1!}+\frac{x^2}{2!}\right)(1+x)\left(1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}\right)\left(1+\frac{x^2}{2!}+\frac{x^4}{4!}\right)$$

将 $g(x)$展开为$\sum_{r=0} b_\gamma \frac{x^\gamma}{\gamma!}$形式的多项式

$$g(x)=\frac{x}{1!}+5\frac{x^2}{2!}+18\frac{x^3}{3!}+64\frac{x^4}{4!}+215\frac{x^5}{5!}+645\frac{x^6}{6!}$$
$$+1785\frac{x^7}{7!}+140\frac{x^8}{8!}+7650\frac{x^9}{9!}+12600\frac{x^{10}}{10!}$$

由此可见,满足条件的 5 位数共 215 个。

下面我们给出一个实验程序,输入指数母函数的格式完全类同 5.2 节中普通母函数的实验程序。程序将展开指数母函数,并输出各$\frac{x^\gamma}{\gamma!}$系数 $b_\gamma(\gamma=0,1,2,\cdots)$,以帮忙读者进行排列计数的分析。

该程序的算法基本与展开普通型母函数的程序相同,仅有两点区别:

1. 读入项与 $P$ 链上每一项相乘的形式有所改变;即

   积项的系数值=$P$ 结点的系数域值×读入的系数值/次幂值的阶乘;

   积项的次幂值=$P$ 结点的次幂域值+读入的次幂值。

2. 展开指数型母函数后,依次输出各项的系数应为

   该项系数×次幂的阶乘

```
Program ZHL_22;
type
  link      =  ^node;  { 指针 }
  node      =  record  { 项结点 }
                  time :  integer;{ 次幂 }
                  a  :  real;     { 系数 }
                  next :  link    { 指向下项的指针 }
               end;
var
  p,r       :  link; { p——存储以前各多项式的展开式 }
                     { r——加入当前多项式后展开的结果 }
  f         :  text; { 输入文件 }
function pt(n:integer):real; { 计算 n! }
  begin
    if n=0
       then pt:=1
  else pt:=n * pt(n-1)
  end;
procedure init; { 文件读前准备 }
var
  str:string;
begin
  write('File Name = ');
  readln(str);
  assign(f,str);
  reset(f)
end;
procedure free(p:link); { 释放 p 链 }
begin
  if p<> nil
     then begin
            free(p^.next);
            dispose(p)
          end
end;
procedure add(time:integer; a :real; r:link);
{ 通过合并同类项,将 aX^time 链入 r 链 }
var
  t : link;
begin
  if r^.next = nil { 若次幂 time 最大,则 aX^time 加入 r 链尾 }
    then begin
           new(r^.next);
           r^.next^.time := time;
           r^.next^.a:=a; r^.next^.next := nil
```

```
        end
    else if r^.next^.time = time { 将aX^time并入r中次幂为time的项 }
            then begin
                   r^.next^.a := r^.next^.a+a;
                   if r^.next^.a= 0
                      { 删去r中系数为0的项 }
                      then begin
                             t:=r^.next;
                             r^.next := r^.next^.next;
                             dispose(t)
                           end;
                 end
            else if (r^.next^.time > time) and (time>r^.time)
                    { 若time次幂在r的相邻项之间,则在中间插入aX^time }
                    then begin
                           new(t);
                           t^.time:=time; t^.a := a; t^.next:=r^.next;
                           r^.next := t
                           end
                    else add(time,a,r^.next) { 往下递归合并 }
  end;
procedure proceed; { 计算若干多项式之积 }
var
  time : integer;
  a : real;
  t : link;
begin
  new(p); new(p^.next); { 中间p链初始化 }
  p^.next^.time := 0; p^.next^.a:=1; p^.next^.next := nil;
  read(f,a,time); { 读入第1个多项式的首项 }
  while not eof(f) do
    begin
      new(r); r^.next:=nil; { 展开式初始化 }
      repeat { aX^time乘以p链上的各项,并合并同类项,结果存入r链 }
        if a<>0
           then begin
                  t:=p^.next;
             while t<>nil do
               begin
                 add(time+t^.time,a*t^.a/pt(time),r);
                 t:=t^.next
               end
             end;
      read(f,a,time) { 读入当前多项式的下一项 }
    until (a=0) or eof(f);
    read(f,a,time); { 再读下一项 }
    if not eof(f)
```

```
    { 若当前项不是最后一项,则将当前展开的结果转赋给 p,}
    { 释放以前各多项式的展开式 }
      then begin
             t:=p; p:=r; free(t)
           end;
  end
end;
procedure print(r:link); { 打印结果 }
  begin
    if r<>nil
      then begin
             write('a',r^.time,'=',pt(r^.time)*r^.a:3:0,' ');
             print(r^.next)
           end
  end;
begin
  init;                { 文件读前准备 }
  proceed;             { 展开多项式 }
  print(r^.next);      { 打印展开式各项系数 }
  writeln
end.
```

## 习 题 五

1. 设 $s=\{\infty \cdot e_1,\infty \cdot e_2,\cdots,\infty \cdot e_k\}$,$a_n$ 是具有下列附加条件的 $s$ 的 $N$ 组合数，确定序数$\{a_n\}$的一般生成函数。

(1) 每一 $e_i$ 出现奇数次；

(2) 每一 $e_i$ 出现偶数次；

(3) 每一 $e_i$ 出现 $P$ 的倍数次；

(4) 每一 $e_i$ 至少出现 $X$ 次。

2. 口袋中有白球 $n_1$ 个，红球 $n_2$ 个，黑球 $n_3$ 个，每次从中取 $r$ 个，问有多少种不同取法？

3. 求将正整数 $M$ 划分成若干个 $a$，若干个 $b$，若干个 $c$，…方法数的生成函数？

4. 把正整数 $N$ 拆分成各项属于$\{1,2,\cdots,M\}$且 $M$ 至少出现一次的个数的生成函数。

5. 数 $N$ 的一个剖分，如果 $1,2,\cdots,N$ 的每一个整数，都可以用唯一的方法将它写成剖分中剖分项的和，则称该剖分是完备的。

例如 $7=4+2+1$ 就是 7 的一个完备剖分。这是因为

$2=2$

$3=2+1$

$4=4$

$5=4+1$

$6=4+2$

求 $N$ 的完备剖分的数目和方案。

6. 确定用红色、兰色、绿色和紫色为一个 $1\times N$ 的棋盘方块着色的方法个数，如果偶数个方块着红色并且偶数个方块着绿色。

7. 确定用 $N$ 个至少是 4 的数字组成的数的个数，其中 4 和 6 每一个出现偶数次，5 和 7 每个至少出现一次，数字 8 和 9 无限制。

8. 用 3 个 1，2 个 2，5 个 3 这十个数字能构成多少个偶的 4 位数。

# 第六章　递归关系

## 6.1　递归关系的定义和建立

递归是组合数学的一个重要课题，也是计算机编程解题时常用的算法。在这一节里，我们首先给出递归的定义，然后用几个例子来说明是怎样建立递归关系的。

递归定义：

设$\{u_0,u_1,\cdots,u_n,\cdots\}$是一个序列。如从$u_0$项后，$u_n$和前面若干项之间存在某种关系，此关系表明由前$n$项可推出$u_n$这种关系就称为递归关系。

例如：第四章 4.2 节中的错排数

$$D_n=(n-1)(D_{n-1}+D_{n-2}) \qquad \text{递归关系式}$$

$$D_1=0,D_2=1 \qquad \text{递归边界}$$

就是一个递归关系，其递归关系式连同递归边界一起唯一地确定了错排序列

$$\{D_1,D_2,D_3,\cdots\}=\{0,1,2,9,44,256\cdots\}。$$

由此可见，递归关系是计算序列中各项的有效工具。但如何建立递归关系呢？

下面用几个实例加以说明：

[**例 1**]　Hanoi 塔问题。$n$个大小不一的圆盘依半径的大小，自下而上套在柱子$A$上。如图 6-1 所示。

现要求将所有的圆盘从柱子$A$全部转移到柱子$C$上，每次只允许从一根柱子上转移一个圆盘到另一根柱子上，且转移过程中不允许出现大圆盘放在小圆盘上。试问要转移多少次才能将柱子$A$上的$n$个圆盘全部转移到$C$上去？

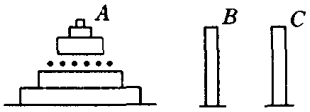



图　6-1

**解**：设$a_n$为从一根柱子上的$n$个圆盘全部转移到另一根柱子上的转移次数。显然，$a_1=1,a_2=3$。

当$n\geqslant 3$时，要将柱子$A$上的$n$个圆盘全部转移到柱子$C$上，可以这样设想：

先把柱子$A$上的$n-1$个圆盘转移到柱子$B$上，这需要转移$a_{n-1}$次，然后把柱子$A$上的最后一个大圆盘转移到柱子$C$上，显然这需要转移一次，最后再把柱子$B$上的$n-1$个圆盘转移到柱子$C$上，这也需要转移$a_{n-1}$次。经过这些步骤后，所有$A$上的圆盘就全部转移到柱子$C$上。由加法原理知，这一共转移了$2a_{n-1}+1$次，于是可以建立如下递归关系

$$a_n=2a_{n-1}+1 \qquad (n\geqslant 2)$$

$$a_1=1$$

[**例 2**]　在一个平面上有一个圆和$n$条直线，这些直线中的每一条在圆内都同其他直线相交。如果没有多于三条的直线相交于一点，试问这些直线将圆分成多少区域？

**解**：这$n$条直线将圆分成的区域数为$a_n$。显然$a_0=1$。如果有$n-1(n\geqslant 1)$条直线将圆

内分成 $a_{n-1}$ 个区域，那么再加入第 $n$ 条直线与在圆内的其他 $n-1$ 条直线相交，该直线被 $n-1$ 条直线在圆内分成 $n$ 条线段，而每线段又将第 $n$ 条直线在圆内经过的区域分成两个区域。这样，加入第 $n$ 条直线后，圆内就增加了 $n$ 个区域。于是对于每个整数 $n$，可以建立如下递归关系

$$a_n = a_{n-1} + n \qquad (n \geqslant 1)$$
$$a_0 = 1$$

递归关系的形式很多，其中有两种典型的递归类型：

1. $k$ 阶常系数线性齐次递归关系

一般形式为

$$a_n = b_1 a_{n-1} + b_2 a_{n-2} + \cdots + b_k a_{n-k} \qquad (n \geqslant k)$$

其中 $b_i (i=1,2,\cdots,k)$ 是常数，且 $b_k \neq 0$。

例如 6.2 节中的 Fibnacci 数

$$F_n = F_{n-1} + F_{n-2} \qquad (n \geqslant 2)$$
$$F_0 = F_1 = 1$$

即为 1 阶常系数线性齐次递归关系。

2. $k$ 阶常系数线性非齐次递归关系

一般形式为

$$a_n = b_1 a_{n-1} + b_2 a_{n-2} + \cdots + b_k a_{n-k} + f_n \quad (n \geqslant k)$$

其中 $b_i (i=1,2,\cdots,k)$ 是常数且 $b_k \neq 0, f_n \neq 0$

例如

[例 1]中的

$$a_n = 2a_{n-1} + 1 \qquad (n \geqslant 2)$$
$$a_1 = 1$$

[例 2]中的

$$a_n = a_{n-1} + n \qquad (n \geqslant 1)$$
$$a_0 = 1$$

即为 1 阶常系数线性非齐次递归关系。

上述两种类型的递归关系，可以通过解特征方程根、迭代法、归纳法、母函数法等数学方法直接求出序列 $\{a_0, a_1, \cdots, a_n, \cdots\}$ 中元素的函数关系式。例如[例 1]的递归关系式的解为 $a_n = 2^n - 1$；[例 2]的递归关系式的解为 $a_n = (n^2+n+2)/2$。但本书对这些方法不作评述，原因是：

1. 掌握这些数学规则所需的知识超出了中学生目前力所能及的范围。况且递归关系的类型还很多，当前还没有一般规则可以用来解所有类型的递归关系，特殊类型的递归关系式只有通过计算机编程来求解。

2. 通过数学规则求出的解只是一个计数，无法枚举递归过程中产生的方案。而递归过程容易将方案枚举出来。

3. 可直接按递归关系式编写程序，结构简单，有很好的可读性。

下面，我们着重介绍几个经典问题上的递归关系：

## 6.2 Fibonacci 数

“Fibonacci”兔子问题是组合数学的著名问题之一。这个问题是指：

从某一年开始时，把雌雄各一的兔子放入养殖场中，雌兔每月产雌雄各一的一对新兔。从第二个月开始每对新兔也是每月产一对兔子。试问第 $n$ 个月后养殖场中共有多少对兔子？

解：

设 $a$——小雄兔　　$b$——小雌兔

$A$——由 $a$ 长大的大雄兔　　$B$——由 $b$ 长大的大雌兔

第一个月兔子：$(a,b)$　　1 对

第二个月兔子：$(A,B)$　　1 对

第三个月兔子：$(A,B)+(a_1,b_1)$　　2 对

第四个月兔子：$(A,B)+(a_2,b_2)+(A_1,B_1)$　　3 对

第五个月兔子：$(A,B)+(a_3,b_3)+(A_2,B_2)$
$+(A_1,B_1)+(a_4,b_4)$　　5 对

第六个月兔子：$(A,B)+(a_5,b_5)+(A_3,B_3)$
$+(A_2,B_2)+(a_6,b_6)+(A_1,B_1)$
$+(a_7,b_7)+(A_4,B_4)$　　8 对

兔对之间表达了直接生殖关系。如果不再细分兔子对的历史，只需记为：第六个月有 $5(A,B)+3(a,b)$，再下一个月，大兔子对继续生存并每对生一对小兔（这一部分兔对数目恰与上一个月兔对数目一样），小兔对长大，小写字母变成大写字母。所以

第七个月兔子：$5(A,B)+5(a,b)+3(A,B)$　共 13 对

第八个月兔子：$8(A,B)+8(a,b)+5(A,B)$　共 21 对

第九个月兔子：$13(A,B)+13(a,b)+8(A,B)$　共 34 对

具体对数可见见图 6-2。

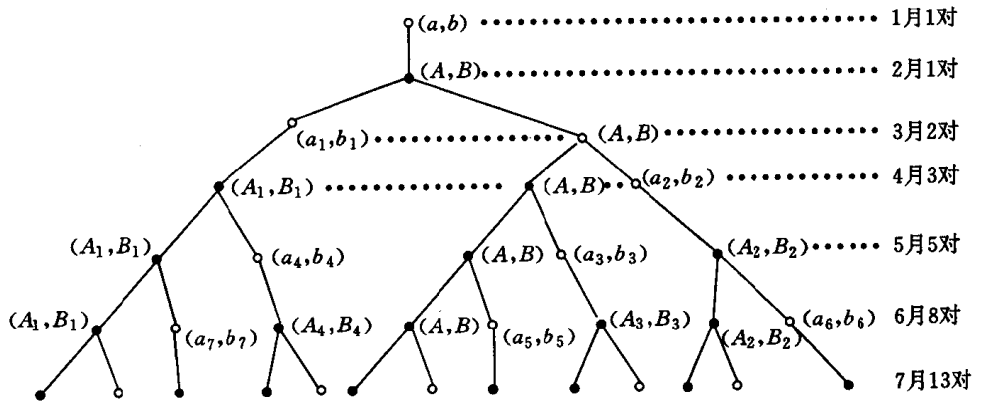



图　6-2

读者不难发现规律：

当月兔对数目＝上月兔对数＋上月中能生小兔的大兔对数

＝上月兔对数＋前月兔对数

对照图 6-2 来看

当月○和●数＝本月●数＋本月○数

＝上月●,○数＋上月●数

＝上月●,○数＋前月●,○数

如记 $F_n$ 为第 $n$ 个月开始时养殖场中的兔对数,则递归关系式为

$$F_n = F_{n-1} + F_{n-2} \qquad (n \geqslant 2)$$

$$F_0 = 1, \quad F_1 = 1$$

利用上式,不难手算出

$$(F_0, F_1, \cdots, F_n, \cdots)$$

$$= (1,1,2,3,5,8,13,21,34,$$

$$55,89,144,233,377,\cdots)$$

常称 $F_n$ 为 Fibonacci 数,$(F_0, F_1, \cdots, F_n, \cdots)$为 Fibonacci 序列。Fibonacci 序列在数学中是一个奇特而又常见的序列,它在算法分析中起着重要的作用。下面举两个例子:

[**例 1**] $2\times n$ 棋盘用 $1\times 2$ 骨牌作完全覆盖,求不同的覆盖方式数 $C_n$。

**解**:$2\times n$ 棋盘可看成两种形式:

第一种方式:$(a_{1,n}, a_{2,n})$位置上两格盖有一块骨牌,此时对 $2\times n$ 棋盘讲,共有 $C_{n-1}$种覆盖方式;

第二种方式:$(a_{1,n-1}, a_{1,n})$位置上两格盖有一块骨牌,此时必有$(a_{2,n-1}, a_{2,n})$。而此情况下 $2\times n$ 棋盘共有 $C_{n-2}$种覆盖方式

根据加法原理:　　$C_n = C_{n-1} + C_{n-2}$。

而 $n=1,2$ 时的覆盖方式为 $C_1 = C_2 = 1$

由此得出递归关系式

$$C_n = C_{n-1} + C_{n-2}$$

$$C_1 = C_2 = 1$$

显然,不同的覆盖方式数 $C_n$ 是 Fibonacci 数,$C_n = F_{n+1}$。

[**例 2**] 集合 $S=\{1,2,\cdots,n\}$,$V$ 是 $S$ 的不含相邻整数的子集,而 $|V|$ 只能取

$$0,1,2\cdots,[(n+1)/2]$$

问这样不同的子集共有多少个?

**解**:设 $f(n,k)$——$|V|=k$ 的不同子集个数。

构造一个二进制 $n$ 位数 $a_1, a_2, \cdots, a_n$。对于任意一个 $S$ 的子集 $V$,当 $i\in V$ 时置 $a_i=1$,否则 $a_i=0$。这样 $S$ 的子集与 $n$ 位二进制数一一对应,而子集 $V$ 对应不含两个相邻 1 的二进制数。

现要求 $|V|=k$,相当于有 $n-k$ 个 0 排成一行,第 1 个 0 之左,任两个 0 之间,最后一个 0 之右,共有 $n-k+1$ 个空隙,每个空隙至多插入一个 1,共插入 $k$ 个 1,从而形成的二进制数正是 $n$ 位无两个 1 相邻共 $k$ 个 1 的数,它正好是对应于 $|V|=k$ 的子集,见图 6-3。

由于二者之间一一对应,故有

$$f(n,k) = C(n-k+1,k)$$

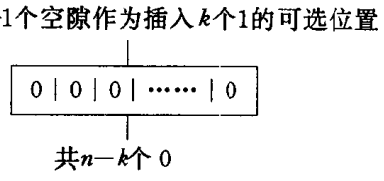


图 6-3

又由于|$V$|由 0 变化至|$(n+1)/2$|,根据加法原理这样的子集总共个数为

$$F_{n+1} = \sum_{k=0}^{[(n+1)/2]} f(n,k) - \sum_{k=0}^{[(n+1)/2]} C(n-k+1,k)$$

虽然 $F_0=1$,表示有一个空集,对应于二进制数 00…0,$F_1=1$,$F_2=C(1+0+1,0)+C(1-1+1,1)=2$,此时即 $S=\{1\}$,其子集为空集 $\varphi$ 和 $S$ 自身。$F_3=C(2-0+1,0)+C(2-1+1,1)=3$。此时 $S=\{1,2\}$,其子集:$\varphi$,{1},{2}。依此推导,不难发现

$$F_{n+1} = F_n + F_{n-1}$$

再比较初始条件,知 $F_n$ 是一个 Fibonacci 数,$F_n=f_{n-1}$。

下面,我们给出求 Fibonacci 数的程序。

```
program fibonacci;

  uses
    crt;

  const
    maxn                = 100;

  type
    listtype            = array[1..maxn] of integer;

  var
    F                   : listtype; {存储 Fibonacci 序列}
    n                   : integer;{自变量}

  procedure init;
    begin
      clrscr;
      repeat write('n='); {输入 Fibonacci 函数的自变量}
             readln(n);
      until (n>0) and (n<=maxn);
    end;

  procedure main;
    var i : integer;
      begin
        F[1]:=1; F[2]:=1;{设置递归边界}
        for i:=3 to n do     {递归}
          F[i]:=F[i-2]+F[i-1];
        writeln('F',n,'=',F[n]);
      end;

begin
  init; { 输入 Fibonacci 函数的自变量 }
  main; { 计算 Fibonacci 数 }
end.
```

## 6.3 Catalan 数

一个凸 $n$ 边形，通过不相交于 $n$ 边形内部的对角线，把 $n$ 边形拆分成若干三角形，不同拆分的数目用 $h_n$ 表之，$h_n$ 即为 Catalan 数。

例如五边形有如下五种拆分方案(图 6-4)，故 $h_5=5$。

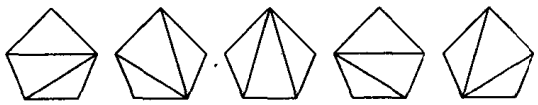


图 6-4

那么对于一个任意的凸 $n$ 边形，如何求 $h_n$ 呢？

我们设一个凸 $n$ 边形，以 $V_1V_{n+1}$ 作为一条边的三角形 $\triangle V_1V_kV_{n+1}$，将凸 $n+1$ 边分割成两部分。一部分是 $k$ 边形，另一部分是 $n-k+2$ 边形，$k=2,3,\cdots,n$，即 $V_k$ 可以是 $V_2$，$V_3,\cdots,V_n$ 中任意一点，见图 6-5。

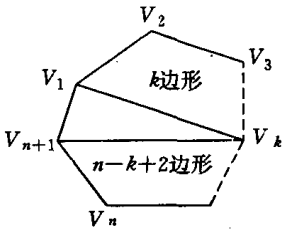



图 6-5

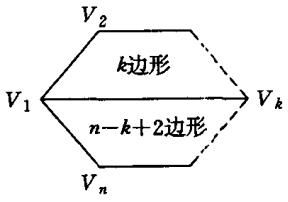



图 6-6

根据加法原理，引出 $h_{n+1}$ 的递归关系式 1：

$$h_{n+1}=\sum_{k=2}^{n}h_kh_{n-k+2}$$
$$=h_2h_n+h_3h_{n-1}+\cdots+h_{n-1}h_3+h_nh_2$$
$$h_2=1$$

我们还可以通过另外一种方法引出 $h_n$ 的递归关系式：

从 $V_1$ 点向其它 $n-3$ 个顶点 $\{V_3,V_4,\cdots,V_{n-1}\}$ 可以引出 $n-3$ 条对角线。其中对角线 $V_1V_k$ 把 $n$ 边形分割成两部分，见图 6-6。

因此 $V_1V_k$ 对角线作为拆分线的方案数为 $h_kh_{n-k+2}$。$V_k$ 可以是 $V_3,V_4,\cdots,V_{n-1}$ 中任一点，对所有这些点求和得：

$$h_3h_{n-1}+h_4h_{n-2}+\cdots+h_{n-2}h_4+h_{n-1}h_3$$

以 $V_2,V_3,\cdots,V_n$ 取代 $V_1$ 点也有类似结果。但考虑到对角线有两个顶点，同一对角线在两个顶点分别计算了一次，作

$$(n/2)\cdot(h_3h_{n-1}+h_4h_{n-2}+\cdots+h_{n-2}h_4+h_{n-1}h_3)$$

但这并不就给出拆分数，无疑其中是有重复的，其重复度是由于一个凸边形的剖分有

$n-3$ 条对角线，而对每一条边计数时该剖分计数了一次，故重复了 $n-3$ 次，即上式的结果是 $h_n$ 的 $n-3$ 倍，因此，引出 $h_n$ 的递归关系式 2：

$$h_n = \{n/[2(n-3)]\} \cdot (h_3h_{n-1} + h_4h_{n-2} + \cdots + h_{n-2}h_4 + h_{n-1}h_3)$$

$$h_2 = 1$$

对于以上两个递归关系式，我们可以通过下述办法将之转换为关于 $n$ 的函数关系：

将 $h_2=1$ 代入 $h_{n+1}$ 的递归关系式 1，得

$$h_{n+1} = 2h_n + h_3h_{n-1} + h_4h_{n-2} + \cdots + h_{n-1}h_3$$

$$= [2(2n-3)h_n]/n \quad \text{（根据 } h_n \text{ 的递归关系式 2 得出）}$$

所以 $$n \cdot h_{n+1} = (4n-6)h_n$$

令 $f_{n+1} = n \cdot h_{n+1}$，则 $f_2 = h_2 = 1$，

$$f_{n+1} = (4n-6) \cdot f_n/(n-1)$$

$$= \frac{(2n-2)(2n-3)}{(n-1)(n-1)} f_n$$

$$f_{n+1} = \frac{f_{n+1}}{f_n} \cdot \frac{f_n}{f_{n-1}} \cdot \frac{f_{n-1}}{f_{n-2}} \cdot \cdots \cdot \frac{f_4}{f_3} \cdot \frac{f_3}{f_2}$$

$$= \frac{(2n-2)(2n-3)}{(n-1)(n-1)} \cdot \frac{(2n-4)(2n-5)}{(n-2)(n-2)} \cdot \cdots \cdot \frac{4 \cdot 3}{2 \cdot 2} \cdot \frac{2 \cdot 1}{1 \cdot 1}$$

$$= \frac{(2n-2)!}{(n-1)!(n-1)!} = C(2n-2, n-1) = n \cdot h_{n+1}$$

所以引出 $h_{n+1}$ 的函数式

$$h_{n+1} = (1/n) \cdot C(2n-2, n-1)$$

$h_{n+1}$ 的函数式可以帮助我们将许多有意义的计数问题引导到对 Catalan 数的分析。

[**例 1**] $n$ 个 1 和 $n$ 个 0 组成一个 $2n$ 位的二进制数，要求从左到右扫描，1 的累计数不小于 0 的累计数。试求满足这种条件的数有多少？

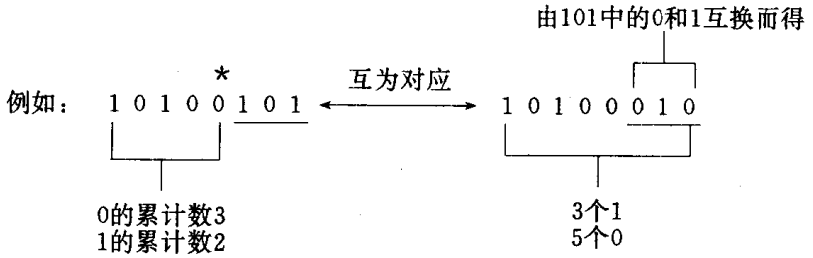
**解**：设 $P_{2n}$ 为满足题意的数的个数。

显然 $P_{2n}$ = ($2n$ 位上填入 $n$ 个 1 的方案数 $C(2n,n)$) − (从左至右扫描过程中 0 的累计数超过 1 的累计数的方案数)。

0 的累计数超过 1 的累计数的一个 $2n$ 位二进制数是一个不合要求的数，它的特征是从左至右扫描时，必然在某 $2m+1$ 位上首先出现 $m+1$ 个 0 的累计数和 $m$ 个 1 的累计数。此后的 $2(n-m)-1$ 位上有 $n-m$ 个 1 和 $n-m-1$ 个 0。如若把后面这部份的 $2(n-m)-1$ 位的 0 与 1 互换，使之成为 $n-m$ 个 0 和 $n-m-1$ 个 1，结果得 1 个由 $n+1$ 个 0 和 $n-1$ 个 1 组成的 $2n$ 位数。即一个不合要求的数对应于一个由 $n+1$ 个 0 和 $n-1$ 个 1 组成的一个排列，这样的方案数为 $C(2n,n+1)$。

反过来，任何一个由 $n+1$ 个 0、$n-1$ 个 1 组成的 $2n$ 位数，由于 0 的个数多二个，$2n$ 是偶数，故必在某一个奇数位上出现 0 的累计数超过 1 的累计数。同样在后面的部份，令 0 和 1 互换，使之成为由 $n$ 个 0 和 $n$ 个 1 组成的 $2n$ 位数。即 $n+1$ 个 0 和 $n-1$ 个 1 组成的 $2n$ 位数，必对应于一个不合要求的数。

用上述方法建立了由 $n+1$ 个 0 和 $n-1$ 个 1 组成的 $2n$ 位数，与由 $n$ 个 0 和 $n$ 个 1 组




成的 $2n$ 位数中从左向右扫描出现 0 的累计数超过 1 的累计数的数一一对应，由此可得出：

$$\begin{aligned} P_{2n} &= C(2n,n) - C(2n,n+1) \\ &= \frac{(2n)!}{(n+1)n!} = \frac{1}{(n+1)}C(2n,n) \\ &= h_{n+3} \end{aligned}$$

因此 $P_{2n}$是一个 Catalan 数。

[**例 2**] $P=a_1,a_2,\cdots,a_n$ 依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，试问有几种乘积方案？编程枚举所有方案。

**解**：令 $P_n$——$n$ 个数乘积的 $n-1$ 对括号插入的不同方案数。则

$$P_n = P_1P_{n-1} + P_2P_{n-2} + \cdots + P_{n-1}P_1 \quad . \quad P_1 = P_2 = 1$$

令 $P_k=h_{k+1},k=1,2,\cdots,n$，故得

$$h_{n+1} = h_2h_n + h_3h_{n-1} + \cdots + h_{n-1}h_3 + h_nh_2$$

而且 $h_2=h_3=1$。故 $P_n$ 即为 Catalan 数 $h_{n+1}$。

以 $n=4$ 为例，$P_4=h_5=\frac{1}{4}C(6,3)=5$。方案为

$((a_1(a_2\ a_3))a_4),(((a_1\ a_2)a_3)a_4)$,

$((a_1\ a_2)(a_3\ a_4)),(a_1(a_2(a_3\ a_4))),$

$(a_1((a_2\ a_3)a_4))$

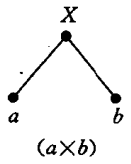



图 6-7

设二叉树表示 $a\times b$ 运算，如图 6-7 所示。

下面建立 $P_4$ 的 5 个不同乘法方案和一个 5 边形不同拆分的一一对应关系，见图 6-8。

下面，根据 $P_n$ 的递归定义设计一个枚举所有不同乘法方案的程序，供读者参考。

算法分析：

由于最里层的一对括号内为两元素乘积，因此 $n$ 个元素可嵌入 $n-1$ 对括号，其方案数为

$$h_{n+1} = \frac{1}{n}C(2n-2,n-1)。$$

初始时，设$(a_1,a_2,\cdots,a_{n-1},a_n)$。

我们可将 $1,2,\cdots,(n-1)$的一个排列

$$b_1,b_2,\cdots,b_{n-1}$$

转换成一种乘积方案。方法如下：

设$a_1$ 前的括号嵌入位置为 1；

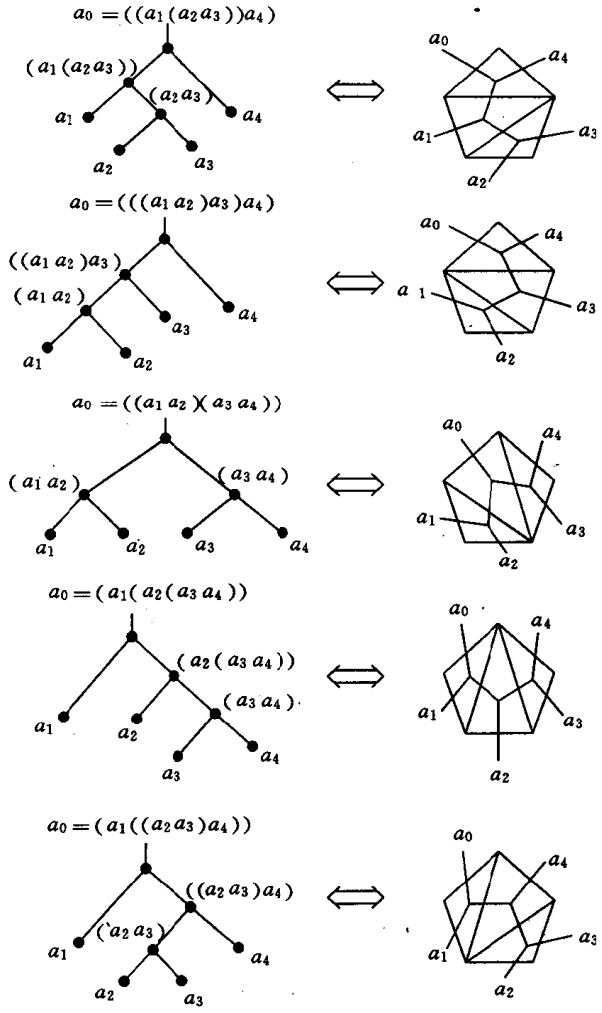



图 6-8

$a_1$ 与 $a_2$ 之间的括号嵌入位置为 2；

……

$a_{n-1}$与 $a_n$ 之间的括号嵌入位置为 $n$；

排列中的元素 $b_i$ 作为‘(’(或‘)’)的嵌入位置，而与之配对的‘)’(或‘(’)的嵌入位置用变量 $h$，$t$ 表示。

具体地讲

$b_i$ 位置嵌入‘)’，则 $h-1$ 位置嵌入一个‘(’与之配对；

$b_i$ 位置嵌入‘(’，则 $t$ 位置嵌入一个‘)’与之配对；

嵌入规则如下：

初始时，设 $h=1$，$t=n$，$(a_1,a_2,\cdots,a_n)$。从 $b_1$ 开始搜索第二对括号的嵌入位置：

(1) $b_i>h$

$$\downarrow \qquad \downarrow$$

$(a_{h-1}\cdots(a_h\cdots a_{bi})\cdots)$

然后试嵌 $b_{i+1}$位置的下一对括号($h$ 不变，$t=b_i$)

(2) $t>b_{i+1}$

$(\cdots a_{bi}\cdots(a_{bi+1}\cdots a_t)\cdots)$

↑ ↑

然后试嵌 $b_{1+b_i-h+1}$位置的下一对括号($h=b_{i+1}$,$t$ 不变)

(3) $b_i\geqslant t$ 和 $b_i\leqslant h$,成功退出或失败退出

例如 $b=(1,3,2)$

初始时$(a_1a_2a_3a_4)$,$h=1$,$t=4$

$b_1=1$,根据规则 2($t=4>b_i+1=2$),$(a_1(a_2a_3a_4))$然后试嵌 $b_{1+1-1+1}=b_2=3$ 位置的一对括号($h=2$,$t=3$);

$b_2=3$,根据规则 1($b_2=3>h=2$),$(a_1((a_2a_3)a_4))$,然后试嵌 $b_{i+1}=b_3=2$ 位置的一对括号($h=2$,$t=3$)

$b_3=2$,根据规则 3($b_3=2\leqslant h=2$),成功退出。即

$b=(1,3,2)$对应的乘法方案是$(a_1((a_2a_3)a_4)$

又如 $b=(2,3,1)$

初始时$(a_1a_2a_3a_4)$,$h=1$ $t=4$

$b_1=2>h=1$,根据规则 1,$((a_1a_2)a_3a_4)$,然后试嵌 $b_2=3$ 位置的一对括号($h=1$,$t=2$)

$b_2=3\geqslant=2$,根据规则 3,失败退出。即

$b=(1,3,2)$无对应的乘法方案;

同理,$b=(1,2,3)$对应的乘法方案是$(a_1(a_2(a_3a_4)))$;

$b=(3,1,2)$对应的乘法方案是$((a_1(a_2a_3))a_4)$;

$b=(3,2,1)$对应的乘法方案是$(((a_1a_2)a_3)a_4)$;

$b=(2,1,3)$对应的乘法方案是$((a_1a_3)(a_3a_4))$;

下面给出程序题解:

```
program s_item;
  uses
    crt;
  const
    maxn          = 50;
  type
    list          = array[1..maxn] of integer;
  var
    n,t           : integer;  { n 为元素数,t 为方案数 }
    a,e,d         : list;  { a 为排列; e,d 为求排列时的辅助变量 }
    kuo           : array[0..maxn] of string[maxn]; { kuo[i]为元素 ai 后括号的排法 }
    good          : boolean;  { 当前排列是否对应一种乘法 }
  procedure init;
    var i : integer;
      begin
        clrscr;
```

```
    repeat write('n='); {输入元素数}
           readln(n);
    until (n>0) and (n<=maxn);
    dec(n);
    for i:=1 to n do
      begin
        a[i]:=i;    { 建立初始排列 }
        d[i]:=i;
        e[i]:=-1;
      end;
    t:=0;
  end;
procedure make_kuo(list_head,list_tail,a_head:integer);
  { 将排列转换成具体的乘法方案,即括号的填法 }
  begin
    if a[a_head]>=list_tail
      then begin
             good:=false;
             exit;
           end;
    if a[a_head]-list_head>0
      then begin
             kuo[list_head-1]:=kuo[list_head-1]+'(';
             kuo[a[a_head]]:=')'+kuo[a[a_head]];
             make_kuo(list_head,a[a_head],a_head+1);
             if not good
               then exit;
           end;
    if list_tail-a[a_head]>1
      then begin
             kuo[a[a_head]]:=kuo[a[a_head]]+'(';
             kuo[list_tail]:=')'+kuo[list_tail];
             make_kuo(a[a_head] + 1, list_tail, a_head+ a[ a_head] -list_head+
             1);
           end;
  end;
procedure show;    { 显示当前排列对应的乘法方案 }
  var i : integer;
    begin
      for i:=1 to n do  {乘法方案初始化}
        kuo[i]:=' ';
      kuo[0]:='(';
      kuo[n+1]:=')';
      good:=true;       {设当前排列可对应一种乘法方案}
      make_kuo(1,n+1,1); {将排列转换为乘法方案}
      if good then      {若转换成功,则打印乘法方案}
        begin
```

```
            inc(t);
            write(t:3,' : ');
            write(kuo[0]);
            for i:=1 to n+1 do
              write('a',i,kuo[i]);
            writeln;
          end;
      end;

  procedure main;
    { 生成所有的排列，显示各排列对应的乘法方案 }
    var k,r,p,q   : integer;
        next      : boolean;
      begin
        repeat q:=0;
               show;          {显示当前排列对应的乘法方案}
               k:=n;
               next:=false;
               repeat
                   d[k]:=d[k]+e[k];
                   p:=d[k];
                   if p=k
                       then e[k]:=-1
                       else if p=0
                                then
                                  begin
                                    e[k]:=1;
                                    q:=q+1;
                                  end
                                else
                                  begin
                                    p:=p+q;
                                    r:=a[p];
                                    a[p]:=a[p+1];
                                    a[p+1]:=r;
                                    next:=true;
                                  end;
                   dec(k);
               until (next) or (k=1);{直至 1…(n-1)的一个排列生成}
        until not next;                  {直至全部排列生成}
    end;

begin
  init;    { 输入元素数,建立初始排列 }
  main;    { 计算所有排列对应的乘法方案 }
  writeln('total number of ways : ',t); { 输出方案数 }
  readln;
end.
```

## 6.4 第二类 Stirling 数

设 $S$ 是一个具有 $n$ 个元素的集合 $S=\{a_1,a_2,\cdots,a_n\}$，现将 $S$ 集合划分成 $k$ 个满足下列条件的子集合 $S_1,S_2,\cdots,S_k$：

(1) $S_i\neq\varphi$

(2) $S_1\cap S_j=\varphi$

(3) $S_1\cup S_2\cup S_3\cup\cdots\cup S_n=S$

$(1\leqslant i,j\leqslant k \quad i\neq j)$

则称 $S_1,S_2,\cdots,S_n$ 是 $S$ 的一个划分。它相当于把 $S$ 集合中的 $n$ 个元素 $a_1,a_2,\cdots,a_n$ 放入 $k$ 个无标号的盒子中，使得没有一个盒子为空的方式数。

例如 $S=\{1,2,3,4\}$，$k=3$。细心的读者稍加分析后，不难得出 $S$ 有 6 种不同的划分方案，即划分数为 6。其方案为

$\{1,2\}\cup\{3\}\cup\{4\}$　　$\{1,3\}\cup\{2\}\cup\{4\}$

$\{1,4\}\cup\{2\}\cup\{3\}$　　$\{2,3\}\cup\{1\}\cup\{4\}$

$\{2,4\}\cup\{1\}\cup\{3\}$　　$\{3,4\}\cup\{1\}\cup\{2\}$

如果对于任意的 $S$ 集合和 $k$ 值，就不能凭藉直觉和经验计算划分数和枚举划分方案了。必须总结出一个数学规律：

设 $n$ 个元素 $a_1,a_2,\cdots,a_n$ 放入 $k$ 个无标号盒的划分数为 $S(n,k)$，在配置过程中，有两种互不相容的情况：

(1) 设 $\{a_n\}$ 是 $k$ 个子集中的一个子集，于是把 $\{a_1,a_2,\cdots,a_{n-1}\}$ 划分为 $k-1$ 子集有 $S(n-1,k-1)$ 个划分数；

(2) 如果 $\{a_n\}$ 不是 $k$ 个子集中的一个，即 $a_n$ 必与其它的元素构成一个子集。首先把 $\{a_1,a_2,\cdots,a_{n-1}\}$ 划分成 $k$ 个子集，这共有 $S(n-1,k)$ 种划分方式。然后再把 $a_n$ 加入到 $k$ 个子集中的一个子集中去，这有 $k$ 种加入方式。对于每一种加入方式，都使集合划分为 $k$ 个子集，因此由乘法原理知，划分数共有

$$k\cdot S(n-1,k)$$

应用加法原理于上述两种情况，得出 $\{a_1,a_2,\cdots,a_n\}$ 划分为 $k$ 个子集的划分数：

$$S(n,k)=S(n-1,k-1)+k\cdot S(n-1,k)$$

$$(n>1,k\geqslant 1)$$

另外，我们不可能把 $n$ 个元素不放进任何一个集合中去，即 $S(n,0)=0$；也不可能在不允许空盒的情况下把 $n$ 个元素放进多于 $n$ 的 $k$ 个集合中去，即 $k>n$ 时 $S(n,k)=0$。

显然，通过上述分析可得出划分数 $S(n,k)$ 的递归关系式：

$$S(n,k)=S(n-1,k-1)+k\cdot S(n-1,k) \qquad (n>k,k\geqslant 1)$$

$$S(n,k)=0 \qquad (n<k) \text{ 或 } (k=0<n)$$

我们称 $S(n,k)$ 为第二类 Stirlihg 数，通常记为 $S_2(n,k)$。

第二类 Stirling 数 $S_2(n,k)$ 具有下列性质：

(1) $S_2(n,1)=1$

$S_2(n,n)=1$

上述性质是显然的。

(2) $S_2(n,2)=2^{n-1}-1$

证：将 $n$ 个不同元素放入 $n$ 个无标号的盒子，首先取出 $a_i(1\leqslant i\leqslant n)$元素。其余的 $n-1$ 个元素每个都有与 $a_i$ 同盒或不同盒两种选择，即有 $2^{n-1}$种选择。但必须排除全体与 $a_n$ 同盒的情况，因这时另一盒将是空盒，因此实际上只有 $2^{n-1}-1$ 种划分方案，即

$$S_2(n,2) = 2^{n-1} - 1$$

(3) $S(n,n-1)=C(n,2)$

证：将 $n$ 个元素放到 $n-1$ 个盒子里，不允许有一空盒，故必有一盒有两个球。从 $n$ 个不同元素中任取 2 个的组合数为 $C(n,2)$，即

$$S_2(n,n-1) = C(n,2)$$

由第二类 Stirling 数 $S_2(n,k)$的定义出发，可引出许多有趣的结论：

[**例 1**] 把 $n$ 个元素放入 $k$ 个不同编号的盒子中，使得没有一个盒子为空的方式数是 $k!\ S_2(n,k)$

**证**：首先不考虑盒子的编号，显然有 $S_2(n,k)$种方式；

其次考虑盒子是编号的。这样，$k$ 个盒子共有 $k!$ 种编号方式。于是由乘法原理得出，共有 $k!S_2(n,k)$种方式，把 $n$ 个不同元素放入 $k$ 个不同编号的盒子中去，而且没有一个盒子为空。

[**例 2**] $S=\{a_1,a_2,\cdots,a_n\}$全部划分的方式数为

$$B_n = \sum_{i=0}^{n} S_2(n,i)$$

**证**：设 $S$ 中的元素放入 $i$ 个无标号盒且不允许空盒的方式数为 $S_2(n,i)$。$S$ 可划分为 1 个子集，2 个子集，…，$n$ 个子集，即 $i$ 可从 1 变化至 $n$。所有各类划分数的和

$$\sum_{i=1}^{n} S_2(n,i)$$

就是全部划分的方式数，记为 $B_n$，有时称之为 Bell 数。

例如，图 6-9 表示 $S=\{a,b,c,d\}$的全部划分。

由图可见，$S=\{a,b,c,d\}$的全部划分数为

$$B_4 = \sum_{i=1}^{4} S_2(4,i) = 1 + 7 + 6 + 1 = 15$$

下面给出求 $S=\{a_1,a_2,\cdots,a_n\}$的全部划分的方式数 $B_n$ 以及全部划分方案的程序。在给出程序之前，我们先简单介绍算法，以帮助读者理解。

首先，我们必须明确

将 $i$ 个元素的集合划分为 1 个子集，仅有 1 个方案$\{1,2,\cdots,i\}$，即 $S=(i,1)=1$；划分为 $i$ 个不相交的子集亦仅有 1 个方案$\{1\},\{2\},\cdots,\{i\}$，即 $S_2(i,i)=1$。

我们从上述两个边界条件出发，根据公式 $S_2(i,j)=S_2(i-1,j-1)+j\times S_2(i-1,j)$ 的定义进行递推。

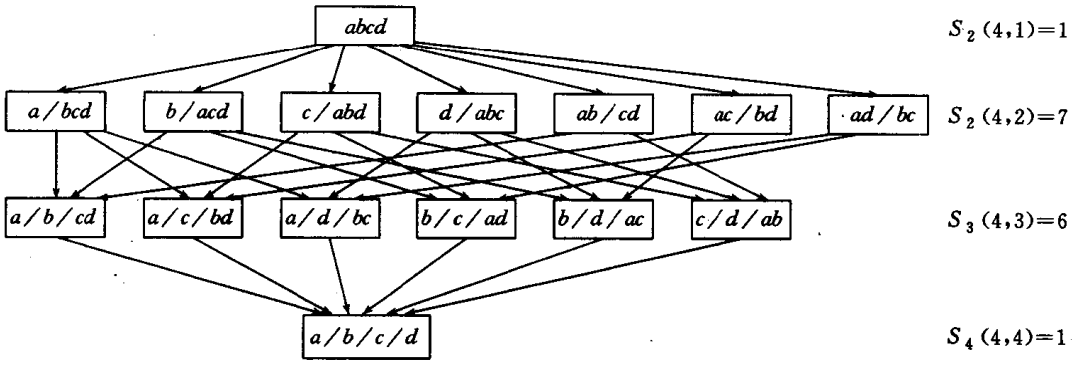



图 6-9

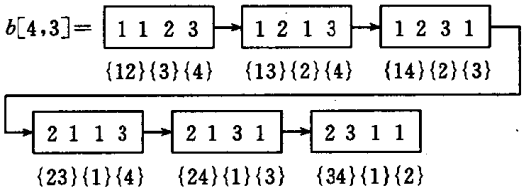



图 6-10

设 $b[i,j]$为单链表的首指针，该链表存储$\{1,2,\cdots,i\}$划分为 $j$ 个子集的方案，每个方案由一个链结点表示，结点的数据域分别指明当前方案中元素 1 至元素 $n$ 所在的子集序号。

例如，将$\{1,2,3,4\}$划分为三个互不相交的非空子集共有 $S_2(4,3)=6$ 种方式，见图 6-10。

递推从 3 个元素的集合开始。在求出$\{1,2,3\}$划分为 2 个子集的 $S_2(3,2)$个方案后，再增加元素 4，分别求$\{1,2,\cdots,4\}$划分为 2 个子集的 $S_2(4,2)$个方案，划分为 3 个子集的 $S_2(4,3)$个方案。依次类推，每增加元素 $i$ 后，分别求出$\{1,2,\cdots,i\}$划分 2 个子集、3 个子集、…、$(i-1)$个子集的方案，直至求出 $S_2(n,2),S_2(n,3),\cdots,S_2(n,n-1)$后加上 $n$ 个元素划分 1 个子集和划分 $n$ 个子集的 $2(S_2(n,1)+S_2(n,n)=2)$个方案，便是 $n$ 个方案的集合划分为不相交的非空子集的所有方案。

那么，在求出 $i-1$ 个元素的集合划分 2 个子集至 $i-1$ 个子集的基础上，如何递推 $i$ 个元素的集合划分 $j(2\leqslant j\leqslant i-1)$个子集的方案呢？

首先取出以 $b[i-1,j-1]$为首指针的链表。对链上的每个方案，元素 $i$ 放入第 $i$ 个子集，形成了 $S_2(i-1,j-1)$个方案链入以 $b[i,j]$为首指针的链表；然后取出由 $b[i-1,j]$为首指针的链表，对链上每个方案，元素 $i$ 依次放入子集 1 至子集 $j$，派生出 $j$ 个方案。这样又形成了 $J*S_2(i-1,j)$个方案，再接入 $b[i,j]$链。这样，$b[i,j]$链上的 $S_2(i,j)=S_2(i-1,j-1)+J*S_2(i-1,j)$个方案便是 $i$ 个元素的集合划为 $J$ 个子集的所有方案了。

依上述方法递推出以 $b[n,1],b[n,2],\cdots,b[n,n-1],b[n,n]$为首指针的 $n$ 条单链表。最后，一条链一条链地输出链上的每个方案。

```
program stirling_two;
  uses
    crt;
  const
    maxn            = 20;
  type
    ltype           = array[1..maxn] of byte;
    ptrtype         = ^nodetype;  { 链指针 }
    nodetype        = record      { 链结点类型 }
                        l : ltype;      { l[i]元素所在的子集序号 }
                        next : ptrtype; { 指向下一方案的指针 }
                      end;
  var
    b               : array[1..maxn,1..maxm] of ptrtype;
  { b[i,j]—单链表。存储 {1..i} 划分 j 个相交子集的所有方案 }
    n               : integer;
  procedure main;
    var i,j,k : integer;
        p,q : ptrtype;
      begin
        for i:=1 to n do
          begin
            new(b[i,1]); new(b[i,1]^.next);
            { i 个元素划分 1 个子集的方案为 {1..i} }
            b[i,1]^.next^.next:=nil;
            for j:=1 to i do b[i,1]^.next^.l[j]:=1;
            new(b[i,i]); new(b[i,i]^.next);
            { i 个元素划分 i 个子集的方案为{1}{2}..{i} }
            b[i,i]^.next^.next:=nil;
            for j:=1 to i do b[i,i]^.next^.l[j]:=j;
          end;
        for i:=3 to n do { 逐一递推 S2(i,j) i=3..n, j=2..i-1 }
          for j:=2 to i-1 do
            begin
              p:=b[i-1,j-1]^.next; new(q); b[i,j]:=q;
              { 1..i-1 划分 j-1 个子集的每个方案中，i 放入第 i 个子集 }
              repeat new(q^.next);
                     q:=q^.next;
                     q^.next:=nil;
                     q^.l:=p^.l;
                     q^.l[i]:=i;
                     p:=p^.next;
              until p=nil;
              p:=b[i-1,j]^.next;
              { 1..i-1 划分 j 个子集的每个方案中，i 分别放入子集 1..子集 j }
              repeat for k:=1 to j do
```

```
                    begin
                      new(q^.next); q:=q^.next; q^.next:=nil;
                      q^.l:=p^.l;
                      q^.l[i]:=k;
                    end;
                  p:=p^.next;
            until p=nil;
          end;
  end;

procedure show;
  var p : ptrtype;
      k,i,tot : integer;
    begin
      tot:=0;
      for k:=1 to n do begin
      { 逐一显示 S2(n,1)..S2(n,n), 并计算 Bn 数 }
        p:=b[n,k]^.next;
        repeat inc(tot); write(tot,') ');
               for i:=1 to n do
                 write(p^.l[i]:3);
               writeln;
               p:=p^.next;
        until p=nil;
      end;
    end;

begin
  clrscr; { 清屏 }
  repeat
    write('n='); readln(n); { 输入参数 n }
  until (n>0) and (n<=maxn);
  main; { 计算并输出 S2(n,1)..S2(n,n)个方案和 Bn 数 }
  show;
end.
```

## 习 题 六

1. 考虑一个 $1\times n$ 的棋盘，假定我们给棋盘的每一方块染上红或蓝色之一，对 $n=1,2,\cdots,N$ 设 $g(n)$表示没有两块染红色的方块邻接的这样染色方式的个数。求 $g(n)$递归公式。

2. $\{1,2,\cdots,n\}$的一个子集称为交替的，如果我们按上升次序列出它的元素时，它们是奇，偶，奇，偶…。例如$\{1,4,7,8\}$和$\{3,4,11\}$都是交替的。空集也算作是交替的，但$\{2,3,4,5\}$就不是，因为它开始是一个偶数。令 $f(n)$表示交替子集的数目。求 $f(n)$的递归公式。

3. 有多少个长度为 $n$ 的 0 与 1 的串，在这些串中，即不包含 010 也不包含 101？例如，

对于 $n=4$，有 10 个这样的串

| | |
|---|---|
| 0000 | 1111 |
| 0001 | 1110 |
| 0011 | 1100 |
| 0110 | 1001 |
| 0111 | 1000 |

4. 设 $f(n,k)$是从集合$\{1,2,\cdots,n\}$中能够选择的没有两个连续整数的 $k$ 元素子集的数目。求递归式 $f(n,k)$。

5. 设 $f_r(n,k)$是能够从集合$\{1,2,\cdots,n\}$中选出的，两两之差均大于 $r$ 的 $k$ 元素子集的数目。求递归式 $f_r(n,k)$。

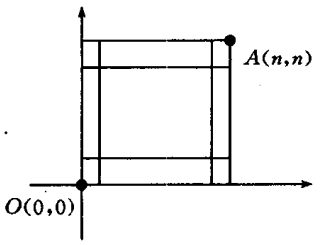
6. $2n$ 个命名的结点均匀分布在圆周上，每两点用一条弦连接起来，使得没有两条弦相交，问有多少方式？（设 $H(r)$为 $2n$ 个结点的不同连接方式数）

7. 在 $x_1x_2\cdots x_n$ 的 $n$ 个乘数中，不打乱次序，取 $k$ 个相邻乘数偶对有多少种不同方法？

8. 试计算从平面坐标(0,0)到$(n,n)$点（见题图 6-1)的：

(1) 在对角线 $OA$ 之上的递增路径的条数；

(2) 在对角线 $OA$ 之上且不触及 $OA$ 的递增路径的条数。




题图 6-1

9. 设 $P(n,t)$是一个 $n$ 元素的集合划分成 $t$ 个非空无序子集的划分数，$T(n,t)$是一个 $n$ 元素的集合划分成 $t$ 个有序子集的划分数。

分别求 $P(n,t)$和 $T(n,t)$的递归关系式。

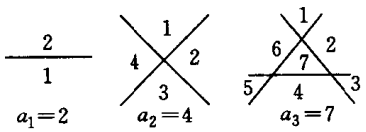
10. 设 $f(n,k)$表示从$\{1,2,\cdots,n-1\}$中取$(n-k)$个不同整数的积之和数。例如，所有从$\{1,2,\cdots,4\}$中取 2 个不同整数的积之和是

$$1\times 2+1\times 3+1\times 4+2\times 3+2\times 4+3\times 4=35$$

求 $f(n,k)$。

11. 设 $a_n$ 是将整数 1 至 $n$ 排成一行的方法数，使其服从这样的条件，除去第一个数外，每个数与它左边的某个数恰好相差 1。找出一个关于 $a_n$ 的递推公式。

12. 平面上有 $n$ 条直线，如果没有两条是平行的，且没有三点是共点的，则称它们处于一般状态。设 $a_n$ 是 $n$ 条处于一般状态的直线分割平面所成的区域数。显然 $a_1=2$，$a_2=4$，$a_3=7$，见题图 6-2。




题图 6-2

试求 $a_n$ 的递推式。

13. 球面上有 $n$ 个大圆，它们没有三个大圆通过同一点。设 $a_n$ 表示这些大圆所形成的

区域数。试求递归式 $a_n$。

14. 设 $h_n$ 表示 $n+2$ 条边的凸多边形为它的对角线划分所得的区域数，其中假定没有三条对角线在凸多边形内有一公共点。定义 $h(0)=0$，求递归式 $h(n)$。

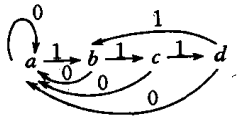
15. 在一个圆的圆周上放置 $n$ 个不同的点，并画出所有可能的通过任意两点的弦，假定这些弦中没有三条在圆内是交于同一点的。令 $a_n$ 表示在圆内形成的区域数。求递推式 $a_n$。

16. 某人有 $n$ 元钱，他每天买一次物品，每次买物品的品种很单调，或者买一元钱的甲物品，或者买二元钱的乙物品，或者买二元钱的丙物品。求他花完 $n$ 元钱的方式数 $f(n)$。

17. 假设将正整数 $n$ 进行划分，使得每块小于或等于 $m$ 的剖分数为 $P(n,m)$。求递归式 $P(n,m)$。

18. 一个质点在水平方向上运动，每秒钟它走过的距离等于它前一秒钟走过距离的两倍。设质点的初始位置为 3，并设第一秒钟走了一个单位长的距离，求第 $r$ 秒钟质点的位置 $f(r)$。

19. 在题图 6-3 上求从顶点 $a$ 起到顶点 $d$ 止的长度为 $n$ 的有向路径的数目。




题图 6-3

# 第七章　Pólya 原理

从组合数学的角度看，对于一个问题的各种不同的组合状态计数，有两个方面的难点，一个是按什么观点来划分组合状态的异同，另一个是如何计算全部互异的组合状态的个数。

[例 1]　对 $N\times N$ 方阵用黑白二种颜色涂色，定义一个正整数 $k(0\leqslant k\leqslant n^2)$，表示方阵中涂黑色的小方块不超过 $k$ 个，问能得到多少种不同的图象？经过转换和翻转使之吻合的两种方案，算是同一种方案。

显然，经过旋转或翻转后吻合的两种图象被认作是相同的，无论怎样旋转或翻转都不能吻合的两种图象被认作是互异的。

由于 $N\times N$ 方阵的对称性，旋转方式有绕图的中心点按反时针方向旋转 0°，90°，180°，270°四种；翻转方式有上下翻转、左右翻转、沿对角线翻转三种。

例如，$N=2,k=4$ 时，对于图 7-1 中的 4 个图形，$C_1$ 旋转 90°得出 $C_2$，$C_2$ 旋转 90°得出 $C_3$，$C_3$ 旋转 90°得出 $C_4$，因此这四个图象被认作是相同的一种方案。同样，我们也可以得出，任何经旋转翻转得出的图象与旋转翻转前的图象也同属一种方案。

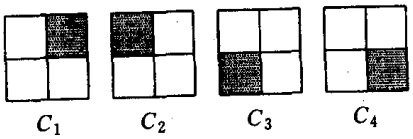



图　7-1

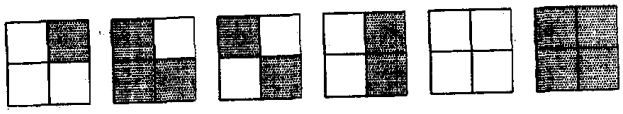


图　7-2

而图 7-2 的 6 个图象中，在两个图象之间都不能通过旋转翻转而吻合，因此它们是互异的。细心的读者可以判断出，图 7-2 中的六个图象正是 $N=2,k=4$ 时问题的解。

如果 $N$ 和 $k$ 值任意，颜色数增加，不同着色方案的计数问题就不能凭借直觉和经验解决，必须借助一种成熟的数学工具——这就是 Pólya 原理。

在介绍 Pólya 原理之前，我们先讲述几个基本概念，它们是 Pólya 原理的基础。

## 7.1　等价关系、群、置换群

### 一、等价关系

#### 1. 加氏积和二元关系

如 $A$ 是一个集合，$A$ 中元素的有序对的全体记为

$$A\times A=\{(a,b)\mid a\in A,b\in A\}$$

称为 $A$ 上的加氏积。

对于 $A$ 中任一有序对的子集，确定了一个二元关系 $R$，$R$ 中任意有序对 $(a,b)$ 可记作

$(a,b)\in R$，不在 $R$ 中的任一有序对$(a,b)$可记作$(a,b)\notin R$。显然，$A$ 中的任一有序对$(a,b)$，或者$(a,b)\in R$，或者$(a,b)\notin R$，二者必居其一。

二元关系除了可用有序对集合的形式表达外，亦可用关系矩阵和关系图形表示。设两个有限集合 $X=\{X_1,X_2,\cdots,X_m\}$，$Y=\{Y_1,Y_2,\cdots,Y_n\}$，$R$ 为 $X$ 到 $Y$ 的一个二元关系，则对应于关系 $R$ 有一个关系矩阵 $M_r=[Y]m\times n$

$$Y_{ij}=\begin{cases}1 & (X_i,Y_j)\in R\\ 0 & (X_i,Y_j)\notin R\end{cases}$$

设 $X=\{X_1,X_2,\cdots,X_4\}$，$Y=\{Y_1,Y_2,\cdots,Y_3\}$

$R=\{(X_1\,Y_1)(X_1\,Y_3)(X_2\,Y_2)(X_2\,Y_3)(X_3\,Y_1)(X_4\,Y_1)(X_4\,Y_2)$，则关系矩阵为

$$M_r=\begin{array}{c}\\X_1\\X_2\\X_3\\X_4\end{array}\begin{array}{c}\begin{array}{ccc}Y_1 & Y_2 & Y_3\end{array}\\\begin{bmatrix}1 & 0 & 1\\0 & 1 & 1\\1 & 0 & 0\\1 & 1 & 0\end{bmatrix}\end{array}$$

$X_1,X_2,\cdots,X_m$，$Y_1,Y_2,\cdots,Y_n$ 作图的顶点，若$(X_i,Y_j)\in R$，则自 $X_i$ 至 $Y_j$ 处作一条有向弧；若$(X_i,Y_j)\notin R$，则 $X_i$ 和 $Y_j$ 之间未有弧连接，得出 $R$ 的关系图。例如对应上述关系 $R$ 的有向图如图 7-3 所示。

**2. 等价关系**

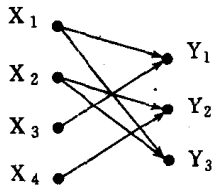


图 7-3

所谓等价关系，是指满足下列三种性质的关系 $R$：

(1) 对于所有 $a\in A$，均有$(a,a)\in R$(自反性)；

(2) 对于所有 $a,b\in A$，当$(a,b)\in R$ 时，恒有$(b,a)\in R$(对称性)；

(3) 对于所有 $a,b,c\in A$，当$(a,b)\in R$ 且$(b,c)\in R$ 时，恒有$(a,c)\in R$(传递性)。

集合的等价关系与集合的划分有深刻的联系。常把相互等价的元素归入一个子集，也就是等价类。记元素 $a$ 所在的等价类为

$[a]_R=\{x|x\in A,(a,x)\in$等价关系 $R\}$

则$[a]_R$ 是 $A$ 的一个非空子集，因为至少有$(a,a)\in R$，显然$a\in[a]_R$。一个等价类可以取其中任何一个元素作为一类元素的代表，如：$(b,c)\in[a]_R$，则$(b,a)\in R$ 且$(c,a)\in R$，$[b]_R=[c]_R=[a]_R$。

一个由等价关系构成的集合是对集合 $A$ 的一种划分。即$\bigcup\{[a]_R|a\in A\}=A$，并且要末$[a]_R=[b]_R$，要末$[a]_R\cap[b]_R=\varphi$，此即表明 $R$ 关系决定了 $A$ 的一个划分$\{[a]_R|a\in A\}$。

反之设 $Aa(Aa=\{[a]_R|a\in A\})$是 $A$ 的一个划分，则$\{Aa\}$决定了一个等价关系。这只要规定同一子集 $Aa$ 内各元素等价即可。

例如第七章前言里的[例 1]中，设 $n=2,k=4$ 时，显然共有 16 种方式如图 7-4 所示。

如果我们将 4 个小方块“钉死”于平面上、下、左、右 4 个绝对位置，那末图 7-4 确实显示了这 16 种“不同”方式。但从本质上看，一些看似不同的方式，例如 2 个相邻块白，另 2

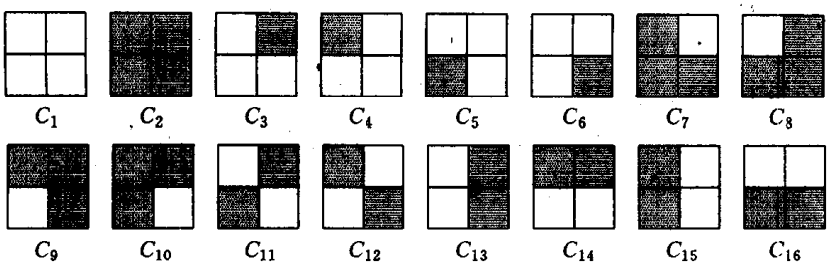



图 7-4

个相邻块黑的 $C_{13}$,$C_{14}$,$C_{15}$,$C_{16}$“宏观”上可以说是相同的,即属于同一个“类”。这样就需要在集合 $A=\{C_1,C_2,\cdots,C_{16}\}$ 上确定一种方式:设想正方形可以绕正方形中心作 0°,90°,180°,270°反时针旋转,如涂色方式 $C_i$ 可由涂色方式 $C_j$ 用上述某种旋转获得,则说 $C_i$,$C_j$ 有关系 $R$,而关系 $R$ 是 $A$ 上的等价关系。因为 $R$ 满足:

(自反性) $C_i$ 旋转 0°得 $C_i$;

(对称性) $C_i$ 旋转 $X$°得到 $C_j$,则 $C_j$ 旋转(360−$X$)°亦可得到 $C_i$;

(传递性) $C_i$ 旋转 $X$°得到 $C_j$,$C_j$旋转 $Y$°得到 $C_k$,则 $C_i$ 旋转$(x+y)$°mod 360°得到 $C_k$。

$$(x,y=0°,90°,180°,270°)$$

我们将互相等价的图象归入一类,由此得出 6 个等价类:

$$[C_1]_R=\{C_1\}$$
$$[C_2]_R=\{C_2\}$$
$$[C_3]_R=\{C_3,C_4,C_5,C_6\}$$
$$[C_7]_R=\{C_7,C_8,C_9,C_{10}\}$$
$$[C_{11}]_R=\{C_{11},C_{12}\}$$
$$[C_{13}]_R=\{C_{13},C_{14},C_{15},C_{16}\}$$

显然,这 6 个等价类组成 $A$ 的一个划分。习惯上,常把由等价关系所划分出来的 $A$ 的等价类的全体记为 $\overline{A}$。$\overline{A}=\{[a]_R|a\in A\}$称为关于等价关系 $R$ 商集,记为 $\overline{A}=A/R$。

为了计算各种等价类的数目,我们必须引出群的概念。

## 二、群和置换群

### 1. 群的定义

带有一种运算 * 的集合 $G$ 如满足以下四个性质:

(1) 封闭性:对于任两个元素 $a,b\in G$,则有 $a*b\in G$;

(2) 结合性:对于任三个元素 $a,b,c\in G$,则有 $a*(b*c)=(a*b)*c$;

(3) 单位元 $e$ 存在:$G$ 中存在一个元素 $e$,对于 $G$ 中任何元素 $X$,满足 $e*X=X*e=X$;

(4) 逆元存在:对 $G$ 中每一个元素 $X$,恒有一个 $Y\in G$,使得 $X*Y=Y*X=e$;

则称 $G$ 是对于 * 运算的群,简记$<G,*>$。若 $G$ 中元素个数有限,则称$<G,*>$为

有限群，$|G|$为该有限群的阶数。

[例 1] 设$G=\{0°,90°,180°,270°\}$表示在平面上几何图形绕形心逆时针旋转角度的 4 种可能情况。设☆是$G$的二元运算。对于$G$中任意两个元素$a,b$，$a$☆$b$表示平面图形连续旋转$a$和$b$得到的总旋转角度，并规定旋转 360°等于原来的状态，就看作没有经过旋转。其状态示于表 7-1。验证$<G,$☆$>$是一个群。

表 7-1

| ☆ | 0° | 90° | 180° | 270° |
|---|---|---|---|---|
| 0 | 0° | 90° | 180° | 270° |
| 90° | 90° | 180° | 270° | 0° |
| 180° | 180° | 270° | 0° | 90° |
| 270° | 270° | 0° | 90° | 180° |

由表 7-1 可见，运算☆在$G$上是封闭的；

对于任意$a,b,c\in R$，$(a$☆$b)$☆$c$表示图形依次旋转$a,b$和$c$，而$a$☆$(b$☆$c)$表示将图形依次旋转$b$、$c$和$a$，而总的旋转度数都等于$a+b+c(\mathrm{mod}360°)$。因此$(a$☆$b)$☆$c=a$☆$(b$☆$c)$；

0°是单位元$e$；

90°、180°、270°的逆元分别是 270°、180°、90°。

因此$(G,$☆$)$是一个群。

由表 7-1 还可看出，群的运算表没有两行(或两列)是相同的。它的每一行或列都是$G$到$G$上的一一变换。例如第二行将 0°变换成 90°，将 90°变换成 180°，将 180°变换成 270°，将 270°变换成 0°；……这种有限集$G$到$G$的一一变换，称为一个置换。当$|G|=n$时，也称为一个$n$元置换。

### 2. 置换群

下面，我们着重讨论群的一种类型——置换群，因为它是 Pólya 原理的基础。

有$n$个元素$1,2,\cdots,n$作某种置换，如元素 1 用$1,2,\cdots,n$中某个数$a_1$取代，2 被$1,2,\cdots,n$中除$a_1$外的某个数$a_2$取代，……，$n$被$1,2,\cdots,n$中除$a_1,a_2,\cdots,a_{n-1}$外的某个数$a_n$取代，表以

$$\begin{pmatrix}1 & 2 & \cdots & n\\ a_1 & a_2 & \cdots & a_n\end{pmatrix}$$

由此看出不同的置换共有$n!$个。

例如：

$$P_1=\begin{pmatrix}1 & 2 & 3 & 4\\ 3 & 1 & 2 & 4\end{pmatrix},\ P_2=\begin{pmatrix}1 & 2 & 3 & 4\\ 4 & 3 & 2 & 1\end{pmatrix}$$

$$P_1P_2=\begin{pmatrix}1 & 2 & 3 & 4\\ 3 & 1 & 2 & 4\end{pmatrix}\begin{pmatrix}1 & 2 & 3 & 4\\ 4 & 3 & 2 & 1\end{pmatrix}$$

$$=\begin{pmatrix}1&2&3&4\\3&1&2&4\end{pmatrix}\begin{pmatrix}3&1&2&4\\2&4&3&1\end{pmatrix}=\begin{pmatrix}1&2&3&4\\2&4&3&1\end{pmatrix}$$

表示先作 $PS_1$ 的置换后再作 $PS_2$ 的置换。如

$$1\xrightarrow{P_1}3\xrightarrow{P_2}2,\ 2\xrightarrow{P_1}1\xrightarrow{P_2}4,\ 3\xrightarrow{P_1}2\xrightarrow{P_2}3,\ 4\xrightarrow{P_1}4\xrightarrow{P_2}1$$

由此得出 $P_1P_2=\begin{pmatrix}1&2&3&4\\2&4&3&1\end{pmatrix}$

一般称 $P_1P_2$ 为置换乘法。

类似有 $P_2P_1=\begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix}\begin{pmatrix}1&2&3&4\\3&1&2&4\end{pmatrix}$

$$=\begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix}\begin{pmatrix}4&3&2&1\\4&2&1&3\end{pmatrix}=\begin{pmatrix}1&2&3&4\\4&2&1&3\end{pmatrix}$$

可见，$P_1P_2\neq P_2P_1$

置换常用一种称为循环的工具来简化表达，并且能反映置换的结构，还便于运算。因此有：对 $n$ 个元素进行置换，如将 1 至 $n$ 中的 $m(1\leqslant m\leqslant n)$ 个数 $a_1,a_2,\cdots,a_m$ 轮流替换，即

$$\begin{pmatrix}a_1 & a_2\cdots a_{m-1} & a_m & a_{m+1}\cdots a_n\\ a_2 & a_3\cdots a_m & a_1 & a_{m+1}\cdots a_n\end{pmatrix}$$

并且保持其余 $n-m$ 个数不变，那么该置换称为 $m$ 阶循环，记作：

$$(a_1a_2\cdots a_m)=\begin{pmatrix}a_1\cdots a_m & a_{m+1}\cdots a_n\\ a_2\cdots a_1 & a_{m+1}\cdots a_n\end{pmatrix}$$

特别是 $m=2$ 时，两阶循环 $(i,j)$ 叫做 $i$ 和 $j$ 的对换。

注意：在循环中不出现的数蕴含着它是不动的。当所有数都不动时，就是单位元 $e$。即 $e=(1)(2)\cdots(n)=(1)=(2)=\cdots=(n)$

例如　对 1,2,3,4,5 置换

$$\begin{pmatrix}1&2&3&4&5\\4&3&1&5&2\end{pmatrix}=(1\ \ 4\ \ 5\ \ 2\ \ 3)$$

$$\begin{pmatrix}1&4&5\\5&1&4\end{pmatrix}=(1\ \ 5\ \ 4)\underbrace{(2)\ (3)}_{\text{2 和 3 保持不变}}=(1\ \ 5\ \ 4)$$

$$\begin{pmatrix}1&2&3&4&5\\3&1&2&5&4\end{pmatrix}=(1\ \ 3\ \ 2)(4\ \ 5)$$

如若两个循环 $(a_1a_2\cdots a_n)$ 和 $(b_1b_2\cdots b_m)$ 没有相同文字，则称为不相交的。不相交的两循环的乘积可交换。例如：

$$(1\ \ 3\ \ 2)(4\ \ 5)=(4\ \ 5)(1\ \ 3\ \ 2)$$

若 $P=(a_1a_2\cdots a_n)$ 则 $P^n=(1)(2)\cdots(n)=e$。例如

$$P=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}=(1\ \ 2\ \ 3)$$

$$P^2=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}=\begin{pmatrix}1&2&3\\3&1&2\end{pmatrix}=(1\ \ 3\ \ 2)$$

$$P^3=\begin{pmatrix}1&2&3\\3&1&2\end{pmatrix}\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}=(1)(2)(3)$$

集合$\{1,2,\cdots,n\}$的置换合体，在置换乘法下构成群，称为$n$个文字的置换群，简称$n$次对称群，记为$(S_n,*)$. 显然$|S_n|=n!$

[**例 2**] 对等边三角形$ABC$(见图 7-5)：

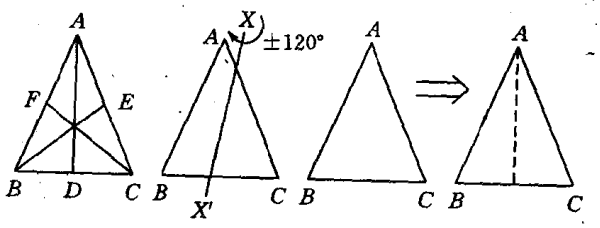



图 7-5

(*a*)绕过中心的$XX'$，沿反时针方向分别旋转 0°，120°，240°；

(*b*)沿三角形$ABC$的中线$AD$、$BE$、$CF$翻转 180°。

经(*a*)，(*b*)的变换，$\triangle ABC$重合于本身，但顶点换了位置。

证明在旋转和翻转运算下的顶点位置的置换构成一个群。

证：$A,B,C$分别代以 1，2，3，可得顶点间的置换如下：

绕$XX'$转 0°的置换为$P_1$，绕$XX'$转 240°的置换为$P_2$。

$$P_1=\begin{pmatrix}1&2&3\\1&2&3\end{pmatrix}=(1)$$

$$P_2=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}=(1\quad 2\quad 3)$$

绕$XX'$转 120°的置换

$$P_3=\begin{pmatrix}1&2&3\\3&1&2\end{pmatrix}=(1\quad 3\quad 2)$$

沿$AD$中翻 180°的置换

$$P_4=\begin{pmatrix}1&2&3\\1&3&2\end{pmatrix}=(2\quad 3)$$

沿$BE$中翻 180°的置换

$$P_5=\begin{pmatrix}1&2&3\\3&2&1\end{pmatrix}=(1\quad 3)$$

沿$CF$中翻 180°的置换

$$P_6=\begin{pmatrix}1&2&3\\2&1&3\end{pmatrix}=(1\quad 2)$$

显然，$P_1,P_2\cdots P_6$为$\{1,2,3\}$置换的全体$S_3$。可以证明$S_3$的这 6 个置换是一个群，见表7-2。

表 7-2

| $P_i \backslash P_j$ | (1) | (123) | (132) | (23) | (13) | (12) |
|---|---|---|---|---|---|---|
| (1) | (1) | (123) | (132) | (23) | (13) | (12) |
| (123) | (123) | (132) | (1) | (13) | (12) | (23) |
| (132) | (132) | (1) | (123) | (12) | (23) | (13) |
| (23) | (23) | (12) | (13) | (1) | (132) | (123) |
| (13) | (13) | (23) | (12) | (123) | (1) | (132) |
| (12) | (12) | (13) | (23) | (132) | (123) | (1) |

从上表可以看出封闭性、结合性、单位元和逆元存在四个条件均满足，故构成一个群。而且从

$$P_2P_5=\begin{pmatrix}1&2&3\\2&3&1\end{pmatrix}\begin{pmatrix}1&2&3\\3&2&1\end{pmatrix}=\begin{pmatrix}1&2&3\\2&1&3\end{pmatrix}=P_6$$

可知，$ABC$ 绕中心轴 $XX'$ 旋转 240°接着绕 $BE$ 翻 180°的结果相当于绕 $CF$ 中线翻转 180°，见图 7-6。

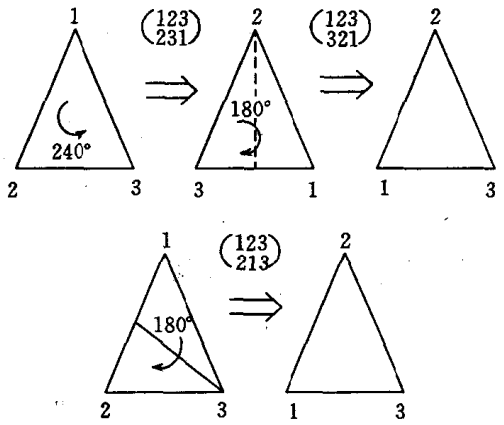



图 7-6

任何一个循环，都可以表达成若干换位之积。但是表达形式不尽统一，甚至连换位个数都不相同。例如

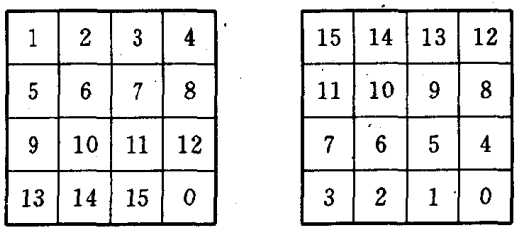
$$(123)=(13)(12)=(23)(31)=(21)(23)=(12)(13)(31)(13)$$

尽管如此，有一个性质却是固有的，它不依换位的个数不同而异，那就是循环分解成换位的乘积时，换位个数奇偶性是不变的，或分解成奇数个换位之积或分解成偶数个换位之积。若一个置换分解成奇数个换位之积，叫做奇置换；若分解成偶数个换位之积叫偶置换。单位置换为偶置换。

显然(1)(123)是偶置换。

[**例 3**] 有一个 4×4 的初始棋局(如图 7-7(*a*))，1，2，…，15 标明棋子序号，0 表示空格。

与空格相邻的棋子与空格互换位置可得另一个棋局。问从初始棋局出发，依据上述规则能否走到图 7-7(*b*)的棋局？



| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 0 |

初始棋局

(*a*)

| 15 | 14 | 13 | 12 |
|---|---|---|---|
| 11 | 10 | 9 | 8 |
| 7 | 6 | 5 | 4 |
| 3 | 2 | 1 | 0 |

终止布局

(*b*)

图 7-7

**解**：与空格相邻的棋子与空格互换位置得到的新棋局，可以看作是某布局的棋子间的一次置换。有些布局是由初始布局经偶数次换位而得，有些则是经奇数次换位而得。由奇数次换位得到的布局不可能通过偶数次换位而得到。

从初始布局和终止布局可以看出，空格(*a*)从右下角出发返回右下角，必须通过偶数次换位才能实现，即向左向右换位的次数必须相同，向上向下换位的次数也必须相同。而实际上，从初始布局到终止布局相当于作下面的置换。

$$P=\begin{pmatrix}0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15\\ 0 & 15 & 14 & 13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1\end{pmatrix}$$

$$=(0)(1\ 15)(2\ 14)(3\ 13)(4\ 12)(5\ 11)(6\ 10)(7\ 9)(8)$$

$P$ 是个奇置换。显然，图 7-7(*a*)的棋局无论如何不可能通过空格与其相邻棋子换位而达到图 7-7(*b*)的棋局。

## 7.2 Burnside 引理

### 一、共轭类

为了使读者对置换群结构有感性认识，我们先给出{1,2,3,4}置换的全体

$$\begin{aligned}S_4=\{&(1)(2)(3)(4),\\&(12),(13),(14),(23),(24),(34)\\&(123),(124),(132),(134),(142),(143),(234),(243)\\&(1234),(1243),(1324),(1342),(1423),(1432)\\&(12)(34),(13)(24),(14)(23)\}\end{aligned}$$

一般可把 $S_n$ 中任一个置换分解成若干互不相交的循环乘积

$$P=\underbrace{(a_1a_2\cdots a_{k1})(b_1b_2\cdots b_{k2})\cdots(h_1h_2\cdots h_{kl})}_{L\text{ 项}}$$

其中 $k_1+k_2+\cdots+k_L=n$。设其中 $k$ 阶循环出现的次数为 $C_k$，$k=1,2,\cdots n$。$k$ 阶循环出现 $C_k$ 次，用 $(k)^{C_k}$ 表示。

$S_n$ 中的置换可按分解

$(1)^{C_1}(2)^{C_2}\cdots(n)^{C_n}$

的不同而分类。$C_i=0$ 的项$(i)^{c_i}$ 可省略$(i=1,2,\cdots,n)$。显然对每一类格式有

$$\sum_{k=1}^{N} kC_k = n$$

例如 $S_4$ 中，具有$(1)^0(2)^2(3)^0(4)^0$ 格式的置换有 3 个，即

(12)(34),(13)(24),(14)(23);

$S_4$ 中具有$(1)^1(3)^1$ 格式中的置换有 8 个，即

(123),(124),(132),(134),(142),(143),(234),(243);

$S_4$ 中具有$(1)^2(2)^1$ 格式的置换有 6 个，即

(12),(13),(14),(23),(24),(34);

$S_4$ 中具有$(1)^4$ 格式的置换有 1 个，即

(1)(2)(3)(4)

$S_4$ 中具有$(4)^1$ 格式的置换有 6 个，即

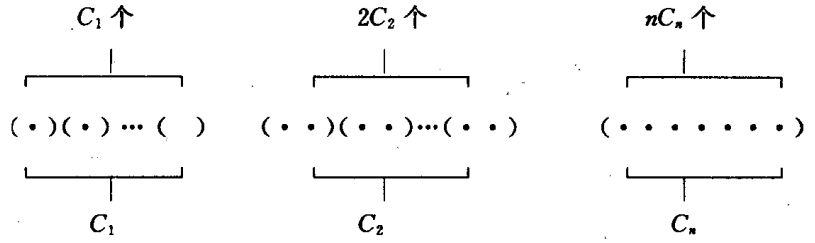
(1234),(1243),(1324),(1342),(1423),(1432)

由此引出共轭类的概念：

在 $S_n$ 中具有相同格式的置换的全体叫做与该格式相应的共轭类。

下面，我们来分析一下，具有相同格式的共轭类中含多少个置换：

设$(1)^{C_1}(2)^{C_2}\cdots(n)^{C_n}$的格式为




1,2,…,$n$ 的全排列共 $n!$ 个，每个排列依顺序填入上述格式，可得属于该共轭类的一个置换；反过来该共轭类的每个置换也可以通过这样而得到。然而由此所得的 $n!$ 个置换中有重复现象。这是因为

(1) 由循环$(a_1\cdots a_k)=(a_2\cdots a_k a_1)\cdots=(a_k a_1\cdots a_{k-1})$引起重复。1 个 $k$ 阶循环可重复 $k$ 次，…$C_k$ 个 $k$ 阶循环可重复 $k^{c_k}$次；

(2) 由互不相交的 $C_k$ 个 $k$ 阶循环乘积的可交换性引起重复。例如

$$(a_1a_2)(a_3a_4)(a_5a_6) = (a_1a_2)(a_5a_6)(a_3a_4) = (a_3a_4)(a_1a_2)(a_5a_6)$$
$$=(a_3a_4)(a_5a_6)(a_1a_2) = (a_5a_6)(a_1a_2)(a_3a_4) = (a_5a_6)(a_3a_4)(a_1a_2)$$

共重复 3! =6 次

一般地，$C_k$ 个 $k$ 阶循环重复了 $C_k!$ 次。

所以属于共轭类$(1)^{C_1}(2)^{C_2}\cdots(n)^{C_n}$格式的置换个数为

$$\frac{n!}{C_1!\ \cdots C_n!\quad 1^{C_1}\cdot 2^{C_2}\cdot n^{C_n}}$$

用上述计数公式计算 $S_4$ 中各共轭类中的置换个数：

$(2)^2$ 共轭类有 $\frac{4!}{2!\ 2^2}=3$ 个置换

$(1)^1(3)^1$ 共轭类有 $\frac{4!}{1!\ 3}=8$ 个置换

$(1)^2(2)^1$ 共轭类有 $\frac{4!}{2!\ 2}=6$ 个置换

$(1)^4$ 共轭类有 $\frac{4!}{4!}=1$ 个置换

$(4)^1$ 共轭类有 $\frac{4!}{4}=6$ 个置换

显然结果与前面分析一致。

## 二、$k$ 不动置换类和等价类

### 1. $k$ 不动置换类

设 $G$ 是 $1,2,\cdots,n$ 的置换群，当然 $G$ 是 $S_n$ 的一个子群。若 $k$ 是 $1,2,\cdots,n$ 中某个整数，$G$ 中使 $k$ 保持不变的置换的全体，记以 $Z_k$，叫做 $G$ 中使 $k$ 保持不动的置换类，简称 $k$ 不动置换类。

例如 $G=\{e,(12),(34),(12)(34)\}$，则

$$Z_1=\{e,(34)\}$$
$$Z_2=\{e,(34)\}$$
$$Z_3=\{e,(12)\}$$
$$Z_4=\{e,(12)\}$$

显然 $Z_k$ 是 $G$ 的一个子群。

### 2. 等价类

再看 $G=\{e,(12),(34),(12)(34)\}$。在群 $G$ 的作用下，数 1 变成 2，2 变成 1；3 变成 4，4 变成 3。因此 1，2 属于同一类，而 3，4 属于另一类。1 或 2 不能在群 $G$ 的作用下变成 3 或 4；同样 3 或 4 不能变成 1 或 2。$k$ 所属的等价类可以看作是 $k$ 在 $G$ 作用下的轨迹，记作 $E_k$。

数学家经反复论证，得出一个结论：

对于数 $k(1\leqslant k\leqslant n)$，关于 $n$ 个文字的置换群 $G$ 有对应的等价类 $E_k$ 和不动置换类 $Z_k$，而

$$|E_k|\cdot|Z_k|=|G|\quad k=1,2,\cdots,n$$

限于篇幅，我们省略了上述原理的证明。

例如 $G=\{e,(12),(34),(12)(34)\}$

$E_1=E_2=\{1,2\}$，$E_3=E_4=\{3,4\}$，$|E_1|=|E_2|=|E_3|=|E_4|=2$；$Z_1=Z_2=\{e,(34)\}$，$Z_3=Z_4=\{e,(12)\}$，$|Z_1|=|Z_2|=|Z_3|=|Z_4|=2$；

显然 $|E_1|\cdot|Z_1|=|E_2|\cdot|Z_2|=|E_3|\cdot|Z_3|=|E_4|\cdot|Z_4|=4=|G|$

再如 $S_4$ 中偶置换的全体 $A_4=\{(1)(2)(3)(4),(123),(124),(132),(134),(142),(143),(234),(243),(12)(34),(13)(24),(14)(23)\}$

$$E_1=\{1,2,3,4\}$$

$$Z_1=\{e_1,(234),(243)\}$$

$$|E_1|\cdot|Z_1|=4\times 3=12=|A_4|$$

具体地讲：

$A_4$ 存在单位元 $P_1=e$，使 $1\xrightarrow{e}1$；存在置换 $P_2=(12)(34)$，使 $1\xrightarrow{P_2}2$；存在置换 $P_3=(13)(24)$，使 $1\xrightarrow{P_3}3$；存在置换 $P_4=(14)(23)$，使 $1\xrightarrow{P_4}4$；

$$Z_1P_1=\{e,(243),(243)\}$$

$$Z_1P_2=\{(12)(34),(124),(123)\}$$

$$Z_1P_3=\{(13)(24),(132),(134)\}$$

$$+)\ Z_1P_4=\{(14)(23),(143),(142)\}$$

$$Z_1P_1\dagger\ Z_1P_2\dagger\ Z_1P_3\dagger\ Z_1P_4=A_4$$

## 三、Burnside 引理

设 $G=\{a_1,a_2,\cdots,a_g\}$，$a_1,a_2,\cdots,a_g$ 是 $N=\{1,2,\cdots,n\}$ 上的置换群，其中 $a_1=e$。下面我们来分析一下，$G$ 在 $N$ 上究竟可引出多少个不同的等价类。

我们把 $N$ 上的每一个置换 $a_k(k=1,2,\cdots,g)$ 分解成不相交的循环的乘积，记 $C_1(a_k)$ 为置换 $a_k$ 中 1 阶循环的个数，即 $a_k$ 作用下保持不变的元素的个数，作出表 7-3。

表 7-3

| $a_j\backslash S_{jk}\backslash k$ | 1 | 2 | 3 | …… | $n$ | $C_1(a_j)$ |
|---|---|---|---|---|---|---|
| $a_1$ | $S_{11}$ | $S_{12}$ | $S_{13}$ | …… | $S_{1n}$ | $C_1(a_1)$ |
| $a_2$ | $S_{21}$ | $S_{22}$ | $S_{23}$ | …… | $S_{2n}$ | $C_1(a_2)$ |
| | | | | …… | | |
| | | | | …… | | |
| | | | | …… | | |
| | | | | …… | | |
| | | | | …… | | |
| | | | | …… | | |
| $a_g$ | $S_{g1}$ | $S_{g2}$ | $S_{g3}$ | …… | $S_{gn}$ | $C_1(a_g)$ |
| $Z_k$ | $\lvert Z_1\rvert$ | $\lvert Z_2\rvert$ | $\lvert Z_3\rvert$ | …… | $\lvert Z_n\rvert$ | $\sum_{k=1}^{n}\lvert Z_k\rvert$ $=\sum_{j=1}^{s}C_1(a_j)$ |

其中

$$S_{jk}=\begin{cases}0 & \text{若 } a_j \notin Z_k\text{，即 } k \xrightarrow{a_j \notin Z_k} L(L \neq k)\\ 1 & \text{若 } a_j \in Z_k\text{，即 } k \xrightarrow{a_j \in Z_k} k\end{cases}$$

因为 $\sum_{k=1}^{n} S_{jk}=C_1(a_j)$；$\sum_{j=1}^{S} S_{jk}=|Z_k|$；所以 $\sum_{j=1}^{S}\sum_{k=1}^{n} S_{jk}=\sum_{j=1}^{S} C_1(a_j)=\sum_{k=1}^{n}|Z_k|$。

若 $N=\{1,2,\cdots,n\}$分解成 $L$ 个等价类，$N=E_1+E_2+\cdots+E_L$，而且当 $J$ 和 $k$ 属于同一等价类时，则 $|Z_j|=|Z_k|$。所以

$$\sum_{k=1}^{n}|Z_k|=\sum_{i=1}^{L}\sum_{k\in E_i}|Z_k|=\sum_{i=1}^{L}|E_1||Z_i|=L\cdot|G|$$

(因为 $|E_i|\cdot|Z_i|=|G|, i=1,2,\cdots,L$)

由此得出 $L=\frac{1}{|G|}\sum_{k=1}^{n}|Z_k|=\frac{1}{|G|}[C_1(a_1)+C_1(a_2)+\cdots+C_1(a_g)]$

即置换群 $a_1\cdots a_g$ 在 $N$ 上可引出不同等价类的数目为所有置换中保持不动的元素个数的总和对 $|G|$ 取平均值。

这就是所谓的 Burnside 引理。

例如 $G=\{e,(12),(34),(12)(34)\}$，对应的写在表 7-4 中。

**表 7-4**

| $a_j\backslash S_{jk}\backslash k$ | 1 | 2 | 3 | 4 | $C_1(a_j)$ | 格式 |
|---|---|---|---|---|---|---|
| (1)(2)(3)(4) | 1 | 1 | 1 | 1 | 4 | $(1)^4(2)^0(3)^0(4)^0$ |
| (12)(3)(4) | 0 | 0 | 1 | 1 | 2 | $(1)^2(2)^1(3)^0(4)^0$ |
| (1)(2)(34) | 1 | 1 | 0 | 0 | 2 | $(1)^2(2)^1(3)^0(4)^0$ |
| (12)(34) | 0 | 0 | 0 | 0 | 0 | $(1)^0(2)^2(3)^0(4)^0$ |
| $Z_k$ | 2 | 2 | 2 | 2 | 8 | |

不同的等价类的个数为 $\frac{4+2+2+0}{4}=2$，即{1,2}和{3,4}

**[例 1]** 正方形均分成 4 个格子，用两种颜色对 4 个格子着色，问能得到多少不同的图象？

**解**：由于每格有两种颜色可供选择，故有 16 种可能方案，由图 7-4 给出。

当每个绕中心点的轴按反时针旋转 90°，180°，270°时得 16 种图象的又一种排列，可以看作是 16 种图象的一种置换，而左右翻转、上下翻转和沿对角线翻转的所有情况都可通过旋转得到，因此我们只考虑旋转 0°，90°，180°，270°四种置换，即 $|G|=4$。

(1) 旋转 0°为不动置换

$$P_1=\underline{(C_1)(C_2)(C_3)(C_4)(C_5)\cdots(C_{16})}$$

即 $C_1(P_1)=16$

(2) 旋转 90°

$$P_2=\underline{(C_1)(C_2)}(C_3C_4C_5C_6)(C_7C_8C_9C_{10})(C_{11}C_{12})(C_{13}C_{14}C_{15}C_{16})$$

即 $C_1(P_2)=2$

(3) 旋转 180°

$$P_3=\underline{(C_1)(C_2)}(C_3C_5)(C_4C_6)(C_7C_9)(C_8C_{10})\underline{(C_{11})(C_{12})}(C_{13}C_{15})(C_{14}C_{16})$$

即 $C_1(P_3)=4$

(4) 旋转 270°

$$P_4=\underline{(C_1)(C_2)}(C_3C_4C_5C_6)(C_7C_8C_9C_{10})(C_{11}C_{12})(C_{13}C_{14}C_{15}C_{16})$$

即 $C_1(P_4)=2$

不同等价类个数为

$$L=\frac{1}{4}(16+2+4+2)=6$$

对应的 6 个不同图象如图 7-8 所示。

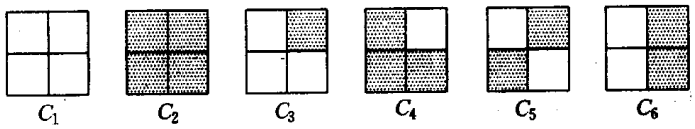



图 7-8

## 7.3 Pólya 定理

由上可知，Burnside 引理中的 $G$ 群是在 $n$ 个对象上用 $m$ 种颜色进行染色后的方案集合上的置换群。如果色数增加的话($m>2$)，这样做会使不同着色方案的计数问题变得复杂。

下面，我们给出一种改进的方法：

构造具有 $n$ 个对象上的置换群 $\overline{G}$，用 $m$ 种颜色之一给每一对象着色。若一种着色方案在 $\overline{G}$ 的作用下变成另一种方案，则认为这两种着色是同一方案。计算不同染色方案数。

例如 7.2 节[例 1]，我们给每个方块标号 1,2,3,4，如图 7-9 所示。

| | |
|---|---|
| 2 | 1 |
| 3 | 4 |

图 7-9

构造置换群 $\overline{G}=\{g_1,g_2,g_3,g_4\}$，$|\overline{G}|=4$，其中 $g_i$ 的循环节数为 $c(g_i)(i=1,2,3,4)$

在 $\overline{G}$ 的作用下，其中

| | | |
|---|---|---|
| $g_1$ 表示不动， | 即 $g_1=(1)(2)(3)(4)$ | $c(g_1)=4$ |
| $g_2$ 表示逆时针转 90°， | 即 $g_2=(1\ 2\ 3\ 4)$ | $c(g_2)=1$ |
| $g_3$ 表示逆时针转 180°， | 即 $g_3=(1\ 3)(2\ 4)$ | $c(g_3)=2$ |
| $g_4$ 表示逆时针转 270°， | 即 $g_4=(4\ 3\ 2\ 1)$ | $c(g_4)=1$ |

我们可以发现，$g_i$ 的循环节中的对象以相同的颜色所得的图象数 $m^{c(g_i)}$ 正好对应 $G$ 中置换 $P_i$ 作用下不变的图象数

$c(g_1)=4$，　$2^{c(g_1)}=2^4=c_1(P_1)=16$

$c(g_2)=1$，　$2^{c(g_2)}=2^1=c_1(P_2)=2$

$c(g_3)=2$，　$2^{c(g_3)}=2^2=c_1(P_3)=4$

$c(g_4)=1$，　$2^{c(g_4)}=2^1=c_1(P_4)=2$

例如 $g_4=(4\ 3\ 2\ 1)$，对应的 $P_4$ 的一阶循环节为 $(c_1)(c_2)$，图象 $c_1$、$c_2$ 正好是 1，2，3，4 着以同一种颜色的结果（如图 7-10）。

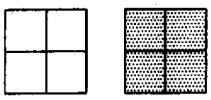


图　7-10

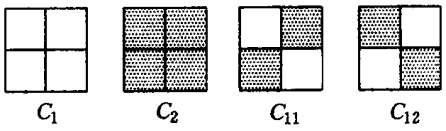



图　7-11

又如 $g_3=(1\ 3)(2\ 4)$，对应的 $P_3$ 的一阶循环节为 $(c_1)(c_2)(c_{11})(c_{12})$，正好是 1 和 3，2 和 4 分别着相同颜色的结果（如图 7-11）。

显然不同染色方案数为　$L=\frac{1}{4}(2^4+2^1+2^2+2^1)=6$（种）

由此得出一个结论：

设 $\overline{G}$ 是 $n$ 个对象的一个置换群，用 $m$ 种颜色涂染着 $n$ 个对象，则不同染色方案为

$$L=\frac{1}{|G|}(m^{c(g_1)}+m^{c(g_2)}+\cdots+m^{c(g_k)})$$

其中 $\overline{G}=\{g_1,g_2,\cdots,g_k\}$　$c(g_i)$ 为置换 $g_i$ 的循环节数 $(i=1,2,\cdots,k)$

这就是所谓的 Pólya 定理。

[**例 1**]　我们用数 1～8 标记立方体的 8 个顶点（图 7-12）。用 $x$ 种颜色给立方体顶点着色的不同方案数是多少？

**解**：上述立方体共有六面。其中

连接顶点 1，4，8，5　组成前一面；

连接顶点 2，3，7，6　组成后一面；

连接顶点 1，2，6，5　组成左一面；

连接顶点 4，3，7，8　组成右一面；

连接顶点 1，2，3，4　组成上一面；

连接顶点 5，6，7，8　组成下一面。

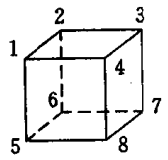



图　7-12

将立方体的 24 种转动看作是 8 个顶点的置换，构成一个群 $G$。

(1) 以前后面中心联线为轴逆时针旋转 0°，90°，180°，270° 共 4 个置换

$g_1=(1)(2)(3)(4)(5)(6)(7)(8)$　　$g_3=(13)(24)(57)(68)$

$g_2=(1432)(5876)$　　$g_4=(1234)(5678)$

(2) 以左右面中心联线为轴逆时针旋转 90°，180°，270° 共 3 个置换

$g_5=(1485)(2376)$　　$g_7=(1584)(2673)$

$g_6=(18)(27)(36)(45)$

(3) 以上下面中心联线为轴逆时针旋转 90°,180°,270°共 3 个置换:

$g_8=(1562)(3487)$　　$g_{10}=(1265)(3784)$

$g_9=(16)(25)(38)(47)$

(4) 以{26,48},(15,37),{12,87},{56,43},{58,23},{14,67} 每组边的中心联线为轴翻转 180°,共 6 个置换:

$g_{11}=(15)(28)(37)(46)$　　$g_{14}=(17)(28)(34)(56)$

$g_{12}=(17)(26)(35)(48)$　　$g_{15}=(17)(23)(46)(58)$

$g_{13}=(14)(28)(35)(67)$　　$g_{16}=(12)(35)(46)(78)$

(5) 分别以对角线 28,36,46,17 为轴旋转 120°和 240°共 8 个置换

$g_{17}=(1)(245)(386)(7)$　　$g_{21}=(186)(247)(3)(5)$

$g_{18}=(1)(254)(368)(7)$　　$g_{22}=(168)(274)(3)(5)$

$g_{19}=(138)(275)(4)(6)$　　$g_{23}=(136)(2)(475)(8)$

$g_{20}=(183)(257)(4)(6)$　　$g_{24}=(163)(2)(457)(8)$

显然,$|G|=4+3+3+6+8=24$

$G$ 中 1 个置换有 8 个循环节数($g_1$);17 个置换有 4 个循环节数($g_3$、$g_6$、$g_9$、$g_{11}$、$g_{12}$、$g_{13}$、$g_{14}$、$g_{15}$、$g_{16}$、$g_{17}$、$g_{18}$、$g_{19}$、$g_{20}$、$g_{21}$、$g_{22}$、$g_{23}$、$g_{24}$);6 个置换有 2 个循环节数($g_2$、$g_4$、$g_5$、$g_7$、$g_8$、$g_{10}$),而颜色数为 $x$,代入 Pólya 公式,即得着色方案数为

$$L=\frac{1}{24}(x^8+17\cdot x^4+6\cdot x^2)$$

最后,我们给出第七章前言中[例 1]的程序题解。该程序不仅给出不同染色方案数,而且还枚举出每一种染色方案。

算法分析:

利用 Pólya 原理或 Burnside 引理的计数公式可得出不同等价类的个数,即无论怎样旋转或翻转都不能吻合的图象个数。但要一一枚举这些图象方案,计数公式就鞭长莫及了。这里,计数公式仅是验证程序结果正确与否的一个测试工具。

为了枚举所有互异的图象,我们将方阵格子由上而下、由左至右编码

1　　2　　3　　4……$N$

$N+1$　$N+2$ ……$N+N$

……

……

$N(N-1)+1$……$N_2$

对上述方阵,我们分别求 1 个黑格的所有互异图象,……,$k$ 个黑格的所有互异图象。在涂 $k_1(1\leqslant k_1\leqslant k)$个黑格时,从位置 1 开始,按位置编码递增的顺序选择方格染黑,即当前染黑的第 $L$ 块方格的位置为 $i$,则第 $L+1$ 块黑色方格的位置 $i+1$ 至 $N\times N$ 区间选择。若搜索完方阵的所有位置,接下来就要判断是否已有等价的方案输出:若产生过等价方案,则前 $k_1$ 个黑色方格的位置必须重新选择;否则这个方案作为同一类代表输出。那么如何判断等价方案呢?

设某方格为$(i,j)$,$a_1=i$,$b_1=j$,则在$N*N$的方阵中与$i$行对称的行位置为$a_2=n-i+1$,与$j$列对称的位置为$b_2=n-j+1$。翻转形式共有8种,如图7-13所示。

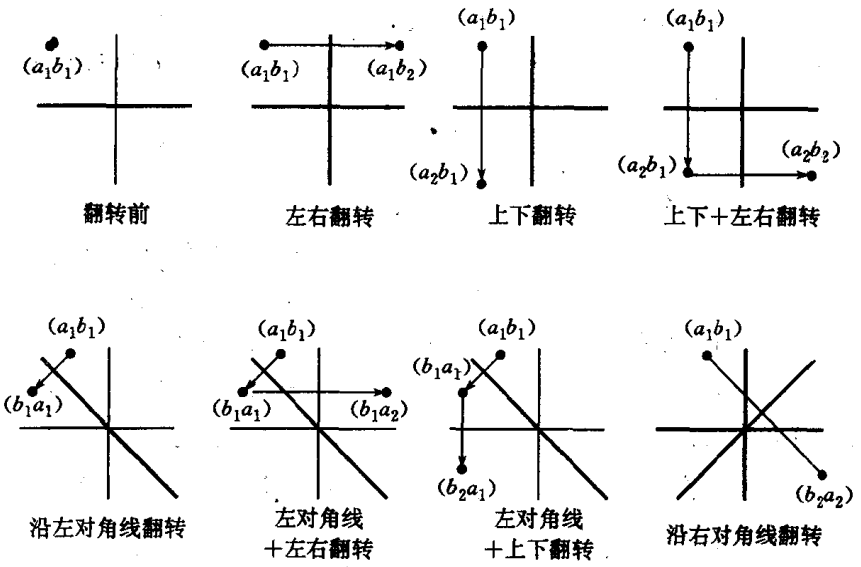



图 7-13

若翻转前后的方格都已涂黑,则属于该翻转形式下的重合现象。若一个方案中的$k$个黑色方格经某形式翻转后,重合于另一方案的$k$个黑色方格,则这两个方案即为等价方案。

在明确了等价的定义后还必须了解判断一个方案是否属于等价的方法。显然,最易得出的一种方法是存储当前已生成的所有不重复方案。每生成一个新方案后即判别该方案是否由上述方案经翻转而来。若是的话,作为等价方案放弃,否则输出并暂存该方案。这个方法虽从原理而言是正确的,但实际执行时颇费时空,得不偿失。

下面介绍一种不需另辟内存(存储已产生方案的)就可以解决问题的简便方法:

每一个方案生成后,我们将各方格对应的7个翻转位置的编码设定为原位置的编码。由于涂色方格的编码1大于空格的编码0,因此我们逐一分析7种翻转形式下各方格的编码与翻转前位置的编码的大小关系:若当前翻转形式下某方格的编码小于翻转前位置的编码,说明已生成过相应的等价方案,该方案放弃;若大于翻转前位置的编码,则分析下一翻转形式……。直至分析完7种翻转形式后,$k_1$个黑色方格的编码都小于等于翻转后位置的编码,则输出这个同类中的最小方案,因为与此相应的等价类还未生成。

我们采用回溯法,自上而下、由左至右地逐个选择方格涂黑。每涂黑$k$格,就依照上述方法判别和输出不同等价类,直至求出1个黑格,…,$k$个黑格的所有互异的图象方案为止。

下面给出程序题解。

```
program program1;
const maxsize                    =10;{ 棋盘规模的最大值 }
type sqtype=array [1..maxsize, 1..maxsize] of 0..1;
```

```
var   n,total,k                :integer;
      { n—棋盘规模; total—方案数; k—黑格数 }
                  {                  {0  空格}
    sq : sqtype; { 棋盘 。sq[i,j]=   {      }
                  {                  {1  黑格}
procedure initialize;
var i: integer;
{ 读入棋盘规模和黑格数，方案数和棋盘初始化 }
begin
    repeat write('n= ');
           readln(n);
    until n in [1..maxsize];
    repeat write('k= ');
           readln(k);
    until k in [1..maxsize * maxsize];
    total :=0; fillchar(sq,sizeof(sq),0);
end;
function small : boolean;
label       break;
var   units :array[1..8] of sqtype; { 旋转的八种方案 }
    h,i,j,a1,a2,b1,b2 :integer;
{ h——辅助变量; i,j——坐标位置; a1,b1——暂存 i,j }
{ a2,b2—与 i 行对称的行位置，与 j 列对称的列位置 }
begin
  fillchar(units,sizeof(units),0); { 旋转方案初始化 }
  for i :=1 to n do
    for j:=1 to n do
      begin a1 :=i;a2 :=n-i+1; { 求与 i 行对称的行位置 }
            b1 :=j;b2 :=n-j+1; { 求与 j 列对称的列位置 }
            units[1,a1,b1] :=sq[i,j]; { 翻转前 }
            units[2,a1,b2] :=sq[i,j]; { 左右翻转 }
            units[3,a2,b1] :=sq[i,j]; { 上下翻转 }
            units[4,a2,b2] :=sq[i,j]; { 上下+左右翻转 }
            units[5,b1,a1] :=sq[i,j]; { 沿左对角线翻转 }
            units[6,b1,a2] :=sq[i,j]; { 左对角线+左右翻转}
            units[7,b2,a1] :=sq[i,j]; { 左对角线+上下翻转}
            units[8,b2,a2] :=sq[i,j]; { 沿右对角线翻转 }
      end;
    small:=false;
    for h :=2 to 8 do   { 将原方案与翻转方案比较 }
     begin
       for i :=1 to n do
         for j :=1 to n do
           if sq[i,j]>units[h,i,j] then exit
            { 若原方案不是同类中最小方案，则返回 false }
           else if sq[i,j]<units[h,i,j] then goto break;
            { 若原方案是目前同类中最小方案，则比较下一翻转方案 }
       break:;
     end;
```

```
        small :=true; { 原方案是所有同类中最小方案,返回 true }
    end;
    procedure print;
    var i,j :integer;
    begin
        if not small then exit; { 若该方案不是最小方案, 回溯 }
        inc(total); { 否则累计和打印方案 }
        writeln('answer no. ',total);
        for i:= 1 to n do
          begin
            for j:=1 to n do if sq[i,j]=1 then write(' * ') else write('. ');
            writeln;
          end;
        readln;
    end;

    procedure make(index,k1:byte);
    { 从第 index+1 个序号位置出发, 递归搜索第 k1+1 个涂黑方格 }
    var i,x,y:byte;
    begin
      if k1>k then exit; { 若已涂的方格数超过 k, 回溯 }
      if index>=n * n then print  { 若搜索完整个棋盘, 则打印方案 }
                      else for i:=index+1 to n * n+1 do
                           { 从 index+1 开始搜索第 k1+1 个涂黑方格 }
                           begin
                             { 将序号位置改成(y,x)坐标 }
                             y:=i div n; x:=i mod n;
                             if x=0 then x:=n else inc(y);
                             sq[y,x]:=1;  { 该方格涂黑 }
                             make(i,k1+1);
                             { 从第 i+1 个序号位置开始递归搜索下一涂黑方格 }
                             sq[y,x]:=0; { 恢复第 i 个序号位置为空格 }
                           end;
    end;
    begin
        initialize; { 初始化 }
        make(0,0); { 从棋盘左上角开始, 递归搜索最多涂黑 k 个方格的所有方案 }
        writeln('Total: ',total); readln; { 打印方案数 }
    end.
```

为了进一步说明枚举所有不同等价类方案的方法,帮助读者编程,我们再举一个例子:

**[例 2]** 有 $2N$ 个棋子,其中有白棋(色码为 1),$N$ 个黑棋(色码为 2)。现在将这 $2N$ 个棋子放入一个 $N * N$ 的棋盘上,使得每行和每列各有一个白棋和一个黑棋。求所有不重复的布棋方案(所谓重复,就是说经过某种翻转后重合)。

算法分析:

我们还是采用回溯法,由上而下、由左至右搜索每行黑、白一对棋子的布棋位置。设当前行为 *step*,黑棋列位置暂定为 $i$、白棋列位置暂定为 $j$。这对棋子的位置是否成立,还得

检查前1至*step*－1行中所有黑棋和白棋的列位置：若某行中黑棋列位置同位于*i*列或白棋列位置同位于*j*列，则放弃当前摆法，重新设定*step*行的另一种摆法，否则*i*列和*j*列分别确定为*step*行黑棋和白棋的列位置，递归搜索*step*＋1行，……在摆完*N*对黑白棋子后，回溯至上一行，继续搜索下一方案。如此递归回溯，直至所有不重复的方案产生为止。

显然上述算法是不难得出的。而问题的难点在于一种方案产生后，如何判断该方案属于不应输出的重复方案？

和[例1]的题解一样，我们设8种翻转形式：

(1) 翻转前；(2) 左右翻转；(3) 上下翻转；(4) 上下翻转＋左右翻转；(5) 沿左对角线翻转；(6) 沿左对角线翻转＋左右翻转；(7) 沿左对角线翻转＋上下翻转；(8) 沿右对角线翻转

当回溯法产生一个符合布棋规则的方案后，我们由上至下，由左至右地分析每个位置的色码。由于布棋按色码由小至大的顺序进行(先布白棋、后布黑棋)，因此同类中的最小方案(该方案下的每个布棋位置的色码都小于等于7种翻转形式(第2种至第8种)下位置的色码)说明相应的等价类还未生成，应予输出。

下面给出程序题解。

```
program program2;
const maxsize                =20;
type sqtype=array [1..maxsize, 1..maxsize] of char;
var   black,white            :array[1..maxsize] of byte;
      { black[i],white[i]——第i行黑棋和白棋的列位置 }
      n,total                :integer; { 棋盘规模、方案数 }
      scr : sqtype;                    { 布棋方案 }
procedure initialize;{ 输入棋盘规模、方案数初始化 }
var i: integer;
begin
    repeat write('n= ');
           readln(n);
    until n in [1..maxsize];
    total :=0;
END;
function small : boolean;
label      break;
var   units :array[1..8] of sqtype;
    h,i,j,a1,a2,b1,b2 :integer;
begin
  for i :=1 to n do
    for j:=1 to n do
      begin a1 :=i;a2 :=n-i+1; { 求与i行对称的行位置 }
            b1:=j; b2 :=n-j+1; { 求与j列对称的列位置 }
            units[1,a1,b1] :=scr[i,j]; { 翻转前 }
            units[2,a1,b2] :=scr[i,j]; { 左右翻转 }
            units[3,a2,b1] :=scr[i,j]; { 上下翻转 }
            units[4,a2,b2] :=scr[i,j]; { 上下＋左右翻转 }
            units[5,b1,a1] :=scr[i,j]; { 沿左对角线翻转 }
```

```
                units[6,b1,a2]:=scr[i,j]; { 左对角线+左右翻转 }
                units[7,b2,a1]:=scr[i,j]; { 左对角线+上下翻转 }
                units[8,b2,a2]:=scr[i,j]; { 沿右对角线翻转 }
            end;
        for h:=2 to 8 do { 将原方案与翻转方案比较 }
            begin for i:=1 to n do
                    for j:=1 to n do
                        if units[1,i,j]<units[h,i,j] then
                        { 若现翻转方案不是目前同类中最小方案，则回溯 }
                            begin small:=false; exit; end
                        else if units[1,i,j]>units[h,i,j] then
                        { 否则再比较下一翻转方案 }
                            goto break;
                  break:;
            end;
        small:=true;
    end;
    procedure print;
    var i,j:integer;
    begin
        fillchar(scr,sizeof(scr),'.');
        for i:=1 to n do { 将 n 对黑白棋布入相应位置 }
            begin scr[i,black[i]]:='b';
                  scr[i,white[i]]:='w';
            end;
        if not small then exit; { 若该方案非同类的最小方案，则回溯 }
        inc(total);             { 否则方案数+1，输出该方案 }
        writeln('answer no. ',total);
        {for i:= 1 to n do
            begin for j:=1 to n do write(scr[i,j]:2);
                  writeln;
            end;
        readln;}
    end;

    procedure solve(step:integer);
    var i,j,k:integer;
        flag:boolean;
    begin
        if step =n { 若布完 n 对黑白棋，则检查是否产生过等价类，输出不重复方案 }
            then print
            else for i:=1 to n do { 由左至右搜索第 step+1 行黑白棋子的位置 }
                    for j:=1 to n do
                      if i<>j then
                          begin flag:=true;
                                k:=1;
                                while (k<=step) and flag do
                                  if (black[k]=i)or(white[k]=j)then flag:=false else inc(k);
                                if flag then { 若前 step 行未有位于 i 列的黑棋和位于 j 列的白棋 }
```

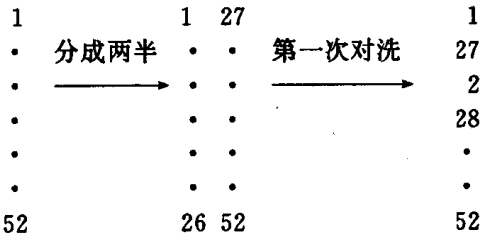
```
                                begin black[step+1]:=i;
                                { 则(step+1, i)布黑棋, (step+1,j)布白棋 }
                                    white[step+1]:=j;
                                    solve(step+1);
                                { 递归搜索下一对黑白棋子的位置 }
                                end;
                        end;
    end;

    begin
        initialize; { 初始化 }
        solve(0);   { 从第一对黑白棋子开始, 递归搜索所有不重复的布棋方案 }
        writeln('Total: ',total); readln; { 输出方案总数 }
    end.
```

## 习 题 七

1. 现有 $n^2-1$ 块标记着 1,2,…,$n^2-1$ 的骨牌,放置在 $n\times n$ 的棋盘上,棋盘留有一个空格,每次只允许与空格相邻的骨牌可以移至空格。

输入一个初始格局和一个目标格局。问可否经过若干次移动,使初始格局变成目标格局。

2. 一副扑克牌 52 张,已按一定顺序排列,并自上而下编号为 1,2,…,52,我们按以下方法洗牌使顶牌和底牌不动,其他牌交替放置。即



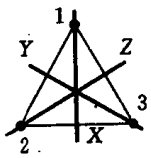

问这种方法洗牌,洗多少次后,才能使一副牌回复原来的顺序?

3. 在有 $n$ 个位置的轮盘的轮上($n\geqslant 3$,见题图 7-1),用 $r(1\leqslant r\leqslant n)$ 种颜色着色,可以有多少种不同方案?

若相邻格不允许同色,可以有多少种不同方案?

(这样轮子能旋转,但不能翻转到它的镜像位置)


题图 7-1

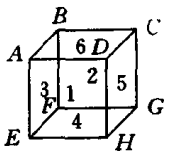



题图 7-2

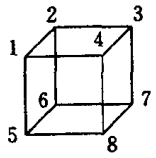
4. 求用三种颜色对一个等边三角形顶点着色的方案数。

题图 7-2 所示的等边三角形可绕中心旋转,并可对称轴翻转。

5. 题图 7-3 为一个立方体，用 1,2,…,6 分别标记为 $ADHE$(前)，$BCGF$(后)，$ABFE$(左)，$EFGH$(下)，$DCGH$(右)，$ABCD$(上)。用六种颜色给立方体的六个面着色，每面颜色不同，并且当一个着色的立方体经转动可得到另一个时，则认为两者相同。问有多少种着色的方案？




题图 7-3




题图 7-4

6. 我们用数 1～8 标记立方体的 8 个顶点(见题图 7-4)，立方体有 24 种转动。用 $x$ 种颜色给立方体顶点着色，不同的方案数是多少？

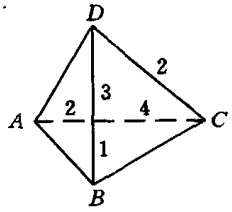
7. 给立方体的面着色，要求颜色 1 涂若干面，颜色 2 涂若干面，……。若经过旋转能吻合的着色认为是相同的，问有多少不同的着色方案？

8. 有四颗珠子，其中两颗蓝色，一颗红色和一颗黄色，能制成多少种项链？

9. 正四面体有四个全等的面(见题图 7-5)，分别用 1,2,3,4 标记 $ABC$，$ABD$，$ACD$，$BCD$。

(1) 用 $m$ 种颜色给四面体着色(每面着一色)，问有多少种方案？

(2) 用两面红、一面蓝、一面黄给四面体着色，共有多少种方案？




题图 7-5

10. 给一根 $n$($n$ 是偶数)尺长的棍子着色，每尺着不同颜色，共有 $m$ 种颜色可供选择。仅有的变换是翻转 180°。问

(1) 给棍子着色，有多少种方案？

(2) 在 $n$ 尺中，有 $n_1$ 尺着颜色 1，$n_2$ 尺着颜色 2，…，$n_k$ 尺着颜色 $k$，$n_1+n_2+\cdots+n_k=n$，有多少种着色方案？

# 第八章　组合设计

## 8.1　问题的提出

组合设计是组合数学的一个重要分支，其中心内容是使用组合论的方法构造 $N$ 个事物的某个排列，使之满足特定的性质。通常解这类问题是非常困难的，但人们经过长期研究，在某些方面，诸如魔方、拉丁方等方面已有所突破，并在实验设计等领域获得了应用。为了说明问题是如何提出的，先看下面几个例子。

[**例 1**]　魔方与魔和

用 $n^2$ 个整数，构造一个 $n\times n$ 数组，使得数组的每行元素之和、每列元素之和、每条对角线之和都等于 $S$，则称该数组为 $n$ 阶魔方，称 $S$ 为魔和。

一般说来，$n$ 阶魔方不是唯一的。但是任何 $n$ 阶魔方，其魔和 $S=\frac{1}{2}\times n(n^2+1)$ 总是常数。

[**例 2**]　36 名军官问题

有 36 名来自 6 个不同团的军官，每个团 6 名且分属于 6 种不同军衔。能否把他们排成 6×6 的方形阵列，使得每行、每列的 6 名军官正好来自 6 个不同的团且属于不同的军衔？

**解**：设有序数对 $(i,j)$ 表示一名军官，其中

$i$——该军官的军衔；　$j$——该军官所属军团　$(i,j=1,2,\cdots,6)$。

于是问题就变成了 36 个有序对排成 6×6 的方阵，使得每行和每列中所有有序对的第一个分量是集合 $\{1,2,\cdots,6\}$ 的全排列，第二个分量也是集合 $\{1,2,\cdots,6\}$ 的全排列。这样的问题又可以叙述为两个 6×6 的方阵，其元素取自集合 $\{1,2,\cdots,6\}$，使得这两个方阵满足下述条件：

(1) 方阵的每一行和每一列都是集合 $\{1,2,\cdots,6\}$ 的一个全排列；

(2) 当两个方阵并置时，所有 36 个不同有序对 $(i,j)$ 都必须出现（其中 $i,j=1,2,\cdots,6$）。

如果在这个问题中，把 6 换成 $n$，36 换成 $n^2$，则问题就一般化了。为了用数学模型来描述和研究这个问题，我们引入两个概念：

1. $n$ 阶拉丁方的定义

设 $S=\{S_1,S_2,\cdots,S_n\}$ 是一个具有 $n$ 个元素的集合，$A=(a_{ij})$ 是 $n\times n$ 的矩阵。若 $A$ 的每一行都是 $S$ 的不同全排列，$A$ 的每一列也是 $S$ 的不同全排列，则称矩阵 $A$ 为 $n$ 阶拉丁方。

为方便起见，通常取 $S=\{1,2,\cdots,n\}$。

例如 $x$ 和 $y$ 都是 3 阶拉丁方，这两个 3 阶拉丁方是不同的。

$$x=\begin{bmatrix}1&2&3\\3&1&2\\2&3&1\end{bmatrix}\qquad y=\begin{bmatrix}1&2&3\\2&3&1\\3&1&2\end{bmatrix}$$

2. 两个 $n$ 阶拉丁方正交的定义

设 $A=(a_{ij})$，$B=(b_{ij})$是两个不同的 $n$ 阶拉丁方，如果由这两个拉丁方中的元素构成的 $n^2$ 个有序对$(a_{ij},b_{ij})(i,j=1\cdots n)$都是不同的，则称拉丁方 $A$ 与 $B$ 是正交的。

例如：

$$(x,y)=\begin{bmatrix}(1,1)&(2,2)&(3,3)\\(3,2)&(1,3)&(2,1)\\(2,3)&(3,1)&(1,2)\end{bmatrix}$$

由于在并置的方阵$(x,y)$中，所有的有序对都是不同的，故 $x$ 与 $y$ 是正交的。

如果把上例看成是 3 个不同团来的具有 3 种不同军衔的 9 名军官，则由于 $x,y$ 的并置方阵$(x,y)$中的数对互不相同，于是$(x,y)$就是题目所要求的 3×3 的方形阵列。这样[例 2]可重新叙述为：是否存在两个正交的 6 阶拉丁方？更一般地，对于任意正整数 $n$，是否存在两个正交的 $n$ 阶拉丁方？

几十年来，数学家们对这个问题进行了艰辛的研究，终于证实了除 1、2、6 外的任意一个正整数 $n$，都存在两个正交的 $n$ 阶拉丁方(即 36 名军官问题的排法是不可能的)。

从表面上看，拉丁方问题似乎是一种数学游戏，但在实际生活中都有着重要的应用，特别是涉及到科学实验的实际问题。

[**例 3**] 在一定类型的冬小麦田里，测试不同份量的水和各种化肥的效果。假定有 $n$ 种水的份量和 $n$ 种化肥要测试，共有 $n^2$ 种可能的水和化肥的组合。试问如何在一块正方形的地里，设计这种测试实验呢？

**解**：我们把正方形的地均分成 $n^2$ 个小正方形。让水和化肥的 $n^2$ 种组合分布在这 $n^2$ 块中，每种一块。但小麦高产与否不仅取决于化肥和水的份量，还和土壤的原有肥力有关。而原有肥力不可能在整块地上都很均匀。为了使土壤原肥力的影响最小，最好把 $n$ 种化肥在 $n^2$ 块土地上的分布形成拉丁方 $A$，把 $n$ 种水在 $n^2$ 块土地上的分布形成拉丁方 $B$，并使 $A,B$ 正交，例如 $n=3$，3 种化肥用 $a,b,c$ 标记，3 种水用 1，2，3 标记。合适的实验设计应该如图 8-1 所示。

| | | |
|---|---|---|
| $a$ | $b$ | $c$ |
| $b$ | $c$ | $a$ |
| $c$ | $a$ | $b$ |

| | | |
|---|---|---|
| 2 | 3 | 1 |
| 1 | 2 | 3 |
| 3 | 1 | 2 |

并置 →

| | | |
|---|---|---|
| $(a,2)$ | $(b,3)$ | $(c,1)$ |
| $(b,1)$ | $(c,2)$ | $(a,3)$ |
| $(c,3)$ | $(a,1)$ | $(b,2)$ |

图 8-1

由于魔方和拉丁方问题涉及离散数学、近代数学中有限域和有限几何的概念、定义、结论，而对这些玄奥、抽象的知识非一般中学生所能了解且又不是组合数学的组成部分，因此我们在这一章中仅给出上述组合设计的算法和程序，而对算法的由来不作证明。

## 8.2 魔方与魔和

用整数 $1,2,\cdots,n^2$ 构造一个 $n$ 阶魔方，其魔和（即每行、每列或每条对角线的元素和）为 $S=\frac{n}{2}\times(n^2+1)$。但是二阶魔方为

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

不存在。因为 $a+b=a+c$，于是 $b=c$，与魔方的构造规则矛盾。

除二阶魔方不存在外，对任何 $n\geqslant 1$ 都可以构造一个 $n$ 阶魔方。但魔方形成不唯一，构造的方法大都也很复杂，目前还未有能构造任意阶魔方的一般方法。

下面介绍一种构造奇数阶魔方的简便方法：

依次把整数 $1,2,\cdots,n^2$ 填入 $n\times n$ 数组的 $n^2$ 个位置上。填写方式如图 8-2 所示。

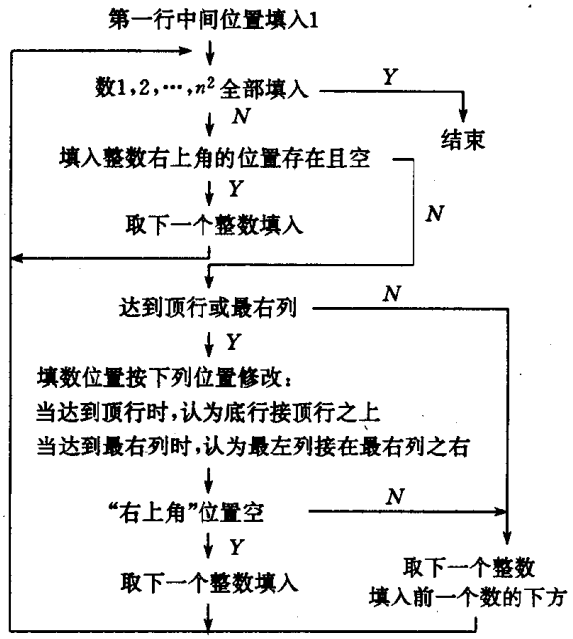
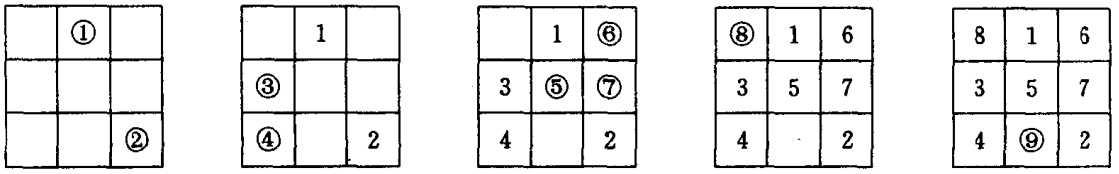



图 8-2

例如，依上述算法构造一个 3 阶魔方，如图 8-3 所示。

最后给出构造奇数阶魔方的程序。

```
program mo_fang;

uses
  crt;

const
  maxn              = 100;
type
```




顶行中间位置(1,2)填入1，然后将底行接顶行之上，在1的"右上角"(3,3)位置填入2；

由于2处于最右列，因此将第1列接在第3列之右，在2的"右上角"填入3，又由于3的右上角非空，因此4填入3的下方(3,1)；

4的右上角填5，5的右上角填6。由于6处于顶行最右列，因此7填入6的下方(2,3)；

由于7处最右列，因此将第1列接在最右列之右，在7的"右上角"填8；

由于8处于最顶行，因此将底行接在顶行之上，在8的"右上角"(3,2)填9。

(a) (b) (c) (d) (e)

图 8-3

```
  listtype          = array[0..maxn,0..maxn] of integer;

var
  list              : listtype;      { 魔方 }
  n                 : integer;

procedure init;
  begin
    clrscr;
    repeat write('n=');
           readln(n);
    until (n>0) and (n<=maxn) and (odd(n));     { 输入阶数 n }
  end;

procedure main;
  var i,j,x,y : integer;
    begin
      fillchar(list,sizeof(list),0);
      x:=0; y:=n div 2;
      for i:=0 to n-1 do
        begin
          for j:=1 to n do
            begin
              list[x,y]:=i*n+j;          { 填入一数 }
              x:=(n+x-1) mod n;          { 向右上方移一格 }
              y:=(y+1) mod n;
            end;
          x:=(x+2) mod n;                { 向下移一格 }
          y:=(n+y-1) mod n;
        end;
    end;
```

```
procedure show;                          { 显示 }
  var i,j : integer;
    begin
      for i:=0 to n-1 do
        begin
          for j:=0 to n-1 do
            write(list[i,j]:3);
          writeln;
        end;
    end;

begin
  init;{ 输入阶数 }
  main;{ 构造奇阶魔方 }
  show;{ 显示结果 }
end.
```

## 8.3 拉丁方的构造

### 一、构造拉丁方的一般方法

**[例 1]**

算法分析：

对于任何正整数 $n$，我们可以按下述方法构造一个 $n$ 阶拉丁方。

(1) 首先将 1,2,…,$n$ 作为 $n$ 阶拉丁方的第一行；

(2) 将第一行最后一列的 $n$ 与前面的 $n-1$ 列交换到第 1 列，就得到 $n$ 阶拉丁方的第二行：

$$\begin{bmatrix} 1 & 2 & \cdots\cdots & \cdots\cdots & n \\ n & 1 & 2 & \cdots\cdots & n-1 \end{bmatrix}$$

(3) 再将第二行中最后一列 $n-1$ 与第二行前面的 $n-1$ 列交换到第一列，就得到 $n$ 阶拉丁方的第三行：

$$\begin{bmatrix} 1 & 2 & \cdots\cdots & n \\ n & 1 & 2\cdots\cdots & n-1 \\ n-1 & n & 1\cdots\cdots & n-2 \end{bmatrix}$$

反复这样的过程，直至得到一个 $n\times n$ 拉丁方为止。

例如下面 6×6 拉丁方就是按以上方法得到的：

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 1 & 2 & 3 & 4 & 5 \\ 5 & 6 & 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 1 & 2 & 3 \\ 3 & 4 & 5 & 6 & 1 & 2 \\ 2 & 3 & 4 & 5 & 6 & 1 \end{bmatrix}$$

这种构造拉丁方的方法，不一定以 1,2,…,$n$ 作第一行，还可以从 1,2,…,$n$ 的任一个全排列作第一行，由此构造一个 $n$ 阶拉丁方。

下面给出上述构造方法的程序：

```
program Norm_LadinSquare;
uses CRT;
var
  n, i, j, Last : integer;
  { n—拉丁方阶数; i,j—辅助变量; Last—上一行的行末数 }
begin
  ClrScr;
  repeat
    Write('n='); Readln(n); { 输入拉丁方的阶数 }
  until (n in [1..25]);
  for i := 1 to n do Write(i : 3); { 输出首行 }
  Writeln;
  Last := n; { 置 Last 为首行的行末数 }
  for i := 2 to n do begin { 输出 2 至 n 行 }
    for j := Last to n do Write(j : 3); { 输出当前行 Last,Last+1,…,n }
    for j := 1 to Last - 1 do Write(j : 3);          { 1, 2, …,Last-1 }
    Writeln;
    Last := j; { 置行尾数为 Last(即下行的行首数据) }
  end;
  Readln;
end.
```

## 二、构造带有约束条件的拉丁方

若要构造带有约束条件的拉丁方，就不能简单套用上述方法，而是要对约束条件认真斟酌一番。若不能直接推导出同行和同列数据间的排列规律，则必须改用搜索的方法求解。

[例 2] 求最小 $n$ 阶拉丁方(即拉丁方每行表示的 $n$ 位十进制数必须最小)。

算法分析：

按由上而下、由左至右的顺序逐格填写。欲填$(y,x)$格时，则分别从 $x$ 列的第一行和 $y$ 行的第一列开始，顺序搜索 $y$ 行和 $x$ 列。若 $x$ 列的 $i$ 行和 $y$ 行的 $i$ 列都未曾填数，则$(y,x)$填写数据 $i$。然后依次填写$(y+[x/n],x \text{ Mod } n+1)$格。

按上述规律不断回溯搜索，直至填写完 $n\times n$ 格。此时产生的方阵即为最小拉丁方。

```
const Maxn = 100; { 拉丁方阵的最大阶数 }
var
  Square : array [1..Maxn, 1..Maxn] of 0..Maxn;
           { 最小拉丁方阵 }
  Markx, {
                                  {true (i,j)已填数 }
         {  Markx[i,j]=           {                 }
         {                        {false (i,j)未填数}
```

```
  {Marky   : array [1..Maxn, 1..Maxn] of boolean;
           {                 {true (j,i)已填数 }
           { Marky[i,j]=     {                 }
           {                 {false (j,i)未填数}
  n, j     : integer;
           { n——拉丁方阵的阶数, j——辅助变量 }

procedure Run(x, y : integer);
{ 递归搜索最小拉丁方阵。(y,x)——当前待填的坐标 }
var i : integer;
begin
  if y > n then begin
  { 若所有格子均已填完，则输出最小拉丁方阵，并退出程序 }
    for i := 1 to n do begin
      for j := 1 to n do Write(Square[i, j] : 3);
      Writeln;
    end;
    Readln;
    Halt;
  end;

  for i := 1 to n do { 按→的顺序检查 y 行，按↓的顺序检查 x 列 }
    if (not Markx[x, i]) and (not Marky[y, i]) then begin
    { 若 i 行的 x 列和 y 行的 i 列未填数 }
      Markx[x, i] := true;   Marky[y, i] := true;   Square[y, x] := i;
      { 则数 i 填入(y,x) }
      Run(x mod n + 1, y + x div n);
      { 递归填写下一格 }
      Markx[x, i] := false;  Marky[y, i] := false;  Square[y, x] := 0;
      { 恢复递归前的数据 }
    end;
end;

begin
  repeat
    Write('N='); Readln(n); { 输入拉丁方阵的阶数 }
  until n in [1..Maxn];
  { 拉丁方阵及有关变量初始化 }
  Fillchar(Square, Sizeof(Square), 0);
  Fillchar(Markx, Sizeof(Markx), false);
  Fillchar(Marky, Sizeof(Marky), false);
  Run(1, 1); { 递归搜索最小拉丁方阵 }
end.
```

## 三、扩充拉丁矩

首先，我们必须明确什么是拉丁矩？

设 $A=(a_{ij})$是一个 $m\times s$ 矩阵。若 $A$ 的任一行是集合 $1,2,\cdots,n$ 中取 $s$ 个元素的排列，

任一列是集合 $1,2,\cdots,n$ 中取 $m$ 个元素的排列，则称 $A$ 是一个 $m\times s$ 拉丁矩($m\leqslant n, s\leqslant n$)。

那么，拉丁矩 $A$ 在什么条件下可以扩展成 $n\times n$ 的拉丁方？如何扩展？下面，我们就来讨论这个问题。

设 $N[i]$——元素 $i(1\leqslant i\leqslant n)$在拉丁矩 $A$ 中出现的次数。

首先我们设法添加 $n-s$ 列构成一个 $m\times n$ 的拉丁矩。由于 $A$ 中每个元素 $i$ 在新添的 $n-s$ 列中，每列至多只能有一个，因此在 $n-s$ 列中至多只有 $n-s$ 个 $i$。故 $i$ 在 $m\times n$ 拉丁矩中出现的次数至多为 $N(i)+n-s$。而在 $m\times n$ 拉丁矩中 $i$ 出现的次数为 $m$，从而有

$$N(i)+n-s\geqslant m$$

即
$$N(i)\geqslant m+s-n$$

换句话说，若原 $m\times n$ 的 $A$ 拉丁矩中每个元素出现的次数大于等于 $m+s-n$，则这个拉丁矩一定可以扩展为 $n$ 阶拉丁方。

我们应从集合 $1,2,\cdots,n$ 中选择具有下述特征的 $m$ 个元素作为一列添加到原拉丁矩上。

(1) 扩展的拉丁矩的任一行不会出现两个或两个以上的相同元素；

(2) 这 $m$ 个元素在原拉丁矩 $A$ 中出现的次数小于 $m$ 且包含了满足

$$N(i)=m+s-n$$

的全部元素 $i$。

按照拉丁方的定义，特征 1 的道理不言而喻。至于特征 2 的理由，简述如下：

在 $m\times s$ 拉丁矩中，设有集合 $1,2,\cdots,n$ 中的 $k$ 个 $i$，使得 $N(i)=m$，于是其余的 $n-k$ 个元素在 $A$ 中共出现 $ms-mk$ 次。对这 $n-k$ 个元素中的每个元素来说，在拉丁矩 $A$ 中出现的平均次数

$$\frac{ms-mk}{n-k}\geqslant m+s-n$$

将上式整理化简得

$$n-k\geqslant m$$

这说明，在集合 $1,2,\cdots,n$ 中可以找到 $m$ 个元素，它们在 $A$ 中出现的次数小于 $m$。

另一方面，在集合 $1,2,\cdots,n$ 中有 $l$ 个元素，使得

$$N(i)=m+s-n$$

其余 $n-l$ 个元素中的每个元素在拉丁矩 $A$ 中都最后出现 $m$ 次，从而有

$$(n-l)m+l(m+s-n)\geqslant ms$$

整理化简得
$$l\leqslant m$$

由上可见，可以选择 $m$ 个在 $A$ 中出现次数小于 $m$ 的元素，这 $m$ 个元素包含了满足

$$N(i)=m+s-n$$

的全部元素 $i$，作为一列添加到原拉丁矩上，并且使得任一行不会出现重复元素。

这样扩展后得到的 $m\times(s+1)$拉丁矩仍然满足式子

$$N(i)\geqslant m+s-n$$

只要 $s+1<n$，就可以按上述方法继续添加列，构造出新的拉丁矩，直至 $m\times n$ 拉丁矩产生

为止。

上述方法对行同样适用，因而可以按以上方法不断添加行，直至构造出 $n$ 阶拉丁方。

例如将 3×3 的拉丁矩

$$\begin{bmatrix}1 & 3 & 6\\ 2 & 5 & 4\\ 4 & 2 & 5\end{bmatrix}$$

扩展成 6 阶拉丁方

**解**：$N(i)\geqslant 3+3-6 \quad (i=1,2,\cdots,6)$

由于在拉丁矩中，集合 1,2,…,6 中的每个元素均出现，且每个元素出现的次数都不足 3，因此可以任意添加三个元构成一列。例如添加 1,2,3，得

$$\begin{bmatrix}1 & 3 & 6 & 2\\ 2 & 5 & 4 & 1\\ 4 & 2 & 5 & 3\end{bmatrix}$$

这时 $N(6)=3+4-6=1$，因此再添新列时需添加 6。由于 2 已出现 3 次，故不能再添加。按此法可得

$$\begin{bmatrix}1 & 3 & 6 & 2 & 4 & 5\\ 2 & 5 & 4 & 1 & 6 & 3\\ 4 & 2 & 5 & 3 & 1 & 6\end{bmatrix}$$

下面再添行。因为对于符合 1,2,…,6 中的每个元素 $i$，$N(i)=3+6-6=3$，从而都应选取。故可得

$$\begin{bmatrix}1 & 3 & 6 & 2 & 4 & 5\\ 2 & 5 & 4 & 1 & 6 & 3\\ 4 & 2 & 5 & 3 & 1 & 6\\ 3 & 6 & 2 & 4 & 5 & 1\\ 5 & 4 & 1 & 6 & 3 & 2\\ 6 & 1 & 3 & 5 & 2 & 4\end{bmatrix}$$

最后给出扩展拉丁矩的程序题解。

```
Program  Lardin;
Const max=20;      { 拉丁方最大规模 }
Var sq,units:array[1..max,1..max]of byte; { 拉丁矩，中间矩阵 }
    n,m,s:byte;   { 拉丁方的阶数，待扩展的拉丁矩的行数和列数 }
    row,col:array[1..max]of set of 1..max; { 行列元素集合 }
    nn:array[1..max]of 1..max; { 频率数组 }
procedure init;  { 初始化 }
 var i,j:integer;
 begin
  fillchar(sq,sizeof(sq),0);  fillchar(nn,sizeof(nn),0); {  拉丁矩和频率数组初始化 }
  fillchar(row,sizeof(row),0);fillchar(col,sizeof(col),0); {  行元素集合和列元素集合初始化 }
  repeat write('M, S, N = '); readln(m,s,n);
{ 输入拉丁矩的行、列数和扩展后的拉丁方的阶数 }
  until (m in [1..max])and(n in [1..max])and(s in [1..max])and(n>=m)and(n>=s);
```

```
  writeln('input lartin-square');
  for i:=1 to m do{ 输入 m*s 的拉丁矩 }
   begin
    for j:=1 to s do
     begin
      read(sq[i,j]); inc(nn[sq[i,j]]); { 输入(i,j)的元素,计算其频率 }
      if (not sq[i,j] in [1..n])or(sq[i,j] in row[i])or(sq[i,j] in col[j])
          then begin writeln('Input error');halt end;
      row[i]:=row[i]+[sq[i,j]]; col[j]:=col[j]+[sq[i,j]];
     {若该元素符合拉丁矩要求,则分别进入 i 行和 j 列的元素集合 }
     end;
    readln;
   end;
  j:=m+s-n;
  for i:=1 to n do if nn[i]<j then begin writeln('No answer at ',i); halt end;
  { 若存在频率小于 m+s-n 的元素,则拉丁矩无法扩充为 n 阶拉丁方,失败退出 }
 end;
procedure turn; { 转置 }
 var i,j:byte;
 begin
  fillchar(units,sizeof(units),0); { 中间矩阵初始化 }
  for i:=1 to n do
   for j:=1 to n do
    units[i,j]:=sq[j,n-i+1];
  sq:=units;
  s:=m;m:=n;
  for i:=1 to m do row[i]:=col[n-i+1];
  fillchar(col,sizeof(col),0);
 end;
procedure make(x,y:byte); { 从 m*s 的拉丁矩中的(x, y) 出发,递归扩展 n 阶拉丁矩 }
 var i,j:integer;
     limit:set of 1..max; { x 列和 y 行的元素集合 }
 begin
  if y>m then begin inc(x); y:=1;inc(s) end; { 新增一列 }
  if x>n then
    if m<n then begin  { 若扩充完 n 列且剩有行待扩充,则 }
                  turn; { 转置 }
                  make(s+1,1) { 扩充转置拉丁矩 }
                 end
           else begin  { 打印转置后的拉丁方 }
                  for i:=1 to n do
                    begin for j:=1 to n do write(sq[n-j+1,i]:3);  writeln end;
                  writeln;
                  halt;
                 end;
  limit:=row[y]+col[x]; { 求 x 列和 y 行的元素集合 }
  j:=m+s-n;
  if j>=0 then
    for i:=1 to n do
```

```
        if (nn[i]=j) and not (i in limit) then
        { i的出现频率为 m+s-n 且不在 x 列和 y 行，则 i 进入(x,y),累计频率}
          begin
            sq[y,x]:=i;col[x]:=col[x]+[i];row[y]:=row[y]+[i]; inc(nn[i]);
            make(x,y+1); { 递归扩充(x,y+1) }
            col[x]:=col[x]-[i];row[y]:=row[y]-[i]; dec(nn[i]); { 恢复,i 退出(x,y) }
          end;
    for i:=1 to n do
     if not (i in limit)and(nn[i]<m)and(nn[i]<>j) then
     { i的出现频率小于 m 且不等于 m+s-n 且不在 x 列和 y 行，则 i 进入(x,y),累计频率}
      begin
        sq[y,x]:=i;col[x]:=col[x]+[i];row[y]:=row[y]+[i]; inc(nn[i]);
        make(x,y+1); { 递归扩充(x,y+1) }
        col[x]:=col[x]-[i];row[y]:=row[y]-[i]; dec(nn[i]); { 恢复,i 退出(x,y) }
      end;
   end;
  begin
   init; { 输入拉丁矩 }
   make(s+1,1); { 从(s+1,1)开始，递归扩充 n 阶拉丁方 }
  end.
```

## 8.4 构造奇数阶正交拉丁方

在 8.1 节中已经定义过正交拉丁方。虽然对除 1,2,6 外的任意一个正整数 $n$，都存在两个正交的 $n$ 阶拉丁方，但现在还未有一个构造任意阶正交拉丁方的一般方法。因此在这一节里，我们仅对构造一对奇数阶正交拉丁方的算法展开讨论。

设 $n$ 是奇数，因此 $n$ 可以分解成 $n=P_1^{t_1}\cdot P_2^{t_2}\cdots P_k^{t_k}$ 的质因数形式。这里，$P_i$ 表示除 2 以外的质数，$t_i$ 表示正整数($1\leqslant i\leqslant k$)

构造 $n$ 阶正交拉丁方的算法蕴含在以下定理中：

若已有一对 $n_1$ 阶正交拉丁方和一对 $n_2$ 阶正交拉丁方，则可构造出 $n=n_1\cdot n_2$ 阶正交拉丁方。

显然，我们可以先应用这个定理到 $P_1^{t_1}\cdot P_2^{t_2}$，构造出一对 $n_1=P_1^{t_1}\cdot P_2^{t_2}$ 阶的正交拉丁方，再应用它到$(P_1^{t_1}\cdot P_2^{t_2})\cdot P_3^{t_3}$，构造出一对 $n_2=n_1\cdot P_3^{t_3}$ 阶正交拉丁方……，如此以往 $k-1$ 次，即可构造出一对 $n=n_{k-2}\cdot P_k^{t_k}$ 阶正交拉丁方。

### 一、构造 $n=P^t(n\geqslant 3)$阶正交拉丁方

当 $n=P^t$($P$ 为质数，$t$ 为正整数，$n\geqslant 3$)时，我们可以通过下述算法构造出 $n-1$ 个两两正交的拉丁方 $L_1,L_2,\cdots,L_{n-1}$：

(1) $L_i$ 的第一行总是$(1,2,\cdots,n)$；

(2) 下一行的第一个元素值等于上行行尾的元素值加$(i+1)$。若结果值大于 $n$，则把结果值减去 $n$；

(3) 若计算出行首元素值为 $j$，则本行上的 $n$ 个元素值是

$$(j, j+1, \cdots n, 1, 2, \cdots, j-1) \quad (i = 1, 2, \cdots, n-1)$$

例如　当 $n=3$ 时，用上述算法很容易构造出 2 个 3 阶正交拉丁方

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

同样，当 $n=5$ 时，可用上述算法构造出 4 个两两正交的 5 阶拉丁方

$$B_1 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \\ 3 & 4 & 5 & 1 & 2 \\ 4 & 5 & 1 & 2 & 3 \\ 5 & 1 & 2 & 3 & 4 \end{bmatrix} \qquad B_2 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 1 & 2 \\ 5 & 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 & 1 \\ 4 & 5 & 1 & 2 & 3 \end{bmatrix}$$

$$B_3 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 1 & 2 & 3 \\ 2 & 3 & 4 & 5 & 1 \\ 5 & 1 & 2 & 3 & 4 \\ 3 & 4 & 5 & 1 & 2 \end{bmatrix} \qquad B_4 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \\ 4 & 5 & 1 & 2 & 3 \\ 3 & 4 & 5 & 1 & 2 \\ 2 & 3 & 4 & 5 & 1 \end{bmatrix}$$

## 二、构造 $n=n_1 * n_2$ 阶的一对正交拉丁方

若已知一对 $n_1$ 阶正交拉丁方 $A_1$、$A_2$ 和一对 $n_2$ 阶正交拉丁方 $B_1$、$B_2$，则可构造一对 $n=n_1 \cdot n_2$ 阶正交拉丁方 $C_1$、$C_2$。构造过程是：

用 $A_1$ 和 $B_1$ 构造 $C_1$，用 $A_2$ 和 $B_2$ 构造 $C_2$。

构造 $C_1$ 时，$C_1$ 按 $A_1$ 式样分块，用 $A_1$ 的元素标记 $C_1$ 的每一块，再把 $B_1$ 填入各块。$C_1$ 的元素是偶时，其第一分量是块的标记，第二分量就是 $B_1$ 填入时的相应元素。最后用数字 $1,2,\cdots,n_1 * n_2$ 代替序偶，即

$(1,1)=1$，　　$(1,2)=2$，　　…… $(1,n_2)=n_2$

$(2,1)=n_2+1$，　　$(2,2)=n_2+2$，　　…… $(2,n_2)=2n_2$

……

……

$(n_1,1)=n_2*(n_1-1)+1$，　$(n_1,2)=n_2(n_1-1)+2$，　…… $(n_1,n_2)=n_1 * n_2$

得到 $n=n_1 * n_2$ 阶拉丁方 $C_1$。用同样方法构造 $C_2$，即得到一对 $n$ 阶正交拉丁方。

例如　利用构造 $n=P^t(n\geqslant 3)$ 阶正交拉丁方的规则，得出二对正交拉丁方

$$A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

$$B_1=\begin{bmatrix}1&2&3&4&5\\2&3&4&5&1\\3&4&5&1&2\\4&5&1&2&3\\5&1&2&3&4\end{bmatrix}\qquad B_2=\begin{bmatrix}1&2&3&4&5\\3&4&5&1&2\\5&1&2&3&4\\2&3&4&5&1\\4&5&1&2&3\end{bmatrix}$$

用 $A_1$ 和 $B_1$ 构造一个 15×15 的拉丁方 $C_1$ 的方法：

把 $C_1$ 分成 9 块，每一块是 5×5 的方阵，用 $A_1$ 的元素来标记对应块

$$C_1=\left[\begin{array}{c:c:c}1&2&3\\\hdashline 2&3&1\\\hdashline 3&1&2\end{array}\right]$$

然后把方阵 $B_1$ 填在 $C_1$ 每一块中，但是 $B_1$ 的元素在 $C_1$ 中变成了偶对，偶对的第一个分量是块标记，第二个分量是原 $B_1$ 的元素。例如 $C_1$ 左下角把标记为 3 的块变成

$$\begin{bmatrix}(3,1)&(3,2)&(3,3)&(3,4)&(3,5)\\(3,2)&(3,3)&(3,4)&(3,5)&(3,1)\\(3,3)&(3,4)&(3,5)&(3,1)&(3,2)\\(3,4)&(3,5)&(3,1)&(3,2)&(3,3)\\(3,5)&(3,1)&(3,2)&(3,3)&(3,4)\end{bmatrix}$$

最后用数字 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 分别代替序偶(1,1),(1,2),(1,3),(1,4),(1,5),(2,1),(2,2),(2,3),(2,4),(2,5),(3,1),(3,2),(3,3),(3,4),(3,5),就得到拉丁方 $C_1$

$$C_1=\begin{bmatrix}1&2&3&4&5&6&7&8&9&10&11&12&13&14&15\\2&3&4&5&1&7&8&9&10&6&12&13&14&15&11\\3&4&5&1&2&8&9&10&6&7&13&14&15&11&12\\4&5&1&2&3&9&10&6&7&8&14&15&11&12&13\\5&1&2&3&4&10&6&7&8&9&15&11&12&13&14\\6&7&8&9&10&11&12&13&14&15&1&2&3&4&5\\7&8&9&10&6&12&13&14&15&11&2&3&4&5&1\\8&9&10&6&7&13&14&15&11&12&3&4&5&1&2\\9&10&6&7&8&14&15&11&12&13&4&5&1&2&3\\10&6&7&8&9&15&11&12&13&14&5&1&2&3&4\\11&12&13&14&15&1&2&3&4&5&6&7&8&9&10\\12&13&14&15&11&2&3&4&5&1&7&8&9&10&6\\13&14&15&11&12&3&4&5&1&2&8&9&10&6&7\\14&15&11&12&13&4&5&1&2&3&9&10&6&7&8\\15&11&12&13&14&5&1&2&3&4&10&6&7&8&9\end{bmatrix}$$

用类似的方法，由 $A_2$ 和 $B_2$ 构造出另一个 15 阶拉丁方 $C_2$

$$C_2=\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15\\
3 & 4 & 5 & 1 & 2 & 8 & 9 & 10 & 6 & 7 & 13 & 14 & 15 & 11 & 12\\
5 & 1 & 2 & 3 & 4 & 10 & 6 & 7 & 8 & 9 & 15 & 11 & 12 & 13 & 14\\
2 & 3 & 4 & 5 & 1 & 7 & 8 & 9 & 10 & 6 & 12 & 13 & 14 & 15 & 11\\
4 & 5 & 1 & 2 & 3 & 9 & 10 & 6 & 7 & 8 & 14 & 15 & 11 & 12 & 13\\
11 & 12 & 13 & 14 & 15 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10\\
13 & 14 & 15 & 11 & 12 & 3 & 4 & 5 & 1 & 2 & 8 & 9 & 10 & 6 & 7\\
15 & 11 & 12 & 13 & 14 & 5 & 1 & 2 & 3 & 4 & 10 & 6 & 7 & 8 & 9\\
12 & 13 & 14 & 15 & 11 & 2 & 3 & 4 & 5 & 1 & 7 & 8 & 9 & 10 & 6\\
14 & 15 & 11 & 12 & 13 & 4 & 5 & 1 & 2 & 3 & 9 & 10 & 6 & 7 & 8\\
6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 1 & 2 & 3 & 4 & 5\\
8 & 9 & 10 & 6 & 7 & 13 & 14 & 15 & 11 & 12 & 3 & 4 & 5 & 1 & 2\\
10 & 6 & 7 & 8 & 9 & 15 & 11 & 12 & 13 & 14 & 5 & 1 & 2 & 3 & 4\\
7 & 8 & 9 & 10 & 6 & 12 & 13 & 14 & 15 & 11 & 2 & 3 & 4 & 5 & 1\\
9 & 10 & 6 & 7 & 8 & 14 & 15 & 11 & 12 & 13 & 4 & 5 & 1 & 2 & 3
\end{bmatrix}$$

可以看出，$C_1$ 和 $C_2$ 是正交的。

最后，我们给出构造一对奇数阶正交拉丁方的程序。

```
program zheng_jiao;

uses
  crt;

const
  maxn                = 50;

type
  ftype               = array [1..maxn, 1..maxn] of integer;{ 拉丁方 }
  fftype              = array [1..2] of ftype;          { 一对正交拉丁方 }

var
  a,b,c               : fftype;  { 三对正交拉丁方 }
  an, bn, cn,                 { 及它们的阶数 }
  n                   : integer; { 阶数 }

procedure init;
  begin
    clrscr;
    repeat write('n='); { 输入阶数 n }
           readln(n);
    until (n>2) and (n<=maxn) and (odd(n));
  end;

procedure make_b1_b2(k:integer); { 构造一对 k 阶正交拉丁方 b, k 为质数 }
  var t,i,j,w:integer;
  begin
```

```
    bn:=k;
    for t:=1 to 2 do
      begin
        w:=1;{ 正交拉丁方左上角元素为 1 }
        for i:=1 to k do{ 逐行构造 }
          begin
            for j:=1 to k do{ 逐列构造该行各元素 }
              begin
                b[t,i,j]:=w;
                inc(w);
                if w>k then dec(w,k);
              end;
            inc(w,t);
            if w>k then dec(w,k);
          end;
      end;
  end;

procedure combine;{ 由 an 和 bn 阶拉丁方得出 an * bn 阶拉丁方 }
  var t,i,j,p,q:integer;
  begin
    cn:=an * bn;{ 存储合并后的阶数 }
    for t:=1 to 2 do
      for i:=1 to an do
        for j:=1 to an do
          for p:=1 to bn do
            for q:=1 to bn do
              c[t,(i-1) * bn+p,(j-1) * bn+q]:=(a[t,i,j]-1) * bn+b[t,p,q];
  end;

procedure main;
  var k:integer;
  begin
    an:=1;  a[1,1,1]:=1; a[2,1,1]:=1;
    repeat k:=2;
           repeat inc(k); until (n mod k=0);{ 求 n 的质因数 k }
           n:=n div k;
           make_b1_b2(k);{ 求 k 阶正交拉丁方 b1,b2}
           combine;          { 将 b1,b2 与 a1,a2 合并成 Cn=k * an 阶正交拉丁方 }
           a:=c; an:=cn;  { 保留合并结果 }
    until n=1;{ 直至 n 阶正交拉丁方构造完毕 }
  end;

procedure show;{ 显示一对 n 阶正交拉丁方 a1 和 a2 }
  var k,i,j:integer;
  begin
    for k:=1 to 2 do
      begin
        writeln('A',k,'=');
        for i:=1 to an do
          begin
            for j:=1 to an do
              write(a[k,i,j]:3);
```

```
            writeln;
          end;
      end;
  end;

begin
  init;{ 输入阶数 }
  main;{ 构造 n 阶拉丁方 }
  show;{ 显示结果 }
end.
```

## 习 题 八

1. 题图 8-1 为一个 3 阶的魔六边形。问能否把数 1,2,…,19 填入其中每一格子,使所有的 15 行上数字之和都相等(个数不等)?还存在其它阶的魔六边形吗?

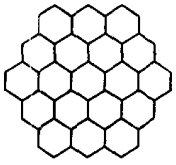


图 8-1

2. 构造一个 $n\times n$ 的拉丁方,并将其补足为一个 $m\times m(m>n)$的拉丁方。

3. 现有车胎 1,2,3,4,刹车闸垫 1,2,3,4。按下述格式填写两表(题表 8-1 为车胎磨损的试验设计,题表 8-2 为刹车闸垫的试验设计):

**题表 8-1**

| | A 号车 | B 号车 | C 号车 | D 号车 |
|---|---|---|---|---|
| 左前轮 | 填入车胎牌号 1,2,3,4 | | | |
| 右前轮 | | | | |
| 左后轮 | | | | |
| 右后轮 | | | | |

**题表 8-2**

| | A 号车 | B 号车 | C 号车 | D 号车 |
|---|---|---|---|---|
| 左前轮 | 填入刹车闸垫牌号 1,2,3,4 | | | |
| 右前轮 | | | | |
| 左后轮 | | | | |
| 右后轮 | | | | |

为了消除试验偏差、顾及两个试验的综合效果,要求两张表中,每个牌号的车胎(刹车闸垫)在每辆车上一次并在每个位置上一次,并且刹车闸垫与车胎磨损的试验设计不重复。

# 第九章　线性规划

## 9.1　线性规划及其数学模型

### 一、线性规划问题的数学模型

数学模型是描述现实世界的一个抽象，从而有助于解决这个被抽象的实际问题，而且起着指导解决其它具有这些共性的实际问题的作用。当我们用线性规划求解一个实际问题时，需要把这个实际问题用适当的数学形式表达出来，这个表达的过程，就是建立数学模型的过程。

在建立数学模型的过程中，首先要明确哪些是变量，哪些是已给出的常数，用字母来表示变量、数字及其它符号表示常数，然后将实际问题中的一些规律或关系，用数学表达式来加以描述。

[例 1]　某校办工厂有 $M_1,M_2,M_3$ 三种机器，生产 $P_1,P_2,P_3$ 三种产品，每单位产品所需要的机器时数和每种机器一周内开动的总时数如表 9-1 如示：

表 9-1

| 产品机器 | $P_1$ | $P_2$ | $P_3$ | 机器 $M_i$ 一周内开动的总时数 |
|---|---|---|---|---|
| $M_1$ | 4 | 5 | 3 | 12 |
| $M_2$ | 3 | 4 | 2 | 10 |
| $M_3$ | 4 | 2 | 1 | 8 |

设 $P_1$ 产品的单位利润为 6，$P_2$ 产品的单位利润为 4，$P_3$ 产品的单位利润为 3。假定生产这三种产品所用的机器没有先后次序之分，试问在一周内如何安排生产，才能使所获得的利润达到最大？

**解**：上述问题可以用以下数学模型来描述：

设：$P_1,P_2,P_3$ 三种产品每周产量为 $X_1,X_2,X_3$，显然产量不可能为负值，$X_i\geqslant 0(i=1,2,3)$。因为每种机器每周开动时间有限，这是一个限制产量的条件，所以在确定产品 $P_1,P_2,P_3$ 的产量时，要考虑生产这些产品所需要的机器时效应在允许的范围内。即用不等式表示为：

$$\begin{aligned} 4X_1+5X_2+3X_3 &\leqslant 12\\ 3X_1+4X_2+2X_3 &\leqslant 10\\ 4X_1+2X_2+X_3 &\leqslant 8\\ X_1,X_2,X_3 &\geqslant 0 \end{aligned}$$

在满足上面条件(称之为约束条件)下，使利润 $Z$ 达到最大。即

$$\max Z = 6X_1 + 4X_2 + 3X_3$$

其中 max $Z$——表示要求最大值；$Z$——称为目标函数。

从上面例子中，我们可以看到线性规划问题的数学模式中所具有的三个基本要素：

(1) 确定“决策变量”$X_i$。例如上题中第 $i$ 种产品每周的产量 $X_i(i=1,2,3)$；

(2) 确定决策变量可能受到的约束，称为约束条件，例如上题中表示每台机器机时限制的三个不等式；

(3) 在满足约束条件的前提下，使某个函数值最大或最小(例如求上题中的最大利润)。由于使该函数(决策变量的函数)为最大或最小正是我们在线性规划中所追求的目标，因此又称该函数为目标函数。

## 二、线性规划问题的一般形式

线性规划的一般形式是

目标函数：

$$\max Z = C_1X_1 + C_2X_2 + \cdots + C_nX_n = \sum_{i=1}^{n} C_iX_i$$

或

$$\min Z = C_1X_1 + C_2X_2 + \cdots + C_nX_n = \sum_{i=1}^{n} C_iX_i$$

约束条件：

$$\sum_{i=1}^{n} a_{1i}X_i = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1n}X_n \leqslant b_1 \quad (\text{或} = b_1, \text{或} \geqslant b_1)$$

$$\vdots$$

$$\vdots$$

$$\sum_{i=1}^{n} a_{mi}X_i = a_{m1}X_1 + a_{m2}X_2 + \cdots + a_{mn}X_n \leqslant b_m \quad (\text{或} = b_m, \text{或} \geqslant b_m)$$

$$X_1, X_2, \cdots, X_n \geqslant 0$$

在一般形式里我们看到，不论约束条件还是目标函数，都是决策变量的线性函数，这就是线性规划名称的由来。另外，我们假定决策变量是非负实数，至于决策变量是非负整数的线性规划(又称整数规划)，我们将在 9.4 节中给出。

## 三、线性规划问题的标准形式

一般线性规划问题有不同的形式。为了方便线性规划问题求解，有必要用一种统一的标准形式表示出来。下面是定义什么是线性规划问题的标准形式，介绍如何将非标准形式化为标准形式的线性规划问题。

具有 $m$ 个约束条件和 $n$ 个决策变量的线性规划问题的标准形式是：

目标函数：

$$\max Z = \sum_{i=1}^{n} C_iX_i = C_1X_1 + C_2X_2 + \cdots + C_nX_n$$

约束条件：

$$\sum_{i=1}^{n} a_{1i}X_i = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1n}X_n = b_1$$

$$\vdots$$

$$\vdots$$

$$\sum_{i=1}^{n} a_{mi}X_i = a_{m1}X_1 + a_{m2}X_2 + \cdots + a_{mn}X_n = b_m$$

其中 $X_1, X_2, \cdots, X_n \geqslant 0$ ； $b_1, b_2, \cdots, b_m \geqslant 0$

标准形式有四大特点：

(1) 目标函数属于最大化类型；

(2) 约束条件全部由等式表示；

(3) 决策变量必须取非负值；

(4) 每一约束条件里的常数是非负的。

那么，非标准形式的线性规划问题如何转化为标准形式的线性规划问题呢？其步骤是：

1. 如果目标函数是最小化类型：$\min Z = \sum_{i=1}^{n} C_i X_i$，那么只要将目标函数乘以(−1)，即可化为等价的最大化问题. 即：

$$\max Z = (-1) * \sum_{i=1}^{n} C_i X_i$$

2. 如果第 $j(1 \leqslant j \leqslant m)$ 个约束条件是小于或等于形式，即 $\sum_{i=1}^{n} a_{ji}X_i \leqslant b_j$，那么在不等式的左边加上一个非负的新变量 $W_j$，使它变成一个等式。即：

$$\sum_{i=1}^{n} a_{ji}X_i + W_j = b_j \quad W_j \geqslant 0$$

新增加的非负变量 $W_j$ 称为松弛变量；

如果约束条件是大于或等于形式，即 $\sum_{i=1}^{n} a_{ji}X_i \geqslant b_j$，那么在不等式左边减去一个非负的新变量 $W_j$，使它变成一个等式。即

$$\sum_{i=1}^{n} a_{ji}X_i - W_j = b_j \quad W_j \geqslant 0$$

新增加的非负变量 $Wj$ 称为剩余变量，也可称为松弛变量；

3. 如果决策变量有非正约束(如 $X_j \leqslant 0$)，则用非负变量 $X'_j$ 代替，使 $X_j = -X'_j$，$X'_j \geqslant 0$；如果决策变量符号不受限制，可用两个非负性约束的新变量 $X'_j$ 和 $X''_j$ 的差来代替，使 $X_j = X'_j - X''_j (X'_j, X''_j \geqslant 0)$；如果决策变量有上、下界 $a \leqslant X_j \leqslant b$，可引进新变量 $X'_j$ 等于原变量 $X_j$ 减去下限值 $a$：

$X'_j = X_j - a$ 代替 $X_j$，则 $0 \leqslant X'_j \leqslant b - a$

注：即新增加一个约束 $X'_j \leqslant b - a$。

4. 如果约束条件右边常数 $b_i$ 非正 $(b_i \leqslant 0)$，$\sum_{j=0}^{n} a_{ij}X_j \leqslant (\geqslant) b_i$，可将不等式两边同乘以(−1)，同时把不等号反向，即

$$(-1)*\sum_{j=1}^{n}a_{ij}X_j \geqslant (\leqslant)(-1)*b_i$$

5. 对于约束条件不全是“≤”不等式类型的线性规划问题，先化成标准形式，然后对于原是“≥”或“=”的约束条件加入人工变量 $W_k$，并在目标函数中增加'$-M*W_k$' 这一项($M$ 是一个很大的正数)。

[**例 2**] 将下列线性规划问题化成标准形式：

(1) 目标函数：$\max Z=4X_1+5X_2$

约束条件：$3X_1+5X_2\leqslant 24$

$4X_1+2X_2\leqslant 16$

$X_1+X_2\geqslant 3$

$X_1,X_2\geqslant 0$

**解**：先依上述方法化为标准型：

目标函数：$\max Z=4X_1+5X_2$

约束条件：$3X_1+5X_2+X_3=24$

$4X_1+2X_2+X_4=16$

$X_1+X_2-X_5=3$

$X_1,X_2,X_3,X_4,X_5\geqslant 0$

由于原约束条件中不全是'≤'，所以对原约束条件 $X_1+X_2\geqslant 3$ 加入人工变量 $X_6$，将之修改成 $X_1+X_2-X_5+X_6=3$

并在目标函数是增加 $-M\cdot X_6$ 这一项，得 $\max Z=4X_1+5X_2-MX_6$($M$ 为一个很大的正数)

最后形成标准型

目标函数：$\max Z=4X_1+5X_2-MX_6$

约束条件：$3X_1+5X_2+X_3=24$

$4X_1+2X_2+X_4=16$

$X_1+X_2-X_5+X_6=3$

$X_1,X_2,\cdots,X_6\geqslant 0$

(2) 目标函数：$\min Z=4X_1+6X_2$

约束条件：$4X_1+12X_2\leqslant -3$

$3X_1+6X_2=4$

$X_1\geqslant 0$，$X_2$ 为任意实数。

**解**：目标函数乘以(−1)，化成最大化问题；

因为 $X_2$ 符号不限，以 $X_2'-X_2''=X_2$ 代入目标函数及所有的约束条件，且 $X_2',X_2''\geqslant 0$；

对第一个约束条件两边乘以(−1)，不等号反向，减去剩余变量 $X_3$；

对第二个约束条件加入人工变量 $W_k$，并且目标函数中增加($-MW_k$)一项；

得到标准形式

目标函数：$\max Z=-4X_1'-6X_2'+6X_2''+(-MW_k)$

约束条件：$-4X_1-12X_2'+12X_2''-X_3=3$

$$3X_1+6X_2'-6X_2''+W_k=4$$

$X_1, X_2', X_2'', X_3 \geqslant 0$，$W_k$ 是一个很大的正数

## 四、用代数法求解引出的问题

无论用哪种方法，求出的决策变量 $X_1, X_2, \cdots, X_n$ 的值若满足约束条件，则称 $X_1, X_2, \cdots, X_n$ 为可行解。可行解可能不止一个，其中能使目标函数达到最大的称为最优解，或称线性规划的解。线性规划对应的目标函数值称为最优值或称线性规则的值。

[例 3] 下面，我们试用代数的方法求线性规划问题

目标函数：$\max Z=40X_1+60X_2$

约束条件：$X_1+X_2 \leqslant 5$

$X_1+2X_2 \leqslant 8$

$X_1, X_2 \geqslant 0$

的解。

首先将上述问题化为标准形式

目标函数：$\max Z=40X_1+60X_2$

约束条件：$X_1+X_2+X_3=5$

$X_1+2X_2+X_4=8$

$X_1, X_2, X_3, X_4 \geqslant 0$

约束条件的系数矩阵为

$$\begin{array}{c} \\ \text{约束条件 1} \\ \text{约束条件 2} \end{array}\begin{array}{c} \begin{array}{cccc} X_1 & X_2 & X_3 & X_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix} \end{array} = A$$

将约束条件两个方程变形为

$$X_3 = 5 - X_1 - X_2$$

$$X_4 = 8 - X_1 - 2X_2$$

其中 $X_1, X_2$ 为自由变量，给它们任一组解，可求得方程组的一组解。令 $X_1=X_2=0$，可得 $X_3=5, X_4=8$，从相对系数矩阵来看，划去第一列和第二列，得到 $A$ 的一个子矩阵

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

其中 $|B|=1$，称之为基矩阵，称基矩阵所在列对应的变量集合 $X_B=(X_3, X_4)$ 为基变量；称其余变量的集合 $X_N=(X_1, X_2)$ 为非基变量。显然，基变量 $X_B$ 和非基变量 $X_N$ 的值满足约束条件，因此

$$X_1=0, X_2=0, X_3=5, X_4=8$$

是原问题的可行解。

由于方程组含有四个决策变量，二个约束方程，可以从中任选两个为基变量，6 组解分别是

1：$X_1=0$，$X_2=0$，$X_3=5$，$X_4=8$，$Z=0$

2：$X_1=2$，$X_2=3$，$X_3=0$，$X_4=0$，$Z=260$

3：$X_1=8$，$X_2=0$，$X_3=-3$，$X_4=0$

4：$X_1=5$，$X_2=0$，$X_3=0$，$X_4=3$，$Z=200$

5：$X_1=0$，$X_2=5$，$X_3=0$，$X_4=-2$

6：$X_1=0$，$X_2=4$，$X_3=1$，$X_4=0$，$Z=240$

其中解 3 和解 5 是非可行解，其余 4 个为可行解。$X_1=2, X_2=3, X_3=X_4=0$ 为最优解，其对应的目标函数的最优值为 260。

下面根据一般情况给出有关概念：

设线性规划问题有 $m$ 个约束条件，$n$ 个决策变量，则约束方程的系数矩阵 $A$ 是 $m*n$ 的矩阵。由系数矩阵 $A$ 中任意 $m$ 列所组成的 $m\times n$ 非奇异子矩阵 $B$(即 $|B|<\geqslant 0$)称为该线性规划问题的基矩阵。与基矩阵 $B$ 的 $m$ 列向量对应的 $m$ 个变量称为基变量，基变量集合用 $X_B$ 表示。其余 $n-m$ 个变量称为非基变量，非基变量集合用 $X_N$ 表示。如果令 $X_N=0$，则由于 $|B|<>0$，$m$ 个方程一定可求出 $X_B$ 个解。又如果 $X_B>0$，即解适合非负条件，则称为基本可行解，否则称为不可行。对给出可行解的基矩阵称之为可行基。如果基变量 $X_B$ 中至少有一个变量为 0，称此基本可行解为退化的。

一般来说，若有 $n$ 个决策变量，$m$ 个方程的线性规划问题的最优解可以通过解 $C(n, m)$ 个联立方程组得到。但这个方法是不行的，因为：

(1) 可能的基本解太多了。如果 $m=5, n=15$，那就必须解 $C(15,5)=3003$ 个联立方程组；

(2) 有许多解是不可行的。上例中 6 个可能的解中只有 4 个是可行的；

(3) 在计算中，目标函数是被动的，因为只是当所有基本可行解被确定之后，它才用来计算函数值，选择其中最大者。

下面介绍的就是专门为避免这些缺陷而设计的单纯形法。

## 9.2 单纯形法

### 一、算法思想

单纯形法的基本思想是：

首先选择一个可行解，计算相应的目标函数值，根据一定的判别条件，可确定它是否为最优解。若不是，转换到另一个可行解，并使目标函数的值逐渐增大。当目标函数达到最大值时(即当 $Z$ 值不可能再增加)，最优解被找到。

一个基本可行解转换到另一个可行解的方法是转轴运算。规定一个基变量为 0，变成非基变量，把另一个非基变量变成基变量。

保证能产生这样一系列解的单纯形法的基础是需要满足两个基本条件：

1. 最优性条件　　使得不会碰到更差的解(相对于目前解而言)；

2. 可行性条件　　保证从一个可行解出发，在计算中只会碰到可行解。

下面通过例子介绍方法。

**[例 1]** 解下列线性规划问题

目标函数：　　$\max Z=6X_1+4X_2+3X_3$

约束条件：

$$\begin{aligned}&4X_1+5X_2+3X_3\leqslant 12\\&3X_1+4X_2+2X_3\leqslant 10\\&4X_1+2X_2+X_3\leqslant 8\\&X_1,X_2,X_3\geqslant 0\end{aligned}$$

的解。

**解**：引入松弛变量 $X_4,X_5,X_6$，将它变成标准形式

$$\left.\begin{aligned}&\text{目标函数：}\max Z=6X_1+4X_2+3X_3+0*X_4+0*X_5+0*X_6\\&\text{约束条件}\quad 4X_1+5X_2+3X_3+X_4=12\\&\qquad\qquad 3X_1+4X_2+2X_3+X_5=10\\&\qquad\qquad 4X_1+2X_2+X_3+X_6=8\\&\qquad\qquad X_1,X_2,\cdots,X_6\geqslant 0\end{aligned}\right\}\qquad (9\text{-}2\text{-}1)$$

约束方法的系数矩阵为

$$A=\begin{array}{c}\begin{matrix}X_1 & X_2 & X_3 & X_4 & X_5 & X_6\end{matrix}\\\begin{bmatrix}4 & 5 & 3 & 1 & 0 & 0\\3 & 4 & 2 & 0 & 1 & 0\\4 & 2 & 1 & 0 & 0 & 1\end{bmatrix}\end{array}$$

显然，$X_4,X_5,X_6$ 对应的子矩阵 $B$ 是一个 $3*3$ 的单位矩阵

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$

以 $B$ 作为可行基，将(9-2-1)中的每个等式移项得

$$\left.\begin{aligned}X_4&=12-4X_1-5X_2-3X_3\\X_5&=10-3X_1-4X_2-2X_3\\X_6&=8-4X_1-2X_2-X_3\end{aligned}\right\}\qquad (9\text{-}2\text{-}2)$$

令非基变量 $X_1=X_2=X_3=0$，由(9-2-2)可以得出，基变量 $X_4=12,X_5=10,X_6=8$。显然，非基变量和基变量可以组成一个可行解(0,0,0,12,10,8)。

这时目标函数

$Z=6X_1+4X_2+3X_3=0$，显然这个解是最差的。

下一步是要确定一个能改进目标函数值的新的可行解。单纯形法是挑选一个目前的非基变量，把它增大到大于零，只要它在目标函数中的系数有可能改进 $Z$ 值。因为一个可行解必须有 $m$ 个(本例为 3)取值为零的非基变量，所以目前的基变量必须有一个成为零，变成非基变量，使得新的解是可行的。变成基变量的目前的非基变量称为进基变量，它是由最优性条件决定的；变成非基变量的目前的基变量称为出基变量，它是由可行性条件确定的。

从本例看，目标函数中 $X_1$ 的系数最大，选 $X_1$ 为进基变量可使 $Z$ 增加最快。对于进基变量 $X_1$，解每个约束条件：

$$X_1=\frac{12}{4}-\frac{5}{4}X_2-\frac{3}{4}X_3-\frac{1}{4}X_4$$

$$X_1 = \frac{10}{3} - \frac{4}{3}X_2 - \frac{2}{3}X_3 - \frac{1}{3}X_5$$

$$X_1 = \frac{3}{4} - \frac{2}{4}X_2 - \frac{1}{4}X_3 - \frac{1}{4}X_6$$

由于 $X_2, X_3$ 是非基变量，它们取值为 0，所以方程又缩减为

$$X_1 = \frac{12}{4} - \frac{1}{4}X_4$$

$$X_1 = \frac{10}{3} - \frac{1}{3}X_5$$

$$X_1 = \frac{8}{4} - \frac{1}{4}X_6$$

那么在目前基变量($X_4, X_5, X_6$)中选哪一个变量出基呢？

如果从基中移去 $X_4$，则 $X_4=0$，得 $X_1=3$

如果从基中移去 $X_5$，则 $X_5=0$，得 $X_1=\frac{10}{3}$

如果从基中移去 $X_6$，则 $X_6=0$，得 $X_1=2$。

但如果 $X_1=3\left(\text{或 } X_1=\frac{10}{3}\right)$ 代入 $X_1=2-\frac{1}{4}X_6$ 中导致 $X_6<0$，这是不可行的。所以 $X_4$ 或 $X_5$ 不能从基中移去。如果 $X_1=2$，则 $X_4, X_5$ 均大于 0，所以 $X_6$ 能变成非基变量，而且可以发现 $X_1=\min\left(\frac{12}{4}, \frac{10}{3}, \frac{8}{4}\right)=2$，所在的方程 $X_1=\frac{8}{4}-\frac{1}{4}X_6$，所以 $X_6$ 出基。

既然我们已经确定 $X_1$ 进入基成为基变量而 $X_6$ 变成非基变量，那就必须通过转轴运算，将系数矩阵 $A$

$$\begin{bmatrix} 4 & 5 & 3 & 1 & 0 & 0 \\ 3 & 4 & 2 & 0 & 1 & 0 \\ 4 & 2 & 1 & 0 & 0 & 1 \end{bmatrix}$$

转换成矩阵

$$\begin{array}{c} X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6 \quad b \\ \begin{bmatrix} 0 & 3 & 2 & 1 & 0 & -1 & 4 \\ 0 & 5/2 & 5/4 & 0 & 1 & -3/4 & 4 \\ 1 & 1/2 & 1/4 & 0 & 0 & 1/4 & 2 \end{bmatrix} \end{array}$$

令 $X_2=X_3=X_6=0$，则 $X_1=2, X_4=4, X_5=4$

$$Z = 2 * 6 = 12$$

上述过程我们实质上进行了一次单纯形法的迭代。

为了取得最优解，我们不断地用新的可行解取代前面的可行解。每次取代后都使相应的目标函数值更佳。由于每次取代的步骤相同，因而可将过程表格化，使演算更紧凑、更有效。

为了使单纯形法计算容易，把原始数据整理在特殊表格 9-2 中。为了编表，把目标函数表示成以下方程形式：

$$-Z + 6X_1 + 4X_2 + 3X_3 + 0X_4 + 0X_5 + 0X_6 = 0$$

写在表 9-2 的最后一行

**表 9-2**

| 基变量 | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $b$ |
|---|---|---|---|---|---|---|---|
| $X_4$ | 4 | 5 | 3 | 1 | 0 | 0 | 12 |
| $X_5$ | 3 | 4 | 2 | 0 | 1 | 0 | 10 |
| $X_6$ | 4 | 2 | 1 | 0 | 0 | 1 | 8 |
| $-Z$ | 6 | 4 | 3 | 0 | 0 | 0 | 0 |

其中：

“基变量栏”标出当前基变量 $X_4, X_5, X_6$；

“$b$ 栏”给出基变量目前的值，即 $X_4=12, X_5=10, X_6=8$；

$-Z$ 行最后一项是目标函数当前值的相反数；

其余各行对应一个约束条件；

表 9-2 有两个特点：

(1) 表中约束条件方程的系数矩阵 $A$ 有单位子矩阵 $B$；

(2) 在 $-Z$ 行中基变量所在列的值是 0，即目标函数是非基变量的表达式。

满足上述两个特点的表格，称为单纯形表，单纯形表所对应的线性规划方程组，称为典范式线性方程组。应该引起注意的是，单纯形法必须从初始可行解开始迭代，开始的表格就是第一个单纯形表。单纯形表的重要性还在于，单纯形法产生一系列单纯形表，每个表对应一个比以前更好的可行解。

由于现在 $Z$ 的值是 0，所以接下来利用此表进行迭代。首先在 $-Z$ 行选出非基变量列的正值最大者，表中是 6，于是该列的非基变量 $X_1$ 成为入基变量，在 6 下面用“↑”表示入基。入基变量所在的列称为主元列。

我们称 $-Z$ 行中非基变量列的值为检验数。对于最大化条件，根据最优化条件，检验数是正的，解才有可能改进。

在 $b$ 列右边增加一列 $Q$，将 $b$ 列各值除以入基变量 $X_1$ 对应的正系数填写在 $Q$ 列，构成表 9-3。最小比(8/4)所在行称为主元行。与主元列的交叉元 4 称之为主元。主元行所在的基变量 $X_6$ 成为出基变量，该列下方用“↓”表示出基。

下面利用转轴运算，使入基变量 $X_1$ 所在列

$$\begin{bmatrix}4\\3\\4\\6\end{bmatrix} \xrightarrow{\text{变为}} \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

其步骤如下：

(1) 将主元行除以 4，得到新行(1,1/2,1/4,0,0,1/4,2)

**表 9-3**

主元列 ↓

| 基变量 | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $b$ | $Q$ |
|---|---|---|---|---|---|---|---|---|
| $X_4$ | 4 | 5 | 3 | 1 | 0 | 0 | 12 | (12/4) |
| $X_5$ | 3 | 4 | 2 | 0 | 1 | 0 | 10 | (10/3) |
| $X_6$ | 4 主元 | 2 | 1 | 0 | 0 | 1 | 8 | (8/4) ←主元行 |
| $-Z$ | 6 | 4 | 3 | 0 | 0 | 0 | 0 | 0 |

↑ 可行解(0,0,0,1,1,1)；　　↓ $Z=0$

(2) 将新行乘以(−4)加到 $X_4$ 行得(0,3,2,1,0,−1,4)

将新行乘以(−3)加到 $X_5$ 行得(0,5/2,5/4,0,1,−3/4,4)

将新行乘以(−6)加到 $-Z$ 行得(0,1,3/2,0,0,−3/2,−12)

这些计算结果构成新的表 9-4，这也是一张单纯形表，由此可得第二个可行解(2,0,0,4,4,0)，目标函数 $Z=12$。由于 $-Z$ 行还出现正项，即非基变量 $X_2$，$X_2$ 的系数是正的，说明目标函数还可能增大，所以表 9-4 不一定是最优解，再进行迭代。

**表 9-4**

主元行 ↓

| 基变量 | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $b$ | $Q$ |
|---|---|---|---|---|---|---|---|---|
| $X_4$ | 0 | 3 | 2 主元 | 1 | 0 | −1 | 4 | 4/2 ←主元行 |
| $X_5$ | 0 | 5/2 | 5/4 | 0 | 1 | −3/4 | 4 | 16/5 |
| $X_1$ | 1 | 1/2 | 1/4 | 0 | 0 | 1/4 | 2 | 8 |
| $-Z$ | 0 | 1 | 3/2 | 0 | 0 | −3/2 | −12 | |

↑ ↓

同上可以找到 $X_3$ 为入基变量，$Q$ 列中(4/2)为最小比，它所在行为主元行，主元为 2，主元行中的基变量 $X_4$ 为出基变量，利用转轴运算，使得 $X_3$ 所在列

$$\begin{bmatrix} 2 \\ (5/4) \\ (1/4) \\ (3/2) \end{bmatrix} \xrightarrow{\text{变为}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

得到表 9-5。

**表 9-5**

| 基变量 | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $b$ | $Q$ |
|---|---|---|---|---|---|---|---|---|
| $X_3$ | 0 | 3/2 | 1 | 1/2 | 0 | −1/2 | 2 | |
| $X_5$ | 0 | 5/8 | 0 | −5/8 | 1 | −1/8 | 3/2 | |
| $X_1$ | 1 | 1/8 | 0 | −1/8 | 0 | 3/8 | 3/2 | |
| $-Z$ | 0 | −5/4 | 0 | −3/4 | 0 | − 3/4 | −15 | |

这也是一张单纯形表，可得到第三个可行解。由于$-Z$行中检验数全部是非正项，因此表格表示最优解为(3/2,0,2,0,3/2,0)，最优值$Z=15$。

下面简述在应用单纯形法时经常遇到的问题，以及针对这类问题的作法与结论：

(1) 非基变量的检验数为 0 的情况

一般地说，在最优表里如果存在检验数为零的非基变量，那么就存在多个最优解的情况，称为多重最优。反之，如果最优表中所有非基变量的检验数都为负数，则最优表中给出的最优解是唯一的。例如表 9-5 中的$-Z$行，非基变量检验数$X_2=-(5/4)$，$X_4=-(3/4)$，$X_6=-(3/4)$，因此最优解唯一。

(2) 检验数相同

选取哪一个非基变量进基，一般是按检验数大小确定的。如果有一个以上的检验数$C_j$相同，一般任选一个非基变量进基。选不同的非基变量进基的差别仅仅是迭代次数不同，而这预先无法预测。选择哪个非基变量进基无妨最优性的要求。

(3) 最小比值相同

选取哪一个基变量出基是应用最小比规则确定的。如果遇到多个约束条件给出相等的比值的特殊情况，一般任选一个基变量出基。选相同比值的变量出基的结果是以后迭代时可能产生退化的可行解(即一个或多个基变量为 0 的可行解)。在退化情况下，可能出现无穷无尽迭代下去的死循环状态。

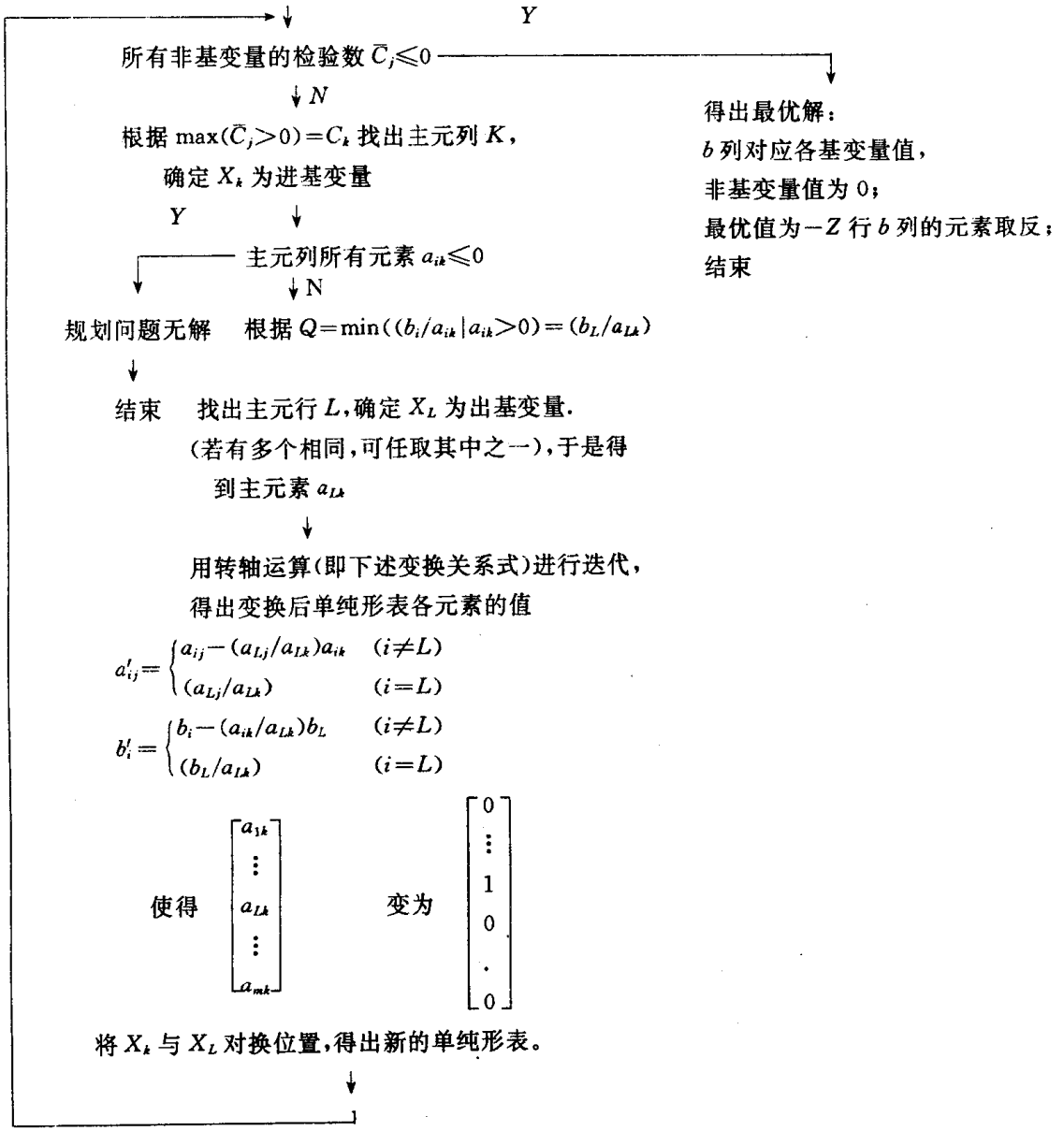
但是，纵然许多实际问题具有退化解，但死循环却是非常罕见的，因此你尽可任意选择具有相同最小比值的变量出基，不必去理会理论上的退化和循环后果。

## 二、算法流程

单纯形法的计算步骤归纳如下：

化线性规则问题为标准形式；

建立初始单纯形表，这是一张典型表。找出初始可行解；




## 三、单纯形法程序

```
Program Dan_Chun_Xing_Fa;

  uses crt;

  const
    max_c       =   10000;   { 大 M }
    maxn        =      20;   { 单纯形表规模 }

  var
    a           :  array [1..maxn,0..maxn] of real; { 单纯形表 }
    c           :  array [0..maxn] of real;
```

```
                { c[i]——目标函数中 xi 的系数 }
  x             : array [1..maxn] of integer;
                { x[i]——基变量 xi 的值 }
  m,n,row,col,  { 约束条件个数、变量个数、单纯形表的行数和列数 }
  k,l           : integer;
        { k——入基变量序号，l——出基变量在单纯形表中的行号 }

procedure init;
  var
    i,j,t,r:integer;
  begin
    clrscr;
    t:=0;
    fillchar(a,sizeof(a),0);fillchar(c,sizeof(c),0);
    { 单纯形表和目标函数初始化 }
    write('N M = ');readln(n,m); { 输入变量个数和约束条件个数 }
    col:=n;
    for i:=1 to n do   { 输入目标函数系数 }
      begin
        write('c[',i,']= ');readln(c[i])
      end;
    for i:=1 to m do   { 输入约束条件 }
      begin
        for j:=1 to n do
          begin
            write('a[',i,',',j,']= ');readln(a[i,j])
          end;
        repeat
          write('<=,=,>= (1,2,3)');readln(r)
        until (r in [1..3]);
        write('b[',i,']= ');readln(a[i,0]);
        if a[i,0]<0
           then begin
                  for j:=0 to n do a[i,j]:=-a[i,j];
                  r:=4-r
                end;
        case r of
          1 : begin
                inc(col);a[i,col]:=1;
                inc(t);x[t]:=col
              end;
          2 : begin
                inc(col);a[i,col]:=1;c[col]:=-max_c;
                inc(t);x[t]:=col
              end;
          3 : begin
                inc(col);a[i,col]:=-1;
                inc(col);a[i,col]:=1;c[col]:=-max_c;
                inc(t);x[t]:=col
```

```
                end
            end
          end;
        for i:=0 to col do { 计算检验数 }
          begin
            a[m+1,i]:=0;
            for j:=1 to m do
              a[m+1,i]:=a[m+1,i]+c[x[j]]*a[j,i];
            a[m+1,i]:=c[i]-a[m+1,i]
          end
      end;

    function max_k:boolean;
  { 若 max_k 返回 false，则表示没有正的检查数,得出最佳解,否则找出入基变量 Xk }
      var
        i:integer;
        min:real;
      begin
        k:=0;min:=0;
        for i:=1 to col do
          if (a[m+1,i]>0) and (a[m+1,i]>min)
             then begin
                    min:=a[m+1,i];k:=i
                  end;
        if k>0 then max_k:=true
               else max_k:=false
      end;

    function min_l:boolean;
    { 若 min_l 返回 false,则表示所有 a[i,k]≤0,此题无可行解;}
    { 否则找出出基变量 Xl }
      var
        i:integer;
        theata:real;
      begin
        theata:=max_c;l:=0;
        for i:=1 to m do
          if (a[i,k]>0) and (a[i,0]/a[i,k]<theata)
             then begin
                    l:=i;theata:=a[i,0]/a[i,k]
                  end;
        if l<>0 then min_l:=true
                else min_l:=false
      end;

    procedure proceed; { 以 a[l,k]为主元素进行迭代 }
      var
        i,j:integer;
        r:real;
```

```
  begin
    r:=a[l,k];
    for i:=0 to col do a[l,i]:=a[l,i]/r;
    for i:=1 to m+1 do
      if i<>l
        then begin
               r:=a[i,k];
               for j:=0 to col do a[i,j]:=a[i,j]-r*a[l,j]
             end;
    x[l]:=k
  end;

procedure print;{ 打印结果 }
  var
    i:integer;
    r:real;
  begin
    writeln(' ':4,' ':10);
    r:=0;
    for i:=1 to m do
      if x[i]<=n
        then writeln(x[i]:4,a[i,0]:10);
    writeln('-----------------------');
    writeln('max z = ',-a[m+1,0]);
    readln
  end;

begin
  init;{ 输入目标函数和约束条件,设置单纯形表 }
  while max_k do { 若找出入基变量,则循环 }
    if min_l      { 若找出出基变量,则以主元素进行迭代 }
      then proceed
      else begin { 否则无解退出 }
             writeln('No solution');readln;
             halt
           end;
  print { 打印最佳结果 }
end.
```

## 9.3 对偶问题

在线性规划问题中,不论从理论方面还是从实际问题方面,都存在一个有趣的问题,这就是对于任何一个最大化的线性规划问题,必有一个含相同数据的最小化规划问题与之相配,反之亦然。如果称前者为原始问题,那么后者便称为对偶问题;或者称后者为原始问题,那么前者即称为对偶问题。两者互为对偶规划问题。

提出对偶规划问题的意义在于:

(1) 在原始和对偶两个线性规划中求解任一个规划时，会自动地给出另一个规划的最优解；

(2) 在某些情况下可利用对偶关系，简化线性规划问题的计算。

## 一、引例

下面通过例子来说明每一个线性规划问题总是伴随着另一个称之为对偶问题的线性规划问题。

[例 1] 某工厂生产甲乙两种产品，每生产一种产品，需要三种资源，资源的限量及每种产品的收益如表 9-6 所示。

表 9-6

| | 资源 $A$ | 资源 $B$ | 资源 $C$ | 每台收益 |
|---|---|---|---|---|
| 产品甲 | 3 | 4 | 2 | 5 |
| 产品乙 | 4 | 2 | 1 | 4 |
| 资源限量 | 14 | 8 | 6 | |

如何安排生产，使获得利润最大？

设产品甲，乙，分别生产 $X_1, X_2$ 个，问题可用最大化线性规划描述如下：

线性规划问题 1：

目标函数：$\max Z=5X_1+4X_2$

约束条件：$3X_1+4X_2 \leqslant 14$

$4X_1+2X_2 \leqslant 8$

$2X_1+X_2 \leqslant 6$

$X_1, X_2 \geqslant 0$

假如工厂考虑不进行生产而把全部可利用资源都转让给其它单位。希望制定一个合理的价格，既能使本厂获得最大收益，又能使别的单位愿意购买。设出让三种资源的售价为 $Y_1, Y_2, Y_3$。

原来生产产品甲每台需要用的资源按现在的单价计算，每台的收益不低于原来生产一台可得到的收益。即

$$3Y_1 + 4Y_2 + 2Y_3 \geqslant 5$$

同样，对产品乙也有相应的约束条件

$$4Y_1 + 2Y_2 + Y_3 \geqslant 4$$

总售价尽可能低，即目标函数为

$$\min D = 14Y_1 + 8Y_2 + 6Y_3$$

于是可得如下线性规划问题 2：

目标函数：$\min D=14Y_1+8Y_2+6Y_3$

约束条件：$3Y_1+4Y_2+2Y_3 \geqslant 5$

$$4Y_1+Yy_2+Y_3\geqslant 4$$

$$Y_1,Y_2,Y_3\geqslant 0$$

工厂决策者所面临的这两个问题有一定的内在联系。事实上，这两个问题的最优值是相同的。

以上从直观的角度引出两个线性规划问题，前者称为原始线性规划问题，后者就是对偶规划问题。当然并不是所有的对偶问题都是可以加以直观解释的。

对比这两个线性规划，可以看出原始问题与对偶问题之间存在着下列关系：

(1) 原始问题求最大化，对偶问题求最小化；

(2) 原始问题每个约束条件是“≤”不等式，而对偶问题每个约束条件是“≥”不等式；

(3) 一个问题中的每一个约束条件对应于另一个问题中的变量；

例如原始问题中限制 $A$ 资源使用的 $3X_1+4X_2\leqslant 14$，对应于对偶问题中出让 $A$ 资源的售价 $y_1$。同样原始问题的约束条件 2 对应于对偶问题的 $Y_2$。

(4) 一个问题中的约束条件系数矩阵是另一个问题的约束条件系数矩阵的转置；

例如 原始问题约束条件系数矩阵 $A=\begin{bmatrix}3 & 4\\ 4 & 2\\ 2 & 1\end{bmatrix}$　　对偶问题约束条件的系数矩阵 $B=\begin{bmatrix}3 & 4 & 2\\ 4 & 2 & 1\end{bmatrix}$

(5) 两个问题的变量都是非负的。

例如：原始问题中的 $X_1$，$X_2\geqslant 0$；对偶问题中的 $Y_1,Y_2,Y_3\geqslant 0$。

## 二、对偶关系的一般描述

把上述对偶关系一般化，可写为

| 原始问题 | 对偶问题 |
| --- | --- |
| 目标函数：$\max Z=\sum_{i=1}^{n}c_iX_i$ | 目标函数：$\min D=\sum_{i=1}^{m}b_iY_i$ |
| 约束条件：$\sum_{i=1}^{n}a_{1i}X_i\leqslant b_1$ | 约束条件：$\sum_{i=1}^{m}a_{i1}Y_i\geqslant c_1$ |
| …… | …… |
| $\sum_{i=1}^{n}a_{mi}X_i\leqslant b_m$ | $\sum_{i=1}^{m}a_{in}Y_i\geqslant c_n$ |
| $X_1,X_2,\cdots,X_n\geqslant 0$ | $Y_1,Y_2,\cdots,Y_m\geqslant 0$ |

以上线性规划问题称为具有对称形式。

[**例 2**] 写出下列原始线性规划问题的对偶问题

目标函数：$\max Z=2X_1+3X_2$

约束条件：$2X_1+2X_2\leqslant 12$

$$X_1+2X_2\leqslant 8$$

$$4X_1\leqslant 16$$

$$4X_2\leqslant 12$$

$$X_1,X_2\geqslant 0$$

**解**：对偶问题如下

目标函数：$\min D=12Y_1+8Y_2+16Y_3+12Y_4$

约束条件：$2Y_1+2Y_2+4Y_3\geqslant 2$

$2Y_1+2Y_2+4Y_3\geqslant 3$

$Y_1,Y_2,Y_3,Y_4\geqslant 0$

如果原始线性规划问题不具备对称形式，那么设法创造条件使之变成上述形式，然后写出它的对偶问题。方法如下：

(1) 如果原始问题的约束条件中的有一个等式，则在对偶问题的条件中用一对“≤”与“≥”不等式替换原等式，即把

$$\sum_{j=1}^{n}a_{ij}X_j=b_i\rightarrow\begin{cases}\sum_{j=1}^{n}a_{ij}\leqslant b_j\\ \sum_{j=1}^{n}a_{ij}\geqslant b_i\end{cases}$$

(2) 若原始问题中 $X_i$ 是符号不受限制的自由变量，则令 $X_i=X_i'-X_i''$ 代入各式，这时 $X'_i,X''_i\geqslant 0$，然后再转换为对偶式；

(3) 对于原始问题中每个约束条件规定一个变量，作为对偶式的决策变量；

(4) 若原始问题求最大化，则把约束条件中的“≥”不等式全部化成“≤”不等式，即用(−1)乘“≥”不等式两边，同时规定对偶问题求最小化，且全部约束条件具有“≥”类型不等式；

若原始问题求最小化，则把约束条件中的“≤”不等式全部化成“≥”不等式，即用(−1)乘“≤”不等式两边，同时规定对偶问题求最大化，且全部约束条件具有“≤”类型不等式；

(5) 规定对偶问题中的约束条件右边的常数向量是原始问题中目标函数系数向量的转置；目标函数系数向量是原始问题中约束条件右边的常数向量的转置；约束条件系数矩阵是原始问题约束条件系数矩阵的转置。

[**例 3**] 求下列线性规划问题的对偶问题

目标函数：$\max Z=5X_1+3X_2+2X_3+4X_4$

约束条件：$5X_1+X_2+X_3+8X_4=10$

$2X_1+4X_2+3X_3+2X_4=10$

$X_1,X_2,X_3,X_4\geqslant 0$

**解**：先将上述规划问题的等式约束条件写成“≤”不等式的约束条件

目标函数：$\max Z=5X_1+3X_2+2X_3+4X_4$

约束条件：$5X_1+X_2+X_3+8X_4\leqslant 10$

$-5X_1-X_2-X_3-8X_4\leqslant -10$

$2X_1+4X_2+3X_3+2X_4\leqslant 10$

$-2X_1-4X_2-3X_3-2X_4\leqslant -10$

$X_1,X_2,X_3,X_4\geqslant 0$

四个约束条件定义四个对偶变量 $Y_1,Y_2,Y_3,Y_4$

目标函数：$\min D=10Y_1-10Y_2+10Y_3-10Y_4$

约束条件：$5Y_1-5Y_2+2Y_3-2Y_4\geqslant 5$

$Y_1-Y_2+4Y_3-4Y_4\geqslant 3$

$Y_1-Y_2+3Y_3-3Y_4\geqslant 2$

$8Y_1-8Y_2+2Y_3-2Y_4\geqslant 4$

$Y_1,Y_2,Y_3,Y_4\geqslant 0$

可能读者发现这两个线性规划问题的对偶关系似乎不像前面所说的那样有规律。其实只要将 $Y'=Y_1-Y_2$，$Y''=Y_3-Y_4$ 代入上面的约束条件和目标函数，就可得

目标函数：$\min D=10Y'+10Y''$

约束条件：$5Y'+2Y''\geqslant 5$

$Y'+4Y''\geqslant 3$

$Y'+3Y''\geqslant 2$

$8Y'+2Y''\geqslant 4$

$Y'$，$Y''$无约束

这样一来，最后的线性规划问题是原问题的对偶问题了。但可能又发现原始问题的约束条件是等式而对偶问题的约束条件不是等式。原始问题的变量≥0，而对偶问题的变量不受约束。关于线性规划的原始问题与对偶问题的关系，其变换形式可以归纳为表 9-7 的形式。

表 9-7

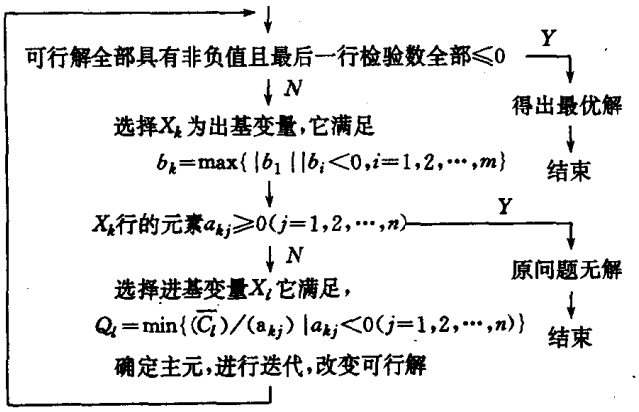
| 原始问题(或对偶问题) | 对偶问题(或原始问题) |
|---|---|
| 目标函数 max $Z$ | 目标函数 min $D$ |
| 约束条件 $m$ 个 | 对偶变量 $m$ 个 |
| 第 $i$ 个约束条件为≤ | 对偶变量 $Y_i\geqslant 0$ |
| 第 $i$ 个约束条件为= | 对偶变量 $Y_i$ 是自由变量 |
| 变量 $X_i$ 数为 $n$ 个 | 约束条件数为 $n$ 个 |
| 变量 $X_j\geqslant 0$ | 第 $j$ 个约束条件≥0 形式 |
| 变量 $X_j$ 为自由变量 | 第 $j$ 个约束条件为=形式 |

## 三、对偶单纯形法的算法和程序

对偶单纯形法的计算步骤：

输入原始问题的线性规划，并将之转换为相应的对偶问题；

用(−1)乘具有负系数的松弛变量的每个约束条件两边，列出初始的对偶单纯形表。初始基是松弛变量的集合。




（注：若此表的检验数不含≤0，则不能用对偶单纯形法）

最后，给出对偶单纯形法的程序。

```
Program Dui_Ou_Dan_Chun_Xing_Fa;
  { 对偶单纯形法 }

  uses crt;

  const
    max_c       =  10000; { 大 M }
    nmax        =  20;    { 对偶单纯形表规模 }

  var
    dl          :  array [1..nmax,0..nmax] of real;
                   { 对偶单纯形表 }
    c           :  array [1..nmax] of real;
                   { c[i]——目标函数中 Xi 的系数 }
    x           :  array [1..nmax] of integer;
                   { X[i]——基变量 Xi 的值 }
    n,m,row,col,   { 变量个数,约束条件个数,对偶单纯形表的行数和列数 }
    l,k         :  integer;
                   { 出基变量在对偶单纯形表的行号,入基变量序号 }

  procedure init; {初始化}
    var
      i,j,t:integer;
    begin
      clrscr;
      fillchar(dl,sizeof(dl),0); { 对偶单纯形表初始化 }
      write('变量个数,条件个数 N M = ');readln(n,m);
      row:=m;col:=n;t:=0;
      for i:=1 to n do              { 读入目标函数系数 }
        begin
          write('c[',i,']= ');readln(c[i]);
          c[i]:=-c[i]
```

```
      end;
    for i:=1 to m do                    { 读入约束条件 }
      begin
        for j:=1 to n do
          begin
            write('a[',i,',',j,'] = ');readln(dl[i,j]);
            dl[i,j]:=-dl[i,j]
          end;
        repeat
          write('<=,>=,=(1,2,3) ');readln(j)
        until (j>0) and (j<=3);
        case j of
          2 : begin
                inc(col);dl[i,col]:=1;c[col]:=0;
                inc(t);X[t]:=col
              end;
          1 : begin
                inc(col);dl[i,col]:=1;c[col]:=-max_c;
                inc(t);X[t]:=col;
                inc(col);dl[i,col]:=-1;c[col]:=0
              end;
          3 : begin
                inc(col);dl[i,col]:=1;c[col]:=-max_c;
                inc(t);X[t]:=col
              end
        end;
        write('b[',i,']= ');readln(dl[i,0]);
        dl[i,0]:=-dl[i,0]
      end;
    for i:=0 to col do                  { 计算检验数 }
      begin
        dl[m+1,i]:=0;
        for j:=1 to m do
          dl[m+1,i]:=dl[m+1,i]+c[X[j]]*dl[j,i];
        dl[m+1,i]:=c[i]-dl[m+1,i]
      end
  end;

function min_1:boolean;
  {确定出基变量,min_1=true,则 b 列的数字都为非负}
  {                 min_1=false,则 b 列的数字有正数  }
  var
    i:integer;
    m:real;
  begin
    m:=max_c;
    for i:=1 to row do
      if (dl[i,0]<0) and (dl[i,0]<m)
         then begin
```

```
                m:=dl[i,0];l:=i;
              end;
    if m<0 then min_l:=false
           else min_l:=true
  end;

procedure min_k;
  {确定入基变量}
  var
    i:integer;
    theata:real;
  begin
    theata:=max_c;k:=0;
    for i:=1 to col do
      if (dl[l,i]<0) and (dl[row+1,i]/dl[l,i]<theata)
         then begin
                k:=i;theata:=dl[row+1,i]/dl[l,i]
              end
  end;

function check:boolean;
  {检查检验数是否保持负数}
  var
    i:integer;
  begin
    for i:=1 to col do
      if dl[row+1,i]>0
         then begin
                check:=false;exit
              end;
    check:=true
  end;

procedure proceed;
  {以 a[l,k]为主元素,在表中进行迭代运算,得到新的计算表}
  var
    i,j:integer;
    t:real;
  begin
    t:=dl[l,k];
    for i:=0 to col do dl[l,i]:=dl[l,i]/t;
    for i:=1 to row+1 do
      if i<>l
         then begin
                t:=dl[i,k];
                for j:=0 to col do dl[i,j]:=dl[i,j]-t*dl[l,j]
              end;
    X[l]:=k
  end;
```

```
procedure print;  {输出对偶问题的最优解}
  var
    i:integer;
    r:real;
  begin
    writeln('变量':4,'最优解':10);
    for i:=1 to row do
      if X[i]<n
         then begin
                writeln(X[i]:4,dl[i,0]:10:2);
                r:=r+c[X[i]]*dl[i,0]
              end;
    writeln('————————————');

    writeln('max Z = ',-r:10:2);
    readln;halt
  end;

begin
  init;          { 输入约束条件和目标函数,设置对偶单纯形表 }
  while check do  { 若检验数全负,则循环 }
    if min_l       { 若 b 列的数全负,则打印最佳解 }
       then print
       else begin { 否则,计算入基变量 Xk }
              min_k;
              proceed { 在对偶单纯形表中进行迭代运算 }
            end;
  writeln('检验数中有正数,无法继续');
  readln
end.
```

## 9.4 整数规划

### 一、问题的提出

在前面讨论的单纯形法或对偶单纯形法问题中,有些最优解可能是小数或分数,但对某些具体问题,常有要求必须是整数的情况(称为整数解)。例如人数、车数或机器数等,小数或分数的解答就不合要求。为了满足整数解的要求,初看起来,似乎只要把得到的带有分数或小数的解经过"舍入化整"就可以了。但这常常是不行的,因为化整后不见得是可行解;或虽是可行解,但不一定是最优解。

[**例 1**] 某厂拟用集装箱托运甲、乙两种货物,每箱的体积、重量,可获利润以及托运所受限制如表 9-8 所示。

问:两种货物各托运多少箱,可使获得利润为最大?

**解**:设 $X_1$,$X_2$ 分别为甲、乙两种货物的托运箱数,这个规划问题,用数学式可表示为:

$$\max Z = 20X_1 + 10X_2$$

$$5X_1 + 4X_2 \leqslant 24$$
$$2X_1 + 5X_2 \leqslant 13$$
$$X_1X_2 \geqslant 0$$
$$X_1, X_2 \text{ 为整数}$$

表 9-8

| 货物 | 体积($m^3$/箱) | 重量(百斤/箱) | 利润(百元/箱) |
|---|---|---|---|
| 甲 | 5 | 2 | 20 |
| 乙 | 4 | 5 | 10 |
| 托运限制 | 24 | 13 | |

它和单纯形法的区别仅在于最后的条件:"$X_1, X_2$ 为整数。"现在我们暂不考虑这一条件,即解前四个式子(以后称这样的问题为和原问题相应的线性规划问题)很容易求得最优解为

$$X_1 = 4.8, \quad X_2 = 0, \quad \max Z = 96。$$

但 $X_1$ 是托运甲种货物的箱数,现在它不是整数,所以不合要求。

是不是可以把所得的非整数的最优解经过"化整"就可得到合乎要求的整数最优解呢?

如将 $X_1=4.8, X_2=0$ 凑整为 $X_1=5, X_2=0$,这样就破坏了条件"$5X_1+4X_2\leqslant 24$",因而它不是可行解;如舍去 $X_1$ 的尾数 0.8,变为 $X_1=4, X_2=0$,这当然满足各约束条件,因而是可行解,但不是最优解,因为

当 $X_1=4, X_2=0$ 时,$Z=80$。

但当 $X_1=4, X_2=1$(这也是可行解)时,$Z=90$。

由[例 1]看出,将其相应的线性规划的最优解"化整"来求最优整数解,虽是最容易想到的,但这个整数解常常不是最优的,甚至根本不是可行解。因此有必要对最优整数解的问题进行专门研究,我们称这样的问题为整数规划。

## 二、分枝定界解法

我们用另一种方法求解[例 1]——穷举变量的所有可行解的整数组合。[例 1]中的变量只有 $X_1$ 和 $X_2$,由条件"$5X_1+4X_2\leqslant 24$"得出 $X_1$ 所能取的整数值为 0,1,…,4,由"$2X_1+5X_2\leqslant 13$"得出 $X_2$ 所能取的整数值为 0,1,…,2,它的组合(不都是可行的)数是 $3\times 5=15$ 个。我们从这 15 个组合方案中找出可行解,然后比较它的目标函数值以定出最优解。显然,对于这一类变量数少,可行的整数组合数少的小型问题,穷举法是可行的,也是有效的。但是对于变量数多,可行的整数组合数多的大型问题,穷举法又如何呢?我们再来看一个例子:

[**例 2**] 有 $n$ 项任务指派给 $n$ 个人去完成,其中第 $i$ 个人去完成第 $j$ 项任务的效益为 $C_{ij}$ $(1\leqslant i, j\leqslant n)$

问应指派何人去完成何工作,使得产生的效益最大?

**解**:设

$$X_{ij}=\begin{cases}1 & \text{当指派第 } i \text{ 人去完成第 } j \text{ 项任务;}\\ 0 & \text{当不指派第 } i \text{ 人去完成第 } j \text{ 项任务。}\end{cases}$$

由于每项任务只能由 1 人去完成,因此

$$\sum_{i=1}^{n}X_{i1}=1;$$

$$\sum_{i=1}^{n}X_{i2}=1; \qquad \text{即} \quad \sum_{i}X_{ij}=1, \quad j=1,2,\cdots,n$$

$$\cdots\cdots\cdots$$

$$\sum_{i=1}^{n}X_{in}=1;$$

又由于每个人只能完成 1 项任务,因此

$$\sum_{j=1}^{n}X_{1j}=1;$$

$$\sum_{j=1}^{n}X_{2j}=1; \qquad \text{即} \quad \sum_{j}X_{ij}=1, i=1,2,\cdots,n$$

$$\cdots\cdots\cdots$$

$$\sum_{j=1}^{n}X_{nj}=1;$$

由此得出数学模型:

$$\max Z\sum_{i}\sum_{j}C_{ij}X_{ij}$$

$$\sum_{i}X_{ij}=1, j=1,2,\cdots,n$$

$$\sum_{j}X_{ij}=1, i=1,2,\cdots,n$$

$$X_{ij}=1 \quad \text{或} \quad 0$$

上述问题称为指派问题,显然它也是一种整数规划。将 $n$ 项任务指派 $n$ 个人去完成,不同的指派方案共有 $n!$ 种。当 $n=10$,这个数就超过三百万,当 $n=20$,这个数就超过 $2\times10^{18}$,如果一一计算,就是用每秒百万次的计算机,也要几万年的功夫。很显然,解这样的问题穷举法是不可取的(注:用整数规划解指派问题是不合算的,有关指派问题的专门解法,将在 9.5 节中给出)。所以优化的办法一般应是仅检查可行的整数组合的一部分,就能定出最优的整数解。分枝定界解法就是其中的一个。

设最大化整数规划问题 $A$,$A$ 的最优目标函数值记为 $Z^*$。略去 $A$ 中决策变量为整数的条件,产生相应的线性规划问题 $B$。从解问题 $B$ 开始,若其解不符合 $A$ 的整数条件,那么 $B$ 的最优目标函数值 $\overline{Z}\geqslant Z^*$,$Z$ 为 $A$ 的最优值的上界;而 $A$ 的任意可行解目标函数值 $\underline{Z}\leqslant Z^*$,$\underline{Z}$ 为 $A$ 的最优值的下界。分枝定界法就是将 $B$ 的可行解范围分解成子范围(称为分枝)的方法,逐步减少上界 $\overline{Z}$ 和增大下界 $\underline{Z}$,最终求得 $A$ 的最优值 $Z^*$。现举例说明:

[**例 3**] 求解 $A$

$$\text{整数规划问题 } A\begin{cases}\left.\begin{array}{l}\max = 40X_1 + 90X_2\\ 9X_1 + 7X_2 \leqslant 56\\ 7X_1 + 20X_2 \leqslant 70\end{array}\right\}\text{相应的线性规划问题 } B\\ X_1X_2 \geqslant 0\\ X_1, X_2 \quad \text{整数}\end{cases}$$

**解**：求出相应的线性规划问题 $B$ 的最优解，见图 9-1。

| 问题 $B$ |
|---|
| $X_1=4.81$ |
| $X_2=1.82$ |
| $Z_0=356$ |

图 9-1

虽然它不符合问题 $A$ 的整数条件，问题 $A$ 的最优值 $Z^*$ 的上界 $\overline{Z}=Z_0$，$Z^*$ 的下界 $\underline{Z}=0$（取问题 $A$ 的一个整数可行解 $X_1=X_2=0$ 时的函数值），即 $0\leqslant Z^*\leqslant 365$。

分枝定界法的解法，首先注意其中一个非整数变量的解，如 $X_1$，在问题 $B$ 的解中 $X_1=4.81$，于是将问题 $B$ 分解成两枝：

问题 $B$ 中增加约束条件 $X_1\leqslant 4$，形成子问题 $B_1$，

问题 $B$ 中增加约束条件 $X_2\geqslant 5$，形成子问题 $B_2$。

分别求 $B_3$ 和 $B_4$ 的最优解，见图 9-2。

显然没有得到全部变量是整数的解。因 $Z_1>Z_2$，故 $\overline{Z}=Z_1=349$，即 $0\leqslant Z^*\leqslant 349$。继续对问题 $B_1$ 和 $B_2$ 进行分解。因 $Z_1>Z_2$，故先分解 $B_1$ 为两枝：

问题 $B_1$ 中增加约束条件 $X_2\leqslant 2$，形成子问题 $B_3$；

问题 $B_1$ 中增加约束条件 $X_2\geqslant 3$，形成子问题 $B_4$。

分别求 $B_3$ 和 $B_4$ 的最优解，见图 9-3。

由于问题 $B_3$ 的解已都是整数，因此 $B_3$ 的最优值 $Z_3$ 可取为 $\underline{Z}$，即 $\underline{Z}=Z_3=340$。又因为 $Z_3>Z_4=327$，所以再分解 $B_4$ 已无必要。而问题 $B_2$ 的 $Z_2=341$，很明显，在 $340\leqslant Z^*\leqslant 341$ 范围内，问题 $A$ 有整数解。于是对 $B_2$ 分解：

问题 $B_2$ 中增加约束条件 $X_2\leqslant 1$，形成子问题 $B_5$；

问题 $B_2$ 中增加约束条件 $X_2\geqslant 2$，形成子问题 $B_6$；

$B_5$ 即非整数解且 $Z_5=308<\underline{Z}$，没有必要再分解下去，而问题 $B_6$ 为无可行解，不可再分解，于是可以断定

$$Z_3=\underline{Z}=Z^*=340$$

问题 $B_3$ 的解 $X_1=4, X_2=2$ 为最优整数解，见图 9-4。

从以上解题过程可得出用分枝定界法求解整数规划（最大化）问题的步骤为：

设问题 $A$——待求解的整数规划问题，$A$ 的最优目标函数值为 $Z^*$；

问题 $B$——与问题 $A$ 相应的线性规划问题，即略去 $A$ 问题中的整数条件；

$\underline{Z}$——问题 $A$ 的最优值 $Z^*$ 的下界，即目前符合整数条件的各分支结果中目标函

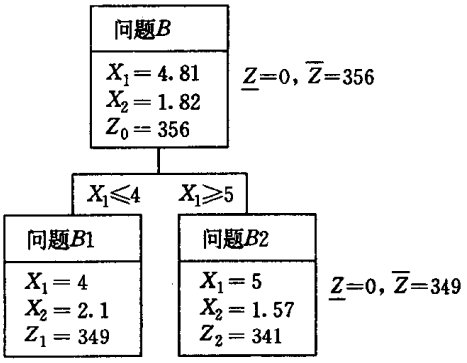



图 9-2

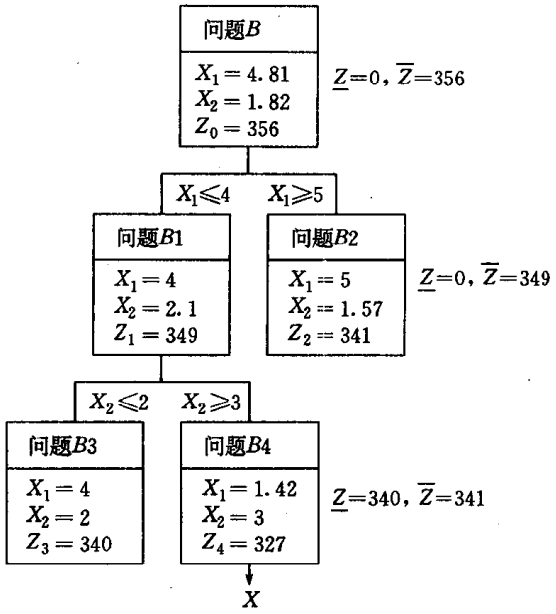



图 9-3

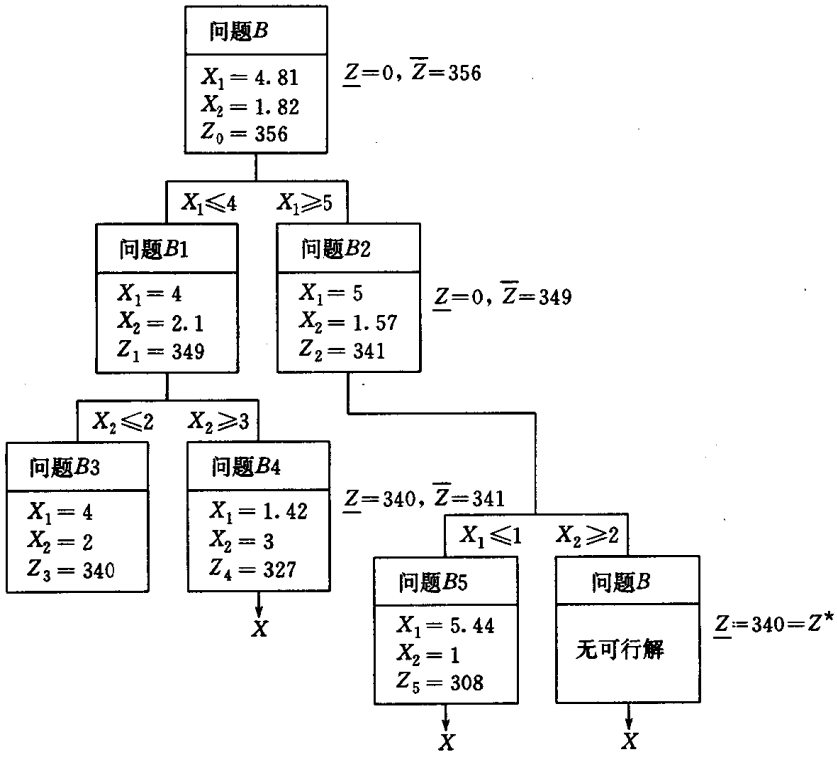
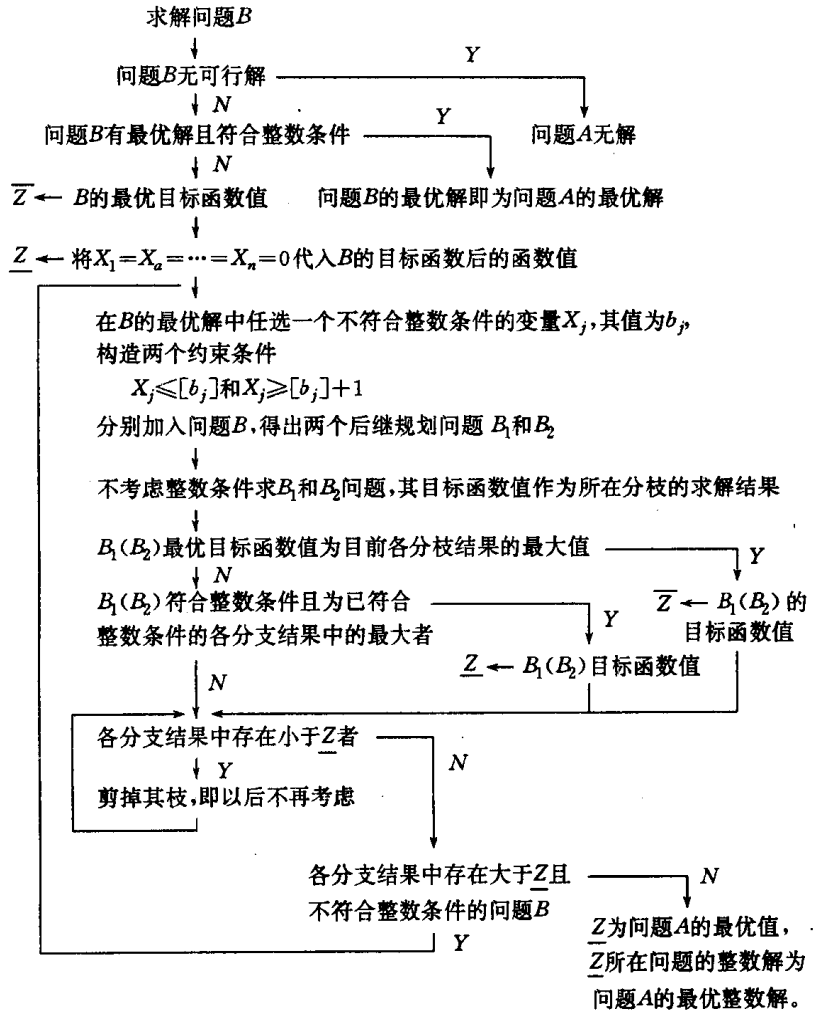



图 9-4

数值的最大者；

$\overline{Z}$——问题 $A$ 的最优值 $Z^*$ 的上界，即目前各分支结果中目标函数值的最大者。

显然，$\underline{Z} \leqslant Z^* \leqslant \overline{Z}$




用分枝定界法解整数规划问题比穷举法优越，因为它仅在一部分符合约束条件的整数解中寻求最优解，计算量比穷举法小。但若变量数目很大(例如指派问题有 $n \times n$ 个变量)，其计算工作量也是相当可观的。

最后，给出用分枝定界法解整数规划的程序：

```
Program Dan_Chun;
uses crt;
Const
  Max_c                 =   5000;{ 大M }
  Maxn                  =     40;{ 单纯形表规模 }


var
  a,ta                  :  array [0..maxn,0..maxn] of real;
{ a——单纯形表；ta——暂存单纯形表 }
  c,bx, xr              :  array [0..maxn] of real;
{ c——目标函数系数表；bx——暂存当前解；xr——当前解 }
  x, tx                 :  array [1..maxn] of integer;
```

```
{ x——基变量序号表; tx——取整处理后的基变量序号表 }
  m,n,col,
{ m——约束条件个数; n——变量个数; col——列序号 }
  k,l                    :  integer;
{ 入基和出基变量序号 }
  f                      :  text;
{ 文件变量 }
  z_max                  :  real;
{ 整数条件下各分支结果的最大者 }

procedure init;
  var
    i,j,t      :  integer;
  begin
    fillchar(a,sizeof(a),0); { 单纯形表初始化 }
    assign(f,'Inp. Txt');     { 文件名和文件变量连接 }
    reset(f);                  { 文件读准备 }
    Readln(f,n,m);             { 读入变量数和约束条件个数 }
    col:=n; t:=0;
    fillchar(c,sizeof(c),0); { 目标函数初始化 }
    for i:=1 to n do read(f,c[i]); { 输入目标函数 }
    for i:=1 to m do               { 输入约束条件 }
      begin
        for j:=0 to n do read(f,a[i,j]);
        inc(col); a[i,col]:=1;
        inc(t); x[t]:=col
      end;
    for i:=1 to m do tx[i] := i+n
  end;

function max_k:boolean;
{ 检查检验数是否全负,若全负,则返回 false,表示得出最佳解;否则找出入基变量 Xk,并检查 k 列
  元素是否全部非正。若是,则失败退出}
  var
    i:integer;
    min:real;
  begin
    k:=0; min:=0;
    for i:=1 to col do
      if (a[0,i]>0) and (a[0,i]>min)
          then begin
                 min:=a[0,i]; k:=i
               end;
    if k>0 then max_k := true
           else max_k := false;
    for i:=1 to m do
      if a[i,k] > 0 then exit;
    writeln('No ranged solution');
    halt
```

```
  end;

function min_l :  boolean;
{ 若 min_l 返回 false,则表示 k 列元素全部非正,无可行解;否则找出出基变量 Xl }
var
  i :integer;
  theata:real;
begin
  theata:=max_c; l:=0;
  for i:=1 to m do
    if (a[i,k]>0) and (a[i,0]/a[i,k]<theata)
       then begin
              l:=i; theata:=a[i,0]/a[i,k]
            end;
  if l<>0 then min_l :=true
          else min_l :=false
end;

procedure proceed; { 以 a[i,k]为主元素进行迭代 }
var
  i,j:integer;
  r:real;
begin
  r:=a[l,k];
  for i:=0 to col do a[l,i]:=a[l,i]/r;
  for i:=0 to m do
    if i<>l
       then begin
              r:=a[i,k];
              for j:=0 to col do a[i,j]:=a[i,j]-r * a[l,j]
            end;
  x[l]:=k
end;

procedure print; { 打印整数规划的解 }
  var
    i:integer;
    r:real;
  begin
    r:=0;
    for i:=1 to n do
      writeln('x',i,' ':4,bx[i]:4:0);
    writeln('Max z = ',z_max:10:4);
  end;

function dan:real; { 对当前线性规划进行求解,并返回解的目标函数值 }
var
  i,j: integer;
  r  :  real;
```

```
begin
  ta := a;
  x := tx;
  for i:=0 to col do
    begin
      a[0,i]:=0;
      for j:=1 to m do
        a[0,i]:=a[0,i]+c[x[j]]*a[j,i];
      a[0,i] := c[i]-a[0,i]
    end;
  while max_k do
    if min_l
       then proceed
       else begin
              dan := -max_c; exit
            end;
  r:=0;
  fillchar(xr,sizeof(xr),0);
  for i:=1 to m do xr[x[i]]:=a[i,0];
  dan:=-a[0,0];
  a := ta;
end;

procedure search;{ 利用分枝定界求解整数规划 }
var
  t : real;
  i,j,x:integer;
begin
  t := dan;{ 计算当前线性规划的目标函数值 }
  if t<=z_max then exit;
{ 若当前目标函数值非整数条件下各分支结果的最大者，则回溯 }
  i:=1;
  while (i<=n) and (trunc(xr[i]*10000)=trunc(xr[i])*10000) do inc(i);
  { 找出一个非整数的变量 Xi }
  if i>n
{ 若所有变量都是整数,则回溯(注：若目标函数值为目前最大，则记下其解) }
     then begin
            if t>z_max then begin
              z_max:=t; bx:=xr
            end;
            exit;
          end;
  x:=trunc(xr[i]);
  inc(m); inc(col);
  for j:=0 to col do a[m,j]:=0;
  a[m,0]:=x; a[m,i]:=1; a[m,col]:=1;
  tx[m] := col;
  search;  { 限制 Xi<TRUNC(XR[i])，进行搜索 }
  a[m,0]:=x+1;a[m,i]:=1; a[m,col]:=-1;
```

```
    inc(col); a[m,col]:=1; c[col]:=-max_c;
    tx[m] := col; a[0, col] := -max_c;
    search; { 限制 Xi>TRUNC(XR[i])+1, 进行搜索 }
    c[col]:=0;
    dec(m); dec(col, 2)
  end;

  begin
    init;        { 输入数据,建立单纯形表 }
    z_max:=0;    { 目标函数值初始化 }
    search;      { 用分枝定界法求解整数规划 }
    print        { 打印整数规划的解 }
  end.
```

## 9.5 指派问题

### 一、最佳匹配与最优指派

我们曾在 9.4 的[例 2]中给出了最大化指派问题的提法和数学模型。现在我们给出指派问题的算法。

构造一个图 $G$,$G$ 中的顶点分为两部分 $X$ 和 $Y$,$X$ 为工作人员的集合 $X=\{X_1,X_2,\cdots,X_n\}$,$Y$ 为任务的集合 $Y=\{Y_1,Y_2,\cdots,Y_n\}$,当且仅当 $X_i$ 适合于工作 $Y_j$ 时,$X_i$ 与 $Y_j$ 之间连一条边,边上的权为 $X_i$ 做 $Y_j$ 工作的效益。显然图 $G$ 为一个带权的二分图。

所谓最优指派实际上是在二分图 $G$ 上求一个满足下述条件的边集 $M$:

(1) 由于每项任务只能由 1 人完成,而每人只能承担一项任务,因此 $M$ 中的任意两条边没有公共端点,即 $M$ 是匹配;

(2) 由于 $n$ 个工作人员必须完成这 $n$ 项任务,因此 $G$ 中没有一个顶点不与 $M$ 中的边关联,即 $M$ 是完备匹配;

(3) 由于要求完成 $n$ 项任务所产生的效益最大,因此完备匹配 $M$ 中边上的权的总和必须最大,即 $M$ 是最佳匹配。

显然,最佳匹配 $M$ 与最优指派问题一一对应。

[**例 1**] 5 个工作人员$\{X_1,X_2,,\cdots,X_5\}$ 做 5 项工作$\{Y_1,Y_2,\cdots,Y_5\}$, 每人所做工作的效益用 $n\times n$ 的矩阵表示:

$$C=\begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{array}\begin{array}{c} \begin{array}{ccccc} Y_1 & Y_2 & Y_3 & Y_4 & Y_5 \end{array} \\ \begin{bmatrix} 3 & 5 & 5 & 4 & 1 \\ 2 & 2 & 0 & 2 & 2 \\ 2 & 4 & 4 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 3 & 3 \end{bmatrix} \end{array}$$

问指派何人做何项工作时,产生的总效益最大。

**解**: 效益矩阵 $C$ 对应一个二分图 $G$,见图 9-5。

那么,对二分图 $G$ 如何求最佳匹配,即最优指派的解呢?

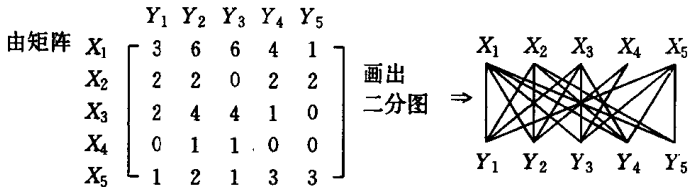



图 9-5

## 二、如何求完备匹配

所谓最佳匹配，无非是指所有完备匹配中权的总和最大的一种匹配。因此，我们首先必须知道如何求完备匹配，下面介绍一种利用增广轨求完备匹配的算法。

假设 $M$ 是图 $G$ 的一个匹配，则称 $M$ 中的边关联的顶点为盖点，$G$ 中其它的顶点为未盖点。若 $P$ 是 $G$ 中一条连通两个未盖点的路径，并且属于 $M$ 的边和不属 $M$ 的边在 $P$ 上交错出现，则称 $P$ 为相对于 $M$ 的一条增广轨。由此定义，我们可得三个推论：

(1) 一条增广轨$(V_1,V_2,\cdots,V_j)$的路径长度必为奇数，并且路径上关联 $V_1$ 的第一条边和关联 $V_j$ 的最后一条边都不属于 $M$；

(2) 假如将增广轨 $P$ 看作路径上所有边的集合，则可通过运算 $M' \leftarrow M \oplus P$ 得到更大的匹配 $M'$。其中⊕为集合的异或运算，即 $M'$ 包含属于 $M$ 或属于 $P$，但不同时在 $M$ 和 $P$ 中的边。换句话说，$P$ 中的匹配边改为非匹配边，非匹配边改为匹配边，$M$ 中除 $P$ 外的匹配边不变。这样，将 $M$ 改造为多包含一条边的匹配 $M'$；

(3) $M$ 为 $G$ 的完备匹配当且仅当不存在相对于 $M$ 的增广轨 $P$ 且 $G$ 中不含未盖点。

这三条推论告诉我们，在找到一条相对于 $M$ 的增广轨后，容易求得更大的匹配 $M'$。例如，从图 9.6 可以看出，$M$ 原含 3 条匹配边，一旦找到一条相对于 $M$ 的可增广轨 $P$，可通过 $M \oplus P$ 运算产生一个含 4 条边的匹配 $M'$。可见，关键的问题是如何求得相对于 $M$ 的一条可增广轨。

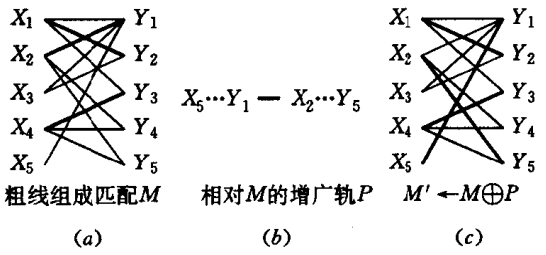



图 9-6

我们可以使用一个类似广度搜索的过程来逐层构造 $G$ 的一棵交错树 $T$。$T$ 的每一树枝上，非匹配边与匹配边交错出现，称为交错路。$T$ 树上偶层的顶点称为外点，标以‘+’。若 $V_j$ 是外点，则连接根与 $V_j$ 的交错路一定含偶数条边，且该路径上与 $V_j$ 相连的一定属于匹配边；$T$ 树上奇层的顶点称为内点，标以‘−’。若 $V_j$ 是内点，则连接根与 $V_j$ 的交错路

一定含奇数条边，且该路径上与 $V_j$ 相连的一定是非匹配边。

逐层扩展交错树 $T$ 的办法如下：

任选一个未盖点 $V_i$ 为 $T$ 的根，看看 $G$ 中有没有与 $T$ 的外点关联且不属于 $T$ 的边 $e$。如果有，就再看看 $e$ 的另一个端点 $V_k$ 是不是属于 $T$，如果 $V_k$ 不属于 $T$，就可以把 $e$ 和 $V_k$ 都加到 $T$ 中去，使 $T$ 扩大。具体扩大的时候，还可以分两种情况：

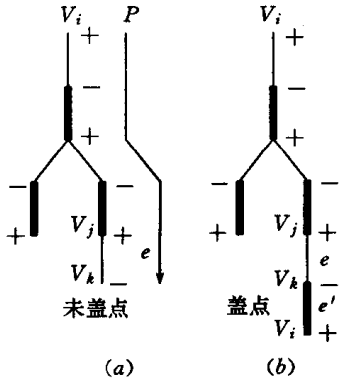



图 9.7

(1) $V_k$ 是未盖点，这时，把 $V_k$ 和 $e$ 加到 $T$ 中去，$T$ 中连接根 $V_i$ 与 $V_k$ 的交错路是一条可增广轨 $P$(见图 9.7($a$))；

(2) $V_k$ 不是未盖点，也就是说，有一条属于匹配 $M$ 的边 $e'$ 与 $V_k$ 关联，这时，在把 $e$ 与 $V_k$ 加到 $T$ 中去后，接着还可以把 $e'$ 以及它的另一个端点 $V_l$ 加到 $T$ 中去，因为很显然，从根 $V_i$ 也可以通过一条交错路到达 $V_l$。另外还容易看出 $V_k$ 应该是内点，而 $V_l$ 是外点(见图 9.7($b$))。

这样逐层扩展交错树 $T$，直至出现下列两种情况之一为止：

(1) 若奇数层上加入一个未盖点 $V$，则找出一条由根 $V_i$ 至 $V$ 的增广轨；

(2) 若与当前任一外点相连的边的端点都在内点集合里，或者说，所有与外点相连的非匹配边都属于 $T$，则需从一个新的未盖点开始，建立另一棵交错树。如果不存在其它未盖点，则说明不存在相应于 $M$ 的增广轨，不可能找出更多的匹配了。由于 $G$ 中还有未盖点 $V_i$，因此 $M$ 不是完备匹配。

例如，我们在图 9.6($c$)的基础上扩展交错树 $T$。首先，选择未盖点 $X_3$ 作为根，在第 1 层添加顶点 $Y_1$，$Y_2$ 和相应的非匹配边($X_3$，$Y_1$)，($X_3$，$Y_2$)，见图 9-8。

在第 2 层加入匹配边($Y_1$，$X_5$)、($Y_2$，$X_1$) 以及相连的顶点 $X_5$、$X_1$，见图 9-9。

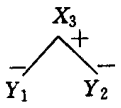



图 9-8

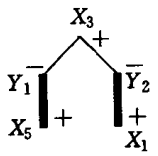



图 9-9

在第 3 层由于与外点 $X_5$ 相连的边的端点在 $T$ 的内点集合里且经非匹配边与 $X_1$ 相连的顶点 $Y_1$ 已在交错树 $T$ 上，故只能在 $T$ 上添加顶点 $Y_3$ 和非匹配边($X_1$，$Y_3$)。而 $Y_3$ 是盖点，因此在第 4 层加入 $X_4$ 和匹配边($Y_3$，$Y_4$)，见图 9-10。

在第 5 层加入新顶点 $Y_4$ 和 $Y_5$ 及相应的未匹配边($X_4$，$Y_4$)、($X_4$，$Y_5$)。由于这是奇数层，而 $Y_4$ 是未盖点，则交错树 $T$ 构造完毕，且得出交错路 $X_3-Y_2-X_1-Y_3-X_4-Y_4$，这是一条相对于匹配 $M'$ 的增广矩阵 $P'$，见图 9-11。

经过运算 $M''\leftarrow M'\oplus P'$，可得一个新的匹配 $M''$，显然 $M''$ 是一个完备匹配，见图

9-12。

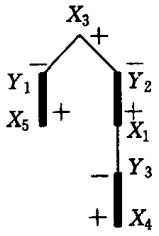



图 9-10

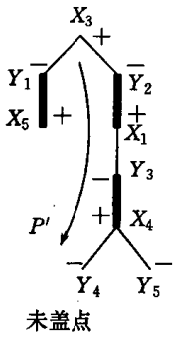



图 9-11

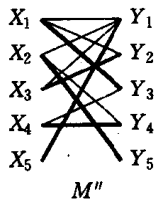



图 9-12

## 三、如何求最佳匹配

如何从二分图的所有完备匹配中寻找权的总和最大的最佳匹配呢？一种方法是求出所有完备匹配，然后从中比较权的总和最大的匹配。显然，这种穷举算法的运算量是相当大的。本节将给出另一种较简便的有效算法。在讲述算法之前，先引入一些概念，作为算法的基础。

### 1. 二分图 $G$ 的可行顶标 $L(u)$

对于 $X$ 集合中的任一顶点 $x$ 和 $Y$ 集合中的任一顶点 $y$，满足条件

$$L(x)+L(y)\geqslant W(xy)$$

则称 $L(u)$是 $G$ 的可行顶标。这个可行顶标是存在的，例如初始时可设

$$L(x)=\max_{y\in Y}W(xy) \qquad x\in X;$$

$$L(y)=0 \qquad y\in Y;$$

### 2. 相等子图 $G_L$

由边集 $E_L=\{xy \mid xy\in E$ 的边集 $E$，$L(x)+L(y)=W(xy)\}$组成的 $G$ 的生成子图，称作相等子图，记为 $G_L$。

为什么我们要给出上述两个概念，原因是欲求二分图 $G$ 的最佳匹配，只需用匈牙利算法求取相等子图 $G_L$ 的完备匹配，即为 $G$ 的最佳匹配。但问题是，当 $G_L$ 中无完备匹配时怎么办，下述算法给出修改顶标的一个方法，使新的相等子图的最大匹配逐渐扩大，最后出现相等子图的完备匹配。

变量说明

$L$——可行顶标集合；

$I$——修改后的顶标集合；

$G_L$——修改顶标后的相等子图；

$A_L$——顶标的改进量；

$S$——$G_L$ 的匹配中，外点的集合，初始时为交错树的根；

$T$——$G_L$ 的匹配中，内点的集合，初始时为空；

$N_{GL}(S)$——$G_L$ 中，与 $S$ 相邻的顶点集合。显然，若 $N_{GL}(S)=T$，$G_L$ 无完备匹配。

$M$——匹配边集合；

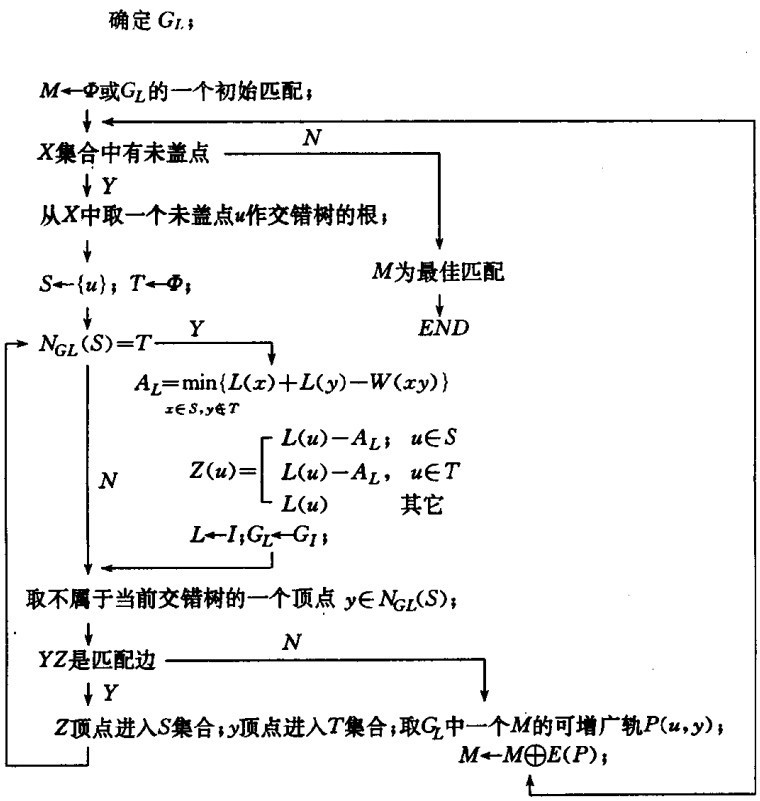
$E(P)$——可增广轨 $P$ 上的边；

$M\odot E(P)$——$M$ 与 $P$ 的边进行异或操作，即 $P$ 轨上的匹配边与非匹配边对换，$P$ 轨外的匹配边不变，使得 $M$ 集合增加一条匹配边。

下面给出求二分图的最佳匹配的算法流程：

选定初始的可行顶标 $L$：

$$L(x)=\max_{y\in Y}W(xy), \qquad x\in X$$
$$L(y)=0 \qquad y\in Y$$




例：已知 $K_{5,5}$的权矩阵为

$$G=\begin{array}{c}\\ X_1\\ X_2\\ X_3\\ X_4\\ X_5\end{array}\begin{array}{c}\begin{array}{ccccc}Y_1 & Y_2 & Y_3 & Y_4 & Y_5\end{array}\\ \begin{bmatrix}3 & 5 & 5 & 4 & 1\\ 2 & 2 & 0 & 2 & 2\\ 2 & 4 & 4 & 1 & 0\\ 0 & 1 & 1 & 0 & 0\\ 1 & 2 & 1 & 3 & 3\end{bmatrix}\end{array}$$

求最佳匹配，其中 $K_{5,5}$的顶点划分为 $X=\{X_i\}$，$Y=\{Y_i\}$，$i=1,2,\cdots,5$。

我们利用上述算法求解：

(1) 取初始的可行顶标 $L(u)$为：

$L(Y_i)=0 \quad i=1,2,\cdots,5$

$L(X_1)=\max\{3,5,5,4,1\}=5$ $\quad L(X_4)=\{0,1,1,0,0)=1$

$L(X_2)=\max\{2,2,0,2,2\}=2$ $\quad L(X_5)=\{1,2,1,3,3)=3$

$L(X_3)=\max\{2,4,4,1,0\}=4$

(2) 确定 $G_L$ 及其上的一个初始匹配 $M=\{X_1Y_2,X_2Y_1,X_3Y_3,X_5Y_5\}$，见图 9-13($a$)。

(3) 以 $u=X_4$ 为根，求交错树，得 $S=\{X_4,X_3,X_1\},T=\{Y_3,Y_2\}$。$N_{GL}(S)=T$，于是

$$A_l=\min_{x\in s,y\notin t}\{L(x)+L(y)-W(xy)\}=1$$

修改顶标：

$I(X_1)=4;$ $\quad I(Y_1)=0;$

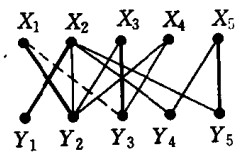
$I(X_2)=2;$ $\quad I(Y_2)=1;$

$I(X_3)=3;$ $\quad I(Y_3)=1;$

$I(X_4)=0;$ $\quad I(Y_4)=0;$

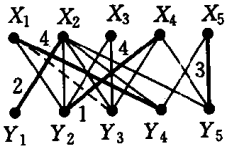
$I(X_5)=3;$ $\quad I(Y_5)=0$

(4) 用修改后的顶标 $I$ 得 $G_i$ 及其上面的一个完备匹配，见图 9-13($b$)。




($a$)

图 9-13($a$)




($b$)

图 9-13($b$)

图中粗实线给出了一个最佳匹配，其最大权是 2+4+1+4+3=14。

由上可以看出：$A_l>0$；修改后的顶标仍是可行顶标；$G_i$ 中仍含 $G_l$ 中的匹配 $M$，$G_i$ 中至少会出现不属于 $M$ 的一条边，所以会造成 $M$ 的逐渐增广。

最后，给出程序题解

```
Program Kuhn_Munkras_Algorithm;

  const
    maxn        =  30;

  type
    gtype       =  array [1..maxn,1..maxn] of integer; { 二分图类型 }
    ltype       =  array [1..maxn] of integer;         { 顶标类型 }
    settype     =  set of 1..maxn;

  var
    n,m         :  integer;       { |x|,|y| }

    g,gt        :  gtype;         { 图 G 矩阵,中间矩阵 }
    l           :  ltype;         { 图 G 矩阵顶标 }
    f           :  text;          { 文件变量 }
```

```
  s,t,cx,cy    :  settype;
  { 已检查的 x 集合,已检查的 y 集合,已匹配的 x 集合,已匹配的 y 集合 }

procedure read_graph;
  var
    str:string;
    i,j:integer;
  begin
    write('Graph file = '); { 读入文件名,并与文件变量连接 }
    readln(str);
    assign(f,str);
    reset(f);
    readln(f,n,m);          { 读入|x|和|y| }
    for i:=1 to n do        { 读入 Xi 做 Yj 工作的效益 }
      for j:=1 to m do read(f,g[i,j]);
    close(f);
  end;

function path(x:integer):boolean;
{ 求 x 出发的增广轨,若存在,则增广轨上的匹配连与非匹配边对换 }
  var
    i,j:integer;
  begin
    path:=false;
    for i:=1 to m do
      if not (i in t) and (g[x,i]<>0) { 若 i∈已检查的 Y 集合,且(X,i)∈E }
        then begin                    { i 进入已检查的 y 集合 }
               t:=t+[i];
               if not (i in cy)
                  { 若 i 不属于已匹配的 y 集合,则<x,i>置匹配边标志 }
                  { i 进入已匹配的 y 集合,返回 true }
                  then begin
                         g[x,i]:=-g[x,i];cy:=cy+[i];
                         path:=true;exit
                       end;
               j:=1;
               { 与 x 相邻的顶点 i 已匹配,搜索一条已 }
               { 匹配边<j,i>(j 不属于已检查的 x 集合) }
               while (j<=m) and not (j in s) and (g[j,i]>=0) do inc(j);
               if j<=m
                  then begin
                         s:=s+[j];   { j 顶点进入已检查的 x 集合 }
                         if path(j)  { 若 j 出发存在增广轨 }
                           then begin
                                  g[x,i]:=-g[x,i];g[j,i]:=-g[j,i];
                                  { 则增广轨的匹配边与非匹配边对换 }
                                  path:=true;exit
                                end
```

```
                    end

                end

  end;

procedure kuhn_munkras;
  var
    u,i,j,al:integer;
  begin
    fillchar(l,sizeof(l),0); { 可行顶标初始化 }
    gt:=g;
    for i:=1 to n do          { 求可行顶标 l(Xi)=max(w(Xi,Yj)) }
      for j:=1 to m do        {                  Yj∈Y           }
        if l[i]<g[i,j] then l[i]:=gt[i,j];
    u:=1;cx:=[];cy:=[]; { 已匹配的 x 顶点集合和 y 顶点集合初始化 }
    for i:=1 to n do     { 求 Cl }
      for j:=1 to m do
        if l[i]+l[n+j]=gt[i,j]
          then g[i,j]:=1
          else g[i,j]:=0;
    while u<=n do
      begin
        s:=[u];t:=[];
        if not (u in cx) { 若 u 不属于匹配的 x 顶点集合 }
          then begin
                 if not path(u) { 若找不到从 u 出发的一条可增广轨 }
                   then begin
                          al:=maxint;
                          { 求 al=min( l(x)+l(y)-w(x,y) }
                          {    x∈S,y∈t                  }
                          for i:=1 to n do
                            for j:=1 to m do
                              if (i in s) and not (j in t)
                                and (l[i]+l[n+j]-gt[i,j]<al)
                                then al:=l[i]+l[n+j]-gt[i,j];
                          { 修改顶标形成 G̃i }
                          for i:=1 to n do if i in s then l[i]:=l[i]-al;
                          for i:=1 to m do if i in t then l[n+i]:=l[n+i]+al;
                          for i:=1 to n do { Gi←G̃i }
                            for j:=1 to m do
                              if l[i]+l[n+j]=gt[i,j]
                                then g[i,j]:=1
                                else g[i,j]:=0;
                          cx:=[];cy:=[]
                        end { then }
                   else cx:=cx+[u]; { u 进入已匹配的 x 顶点集合 }
                 u:=0
               end; { then }
```

```
        inc(u)          { 从下一顶点开始搜索 }
      end
  end;

procedure print; { 打印最佳分配 }
  var
    i,j,tot:integer;
  begin
    tot:=0;
    for i:=1 to n do
      for j:=1 to m do
      { 若第i个人能做第j项工作(工作效益为gt[i,j]),则打印 }
        if g[i,j]<0
          then begin
                 writeln(i,'--',j);tot:=tot+gt[i,j]
               end;
    writeln('Tot = ',tot)
  end;

begin
  read_graph;   { 读入二分图 }
  kuhn_munkras; { 求最佳匹配 }
  print         { 打印结果 }
end.
```

# 习 题 九

1. 某昼夜服务的公交线路每天各时间区段内所需司机和乘务人员数如下：

| 班次 | 时间 | 所需人数 |
|---|---|---|
| 1 | 6:00～10:00 | 60 |
| 2 | 10:00～14:00 | 70 |
| 3 | 14:00～18:00 | 60 |
| 4 | 18:00～22:00 | 50 |
| 5 | 22:00～2:00 | 20 |
| 6 | 2:00～6:00 | 30 |

设司机和乘务人员分别在各时间区段一开始时上班，并连续工作 8 小时，问该公交线路至少配备多少名司机和乘务人员？

2. 某航运公司承担六个港口城市 $A,B,C,D,E,F$ 的四条固定航线的物资运输任务。已知各条航线的起点、终点城市及每天航班数。

| 航线 | 起点城市 | 终点城市 | 每天航班次数 |
|---|---|---|---|
| 1 | $E$ | $D$ | 3 |
| 2 | $B$ | $C$ | 2 |
| 3 | $A$ | $F$ | 1 |
| 4 | $D$ | $B$ | 1 |

假定各条航线使用相同型号的船只，又各城市间的航程天数如题表 9-1 所示。

**题表 9-1**

| 起点＼终点 | $A$ | $B$ | $C$ | $D$ | $E$ | $F$ |
|---|---|---|---|---|---|---|
| $A$ | 0 | 1 | 2 | 14 | 7 | 7 |
| $B$ | 1 | 0 | 3 | 13 | 8 | 8 |
| $C$ | 2 | 3 | 0 | 15 | 5 | 5 |
| $D$ | 14 | 13 | 15 | 0 | 7 | 20 |
| $E$ | 7 | 8 | 5 | 17 | 0 | 3 |
| $F$ | 7 | 8 | 5 | 20 | 3 | 0 |

又知每条船只每次装卸货的时间各需 1 天，则该航运公司至少配备多少条船才能满足所有航线的运货要求？

3. 某厂拟用集装箱托运甲乙两种货物，如题表 9-2 所示。

**题表 9-2**

| 货物 | 体积(米³/箱) | 重量(百斤/箱) | 利润(百元/箱) |
|---|---|---|---|
| 甲 | 5 | 2 | 20 |
| 乙 | 4 | 3 | 10 |
| 托运限制 | 24 | 13 | |

问两种货物各托运多少箱，可使获利润为最大？

4. 某公司拟在市东、西、南三区建立门市部，拟议中有 7 个位置(点)$A_i(i=1,2,\cdots,7)$。规定：

在东区，由 $A_1,A_2,A_3$ 三点中至多选两个；

在西区，由 $A_4,A_5$ 两个点中至少选一个；

在南区，由 $A_6,A_7$ 两个点中至少选一个；

如选用 $A_i$ 点，设备投资估计为 $b_i$ 元，每年可获利润估计为 $c_i$ 元，但投资总额不能超过 $B$ 元。问应选择哪几个点可使年利润为最大？

5. 校办厂有 4 个工人，要指派他们完成 4 项工作，每人做各项工作所消耗的时间如题表 9.3 所示。

**题表 9-3**

| 工人＼工作 | $A$ | $B$ | $C$ | $D$ |
|---|---|---|---|---|
| 甲 | 15 | 18 | 21 | 24 |
| 乙 | 19 | 23 | 22 | 18 |
| 丙 | 26 | 17 | 16 | 19 |
| 丁 | 19 | 21 | 23 | 17 |

问指派哪个人去完成哪项工作，可使总的消耗时间为最小？

# 第十章　动态规划

在现实生活中，有一类活动的过程，由于它的特殊性，可将过程分成若干个互相联系的阶段，在它的每一阶段都需要作出决策，从而使整个过程达到最好的活动效果。因此各个阶段决策的选取不能任意确定，它依赖于当前面临的状态，又影响以后的发展。当各个阶段决策确定后，就组成一个决策序列，因而也就确定了整个过程的一条活动路线。这种把一个问题看作是一个前后关联具有链状结构的多阶段过程(如图 10-1)就称为多阶段决策过程，这种问题称为多阶段决策问题。

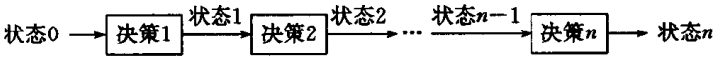



图　10-1

在多阶段决策问题中，各个阶段采取的决策，一般来说是与时间有关的，决策依赖于当前状态，又随即引起状态的转移，一个决策序列就是在变化的状态中产生出来的，故有“动态”的含义，我们称这种解决多阶段决策最优化的过程为动态规划方法。

应指出，动态规划是考察求解多阶段决策问题的一种途径、一种方法，而不是一种特殊算法。不像线性规划那样，具有一个标准的数学表达式和明确定义的一组规划。因此读者在学习时，除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。

## 10.1　动态规划问题的数学描述

首先，例举一个典型的且很直观的多阶段决策问题：

[**例 1**]　图 10-2 给出一个线路网络，两点之间连线上的数字表示两点间的距离(或费用)，试求一条由 $A$ 到 $G$ 的铺管线路，使总距离(或总费用)最小。

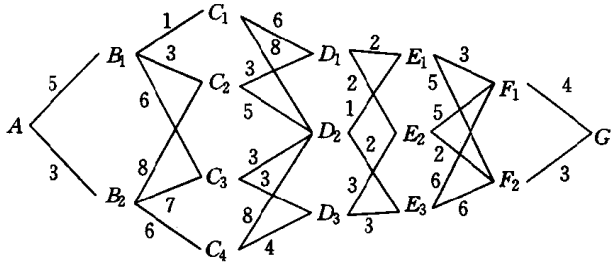



图　10-2

由图 10-2 可知，从 $A$ 到 $G$ 可分为 6 个阶段，除起点 $A$ 和终点 $G$ 外，其它各点即是上

一阶段的终点又是下一阶段的起点。例如从 $A$ 到 $B$ 的第一阶段中，$A$ 为起点，终点有 $B_1$，$B_2$ 两个，因而这时走的路线有两个选择，一是走到 $B_1$；一是走到 $B_2$。若选择 $B_2$ 的决策，$B_2$ 就是第一阶段在我们决策之下的结果，它既是第一阶段路线的终点，又是第二阶段路线的始点。在第二阶段，再从 $B_2$ 点出发，对于 $B_2$ 点就有一个可供选择的终点集合$\{C_2, C_3, C_4\}$；若选择由 $B_2$ 走至 $C_2$ 为第二阶段的决策，则 $C_2$ 就是第二阶段的终点，同时又是第三阶段的始点。同理递推下去，可看到各个阶段的决策不同，铺管线路就不同。很明显，当某阶段的起点给定时，它直接影响着后面各阶段的行进路线和整个路线的长短，而后面各阶段的路线的发展不受这点以前各阶段的影响。故此问题的要求是：在各个阶段选取一个恰当的决策，使由这些决策组成的一个决策序列所决定的一条路线，其总路程最短。

如何解决这个问题呢？可以采用穷举法，把所有由 $A$ 到 $G$ 可能的每一条路线的距离算出来，然后互相比较，找出最短者，相应地得出了最短路线。这样由 $A$ 到 $G$ 的 6 个阶段中共有 $2\times3\times2\times2\times2\times1=24$ 条不同路线，从中找出最短路线为

$$A \to B_1 \to C_2 \to D_1 \to E_2 \to F_2 \to G$$

最短距离为 18。显然这样作计算是相当复杂的。动态规划为此类多阶段决策问题寻求了一种简便的方法。为了便于讨论，我们先引入动态规划问题的一些概念、术语和符号。

**1. 决策和阶段**

在对问题的处理中作出某种选择性的行动就是决策。例如在 $A$ 点需选择下一步到 $B_1$ 还是到 $B_2$，这就是一次决策。一个实际问题可能要有多次决策或多个决策点，为此对整个问题，可按其特点划分成需要作出选择的若干轮次，这些轮次即称为阶段。如图 10-2 中，从 $A$ 到 $G$ 点可以分为 6 个阶段，从 $A$ 到 $B$ 为第一阶段，从 $B$ 到 $C$ 为第二阶段，……，从 $F$ 到 $G$ 为第六阶段。

**2. 状态和状态变量**

某一阶段的出发位置称为状态。通常一个阶段包含若干状态。例如阶段 4 含三种状态 $D_1, D_2, D_3$。从状态 $D_1$ 经一个阶段后可能达到状态 $E_1$，或 $E_2$，但不能达到状态 $E_3$。一般地，状态通常可用一个变量来描述，用来描述状态的变量称为状态变量，记第 $k$ 阶段的状态变量为 $U_k$。例 $U_4=\{D_1, D_2, D_3\}$。

**3. 决策变量和允许决策集合**

在每一个阶段中都需有一次决策，决策也可以用一个变量来描述，称这种变量为决策变量，一般用 $X_k$ 表示第 $k$ 阶段的决策变量。在实际问题中，决策变量的取值往往限制在某一个范围之内，此范围称为允许决策集合，用 $d_k$ 表示第 $k$ 阶段的允许决策集合。例如，图 10-2 中 $D_1=\{B_1, B_2\}$，它表示第一阶段可有两种不同的决策。

当第 $k$ 阶段的状态确定之后，可能作出的决策范围还要受到这一状态的影响，这就是说，决策变量 $X_k$ 还是状态变量 $U_k$ 的函数，记为 $X_k(U_k)$，简记为 $X_k$。把 $X_k$ 的取值范围记为 $D_k(U_k)$，显然有 $X_k(U_k)\in D_k(U_k)$。

例如在图 10-2 的第二阶段中，若从状态 $B_1$ 出发，就可作出三种不同的决策，其允许

决策集合 $D_2(B_1)=\{C_1,C_2,C_3\}$。若选取的点为 $C_2$，则 $C_2$ 是状态 $B_1$ 在决策 $X_2(B_1)$ 作用下的一个新的状态，记作 $X_2(B_1)=C_2$。

**4. 策略和最优策略**

所有阶段依次排列构成问题的全过程。全过程中各阶段决策变量 $X_k(U_k)$ 所组成的有序总体称为策略。

例如图 10-2 中，

$$A \xrightarrow{X_1(A)} B_1 \xrightarrow{X_2(B_1)} C_1 \xrightarrow{X_3(C_1)} D_2 \xrightarrow{X_4(D_2)} E_2 \xrightarrow{X_5(E_2)} F_2 \xrightarrow{X_6(F_2)} G$$

在实际问题中，可供选择的策略有一定的范围，该范围称为允许策略集合 $P$。从 $P$ 中找出最优效果的策略称为最优策略。

**5. 状态转移方程**

前一阶段的终点就是后一阶段的起点，前一阶段的决策变量就是后一阶段的状态变量，这种关系描述了由 $k$ 阶段到 $k+1$ 阶段状态的演变规律，称为状态转移方程。例如[例 1]的状态转移方程为 $U_{k+1}=X_k(U_k)$

**6. 指标函数和最优化概念**

指标函数是用来衡量多阶段决策过程优劣的一种数量指标。显然，它应该在全过程和所有子过程中有定义，并且可量度。在[例 1]中每阶段所走的距离 $d_k(U_k,X_k)(k=1,2,\cdots,6)$ 作为指标函数，而我们的目标是

$$Z=\min\sum_{k=1}^{6}d_k(U_k,X_k)$$

一般可写成

$$Z=\underset{X_1,X_2,\cdots,X_n}{\text{opt}}[r_1(U_1,X_1)\bigstar r_2(U_2,X_2)\bigstar\cdots\bigstar r_n(U_n,X_n)]$$

其中：

opt——最优化 optimization 的缩写，可取为 min 或 max；

★——可以是‘+’也可以是‘·’；

$r_k(U_kX_k)$——第 $k$ 阶段的指标函数。

动态规划的最优化概念是在一定条件下，找到一种途径，在对各阶段的效益经过按问题具体性质所确定的运算以后，使得全过程的总效益达到最优。

## 10.2 动态规划问题的最优化原理

为了帮助大家理解动态规划的基本思想，先说最短路线的一个重要特性：

如果从 $A\longrightarrow B\longrightarrow C\longrightarrow D$ 是 $A$ 至 $D$ 的最短路线，那么从 $B$ 到 $D$ 的最短路线必是 $B\longrightarrow C\longrightarrow D$。更一般地说：如果最短路线在第 $k$ 阶段通过 $P_k$，则由点 $P_k$ 出发到达终点的这条路线对于从 $P_k$ 出发到达终点的所有可能选择的不同路线来说，必定也是最短路线。

这就引出了从终点逐段向始点方向寻找最短路线的一种方法：

若以 $U_k$ 表示第 $k$ 阶段的一个决策点，从终点开始，依逆向求出倒数第一阶段、倒数第二阶段、……、倒数第 $n-1$ 阶段中各点到达终点的最短子路线，最终求出从起点到终点的最短路线。这就是动态规划的基本思想。

下面，我们按照动态规划的方法将[例 1]从最后一段开始计算，由后向前逐步倒推至 $A$ 点。

设 $f_k(U_k)$——从第 $k$ 阶段中的点 $U_k$ 到达终点的最短子路线的长度；

$d_k(X_k,U_k)$——$k$ 阶段中 $X_k$ 至 $U_k$ 的距离。

当 $k=6$ 时，$f_6(F_1)=4$，$f_6(F_2)=3$；

当 $k=5$ 时，

$$f_5(E_1)=\min\begin{bmatrix}d_5(E_1,F_1)+f_6(F_1)\\ d_6(E_1,F_2)+f_6(F_2)\end{bmatrix}=\min\begin{bmatrix}3+4\\ 5+3\end{bmatrix}=7$$

其相应的决策为 $X_5(E_1)=F_1$，这说明 $E_1$ 到 $G$ 的最短距离为 7，最短路线是 $E_1\to F_1\to G$

$$f_5(E_3)=\min\begin{bmatrix}d_5(E_2,F_1)+f_6(F_1)\\ d_5(E_2,F_2)+f_6(F_2)\end{bmatrix}=\min\begin{bmatrix}5+4\\ 2+3\end{bmatrix}=5$$

其相应的决策为 $X_5(E_3)=F_2$，这说明 $E_2$ 到 $G$ 的最短距离为 5，最短路线是 $E_2\to F_2\to G$。

$$f_5(E_3)=\min\begin{bmatrix}d_5(E_3,F_1)+f_6(F_1)\\ d_5(E_3,F_2)+f_6(F_2)\end{bmatrix}=\min\begin{bmatrix}6+4\\ 6+3\end{bmatrix}=9$$

相应的决策为 $X_5(E_3)=F_2$，这说明 $E_3$ 到 $G$ 的最短距离为 9，最短路线是 $E_3\to F_2\to G$。

类似地，可算得

当 $k=4$ 时，有

$$f_4(D_1)=7 \quad X_4(D_1)=E_2$$
$$f_4(D_2)=6 \quad X_4(D_2)=E_2$$
$$f_4(D_3)=8 \quad X_4(D_3)=E_2$$

当 $k=3$ 时，有

$$f_3(C_1)=13 \quad X_3(C_1)=D_1$$
$$f_3(C_2)=10 \quad X_3(C_2)=D_1$$
$$f_3(C_3)=9 \quad X_3(C_3)=D_2$$
$$f_3(C_4)=12 \quad X_3(C_4)=D_3$$

当 $k=2$ 时，有

$$f_2(B_1)=13 \quad X_2(B_1)=C_2$$
$$f_2(B_2)=16 \quad X_2(B_2)=C_3$$

当 $k=1$ 时，出发点只有一个 $A$ 点，则

$$f_1(A)=\min\begin{bmatrix}d_1(A,B_1)+f_2(B_1)\\ d_1(A,B_2)+f_2(B_2)\end{bmatrix}=\min\begin{bmatrix}5+13\\ 3+16\end{bmatrix}=18$$

且 $X_1(A)=B_1$，于是得到从起点到终点 $G$ 的最短距离为 18。

为了找出最短路线，再按计算的顺序反推之，可求出最优决策函数序列$\{X_k\}$，即由 $X_1(A)=B_1, X_2(B_1)=C_2, X_3(C_3)=D_1, X_4(D_1)=E_2, X_5(E_3)=F_2, X_6(F_2)=G$ 组成一个最优策略，因此找出相应的最短路线为

$A\rightarrow B_1\rightarrow C_2\rightarrow D_1\rightarrow E_2\rightarrow F_2\rightarrow G$

从上面的计算过程中，我们可以看出，在求解的各个阶段，我们利用了 $k$ 阶段与 $k+1$ 阶段之间的如下关系：

$$f_k(u_k)=\min_{X_k}\{d_k(u_k,x_k)+f_{k+1}(x_k)\}$$

$$k=6,5,4,3,2,1$$

$$f_7(u_7)=0$$

这种递推关系，叫做动态规划的函数基本方程。

动态规划的最优化原理是“作为整个过程的最优策略具有这样的性质：无论过去的状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。”

与穷举法相比，动态规划的方法有两个明显的优点：

(1) 大大减少了计算量；

(2) 丰富了计算结果。

从[例 1]的求解结果中。是我们得到不仅仅是由 $A$ 点出发到终点 $G$ 的最短路线及最短距离，而且还得到了从所有各中间点到终点的最短路线及最短距离，这对许多实际问题来讲是很有用的。

下面，我们给出[例 1]的程序题解。

```
Program Short;

Const
  Max          =  100;

Var
  N, St, En,                    { 顶点数,起点,终点 }
  I, J, X      :  Integer;
  Way          :  Array [1..Max, 1..Max] of Integer;
                  { 线路网络的带权矩阵 }
  Next         :  Array [1..Max] of Record
                  { Next[j].Ne——从 j 点出发的决策 }
                  { Next[j].Cost——St 至 j 点的最短路线长度 }
                    Ne, Cost   :  Integer
                  End;
  F            :  Text;   { 文件变量 }
  Str          :  String; { 文件名串 }

Begin
  Write('File name = ', Str);  { 读入文件名串 }
  Readln(Str);
  Assign(F, Str);              { 文件名与文件变量连接 }
```

```
Reset(F);                          { 读文件准备 }
Readln(F, N, St, En);              { 读入顶点数,起点,终点 }
For I := 1 to N Do                 { 读入各点间连线的距离 }
  For J := 1 to N Do Read(F, Way[I, J]);
Close(F);
For I := 1 to N Do Next[I].Cost := Maxint;
{ St 至各点的最短路线长度初始化 }
Next[En].Cost := 0; Next[En].Ne := 0;
{ 从最后一段开始,由后向前逐步递推 }
For I := 1 to N Do
  For J := 1 to N Do
    For X := 1 to N Do
      If (Way[J, X] > 0) And (Next[X].Cost <> Maxint)
        And (Way[J, X] + Next[X].Cost < Next[J].Cost) Then
{ 若 En 至 x 点的最短路已经求出,且加入边(j,x)后使得 En 至 j 的路线目前最短 }
        Begin
          Next[J].Cost := Next[X].Cost + Way[J, X];
          { 则记下最短路线长度 }
          Next[J].Ne := X  { 确定 j 点出发的决策为 x }
        End;
If Next[St].Cost = Maxint { 若未推至起点, 则无解 }
  Then Writeln('No Way')
  Else Begin                       { 否则从起点出发,按计算顺序反推最短路线 }
         X := St;
         While X <> 0 Do Begin
           Write(X:3,' ');
           X := Next[X].Ne
         End
       End
End.
```

## 10.3 动态规划应用举例

10.2 节给出了动态规划的逆推解法，下面我们给出另一种解法——从始点向终点方向寻找最佳路线的顺推解法。若以 $U_k$ 表示第 $k$ 阶段的一个决策点，从始点开始，依顺向求出第一阶段，第二阶段，……，第 $n$ 阶段中各点到达始点的最佳路线，最终求出始点到终点的最佳路线。

[**例 1**] 图 10-3($a$)示出了一个数字三角形，请编一个程序，计算从顶至底的某处的一条路径，使该路径所经过的数字的总和最大。

(1) 每一步可沿左斜线向下或右斜线向下；

(2) 1<三角形行数≤100；

(3) 三角形中的数字为 0,1,…,99。

输入数据：

由 $INPUT_1$.TXT 文件中首先读到的是三角形的行数，在例子中 $INPUT_1$.TXT 表示如图 13-3($b$)。

输出数据：

把最大总和(整数)写入 OUTPUT$_1$.TXT 文件。

上例为 30。

**解**：我们按三角形的行划分阶段，若行数为 $n$，则可把该题看作一个 $n-1$ 个阶段的决策问题，见图 10-4。

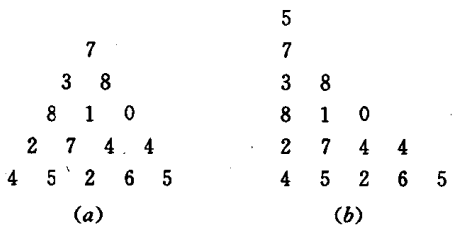



图 10-3

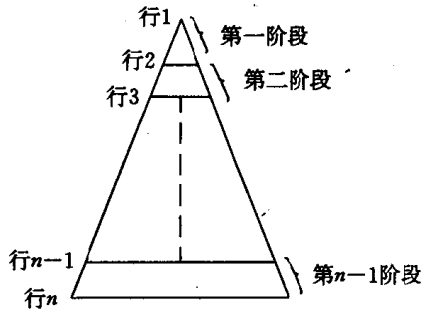



图 10-4

设：

$f_k(U_k)$——从第 $k$ 阶段中的点 $U_k$ 至三角形顶点有一条最佳路径，该路径所经过的数字的总和最大，$f_k(U_k)$表示为这个数字和。

由于每一次决策有两个选择，或沿左斜线向下，或沿右斜线向下，因此设

$U_{k1}$——$k-1$ 阶段中某点 $U_k$ 沿左斜线向下的点；

$U_{k2}$——$k-1$ 阶段中某点 $U_k$ 沿右斜线向下的点；

$d_k(U_{k1})$——$k$ 阶段中 $U_{k1}$的数字；

$d_k(U_{k2})$——$k$ 阶段中 $U_{k2}$的数字。

因而可写出顺推关系式

$$f_k(U_k) = \max\{f_{k-1}(U_k) + d_k(U_{k1}), f_{k-1}(U_k) + d_k(U_{k2})\}$$

$$f_0(U_0) = 0;$$

$$k = 1,2,3,4,\cdots,n$$

经过一次顺推，便可分别求出从顶至底 $N$ 个数的 $N$ 条路径，在这 $N$ 条路径所经过的 $N$ 个数字和中，最大值即为正确答案。

根据上述顺推关系，我们编写程序如下：

```
Program ID1P1;

Const
  Maxn          =  100;

Type
  Node          =  Record
                     Val, Tot : Integer
                     { 当前格数字；从[1,1]到当前格的路径所经过的数字和 }
                   End;
```

```
Var
  List          : Array [1..Maxn, 1..Maxn] of Node; { 计算表 }
  N, Max,         { 行数, 最大总和 }
  I, J          : Integer; { 辅助变量 }
  Fi            : Text;    { 文件变量 }

Procedure Init;
Begin
  Assign(Fi, 'INPUT.TXT'); { 文件名和文件变量连接 }
  Reset(Fi);               { 文件读准备 }
  Readln(Fi, N);           { 读三角形行数 }
  For i := 1 to N Do       { 读入三角形各格的数字 }
    For j := 1 to i Do
      Read(Fi, List[i, j].Val);
  Close(Fi)
End;

Procedure Main;
Begin
  List[1, 1].Tot := List[1, 1].Val; { 从[1,1]位置开始往下顺推 }
  For i := 2 to N Do
    For j := 1 to i Do Begin
      List[i, j].Tot := -1; { 从[1,1]至[i,j]的数字和初始化 }
      If (j <> 1) And
         (List[i - 1, j - 1].Tot + List[i, j].Val > List[i, j].Tot) Then
        List[i, j].Tot := List[i - 1, j - 1].Tot + List[i, j].Val;
  { 若从[i-1,j-1]选择右斜线向下会使[1,1]至[i,j]的数字和最大,则决策该步 }
      If (j <> i) And
         (List[i - 1, j].Tot + List[i, j].Val > List[i, j].Tot) Then
        List[i, j].Tot := List[i - 1, j].Tot + List[i, j].Val
  { 若从[i-1,j]选择左斜线向下会使[1,1]至[i,j]的数字和最大,则决策该步 }
    End;
  Max := 1;
  { [1,1]至底行各点的 N 条路径所经过的数字和中,选择最大的一个输出 }
  For i := 2 to N Do
    If List[N, i].Tot > List[N, Max].Tot Then
      Max := i;
  Writeln(List[N, Max].Tot) { 输出最大总和 }
End;

Begin
  Init; { 读入数字三角形 }
  Main  { 求最大总和 }
End.
```

动态规划不仅可以求单向的最佳路线,而且还适用于求满足某种特定要求的:“最佳”回路问题。

[**例 2**] 给定一张航空图,图中的顶点代表城市,边代表两城市的直通航线。现要求找出一条满足下述限制条件的、含城市最多的旅行路线:

(1) 从最西的一个城市出发，单方向从西向东途经若干城市到达最东一个城市，然后再单方向从东向西飞回起点(可途经若干城市)；

(2) 除起点城市外，任何城市只能访问一次，起点城市被访问两次；出发 1 次，返回 1 次。

解：

设最西的城市序号为 1，最东的城市序号为 $N$。题目实际上是在航空图上求一条含顶点数最多的回路，这条回路上的顶点序号互不相同，且必经顶点 1 和顶点 $N$。因此，我们可以将这条回路对应两条单向航线，即城市 1 至城市 $N$ 的往程航线和城市 $N$ 飞回城市 1 的返程航线。两条航线除起点和终点两城市外，中间城市互不相同。那么如何求这两条单向航线呢？

我们还是采用动态规划中的逆推解法。将整个问题分若干阶段，每一个阶段上有两个决策点要选择，一个是往程航线上的当前城市，一个是返程航线上的当前城市。从城市 $N$ 开始，依逆向求出倒数第一阶段、倒数第二阶段、……、倒数第 $N-1$ 阶段中两个当前城市至城市 $N$ 的两条航线上途经的最多城市数(两条路线上的中间城市互不相同)。

设 $f(i,j)$——城市 $i$ 至城市 $N$ 与城市 $j$ 至城市 $N$ 的两条航线上的最多城市数。

显然，倒数第一个阶段：$f(N,N)=1$；

倒数第二阶段：$f(i,N)=2$(该阶段中城市 $i$ 与城市 $N$ 之间有直通航线)；

其它阶段：$f(i,j)=f(j,i)$。

由此，我们得出逆推关系式：

$f(N,N)=1$；

$f(i,N)=2$；　[存在边$(i,N)$]

$f(j,i)=f(i,j)$；

$f(i,j)=\max\{f(u,j)+1,f(i,j)\}$

$(u<>j)$且[边$(u,i)$存在]且[$f(u,j)<>0$]

$i=N-1,N-2,\cdots,1;j,u=i+1,i+2,\cdots,N$；

在由后向前逐步推至城市 1 后，为了求出最佳旅行路线，再按计算的顺序反推之，使得其中一条作为往程航线，另一条作为返程航线。

最后，给出程序题解。

```
Program AERO_New;

Const
  Maxn                  =  100;

Type
  Node                  =  Record
                             A, B, Vt       :  Byte;
                           End;
Var
  FiNm                  :  Text;{ 文件变量 }
  N, M,                        { 城市顶点数和直通航线(边)数 }
```

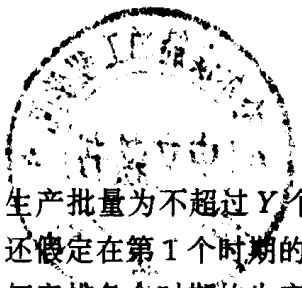
```
  i, j, u, v              :  Integer;
  Lk                      :  Array [1..Maxn, 1..Maxn] of 0..1;
{ 航空图的邻接矩阵 }
  List                    :  Array [1..Maxn, 1..Maxn] of Node;
{ 旅行方案  List[i,j].A——城市 i 至城市 N 的航线上,与城市 i 相连的城市序号 }
{           List[i,j].B——城市 j 至城市 N 的航线上,与城市 j 相连的城市序号 }
{           List[i,j].Vt——上述两条航线途经的最多城市数                    }
{                  i, j——当前两条航线上的决策城市序号                     }

Procedure Init;
Var
  Str             :  String;
Begin
  Write('File name = ');          { 输入文件名 }
  Readln(Str);
  Assign(FiNm, Str);              { 文件名与文件变量连接 }
  Reset(FiNm);                    { 读准备 }
  Readln(FiNm, N, M);             { 从文件中读入城市数与直通航线数 }
  Fillchar(Lk, SizeOf(Lk), 0);
  For i := 1 to M Do Begin        { 从文件中输入邻接矩阵 }
    Readln(FiNm, u, v);
    Lk[u, v] := 1
  End;
  Close(FiNm)
End;

Procedure Main; { 从城市 N 逐段向城市 1 方向寻找最佳旅行路线 }
Begin
  List[N, N].Vt := 1; { 第 1 个访问城市 N }
  For i := N-1 downto 1 Do
    For j := i+1 to N Do Begin
      List[i, j].Vt := 0;
      { 城市 N→城市 i 与城市 N→城市 j 的两条航线上的城市数初始化 }
      If (j = N) And (Lk[i, N] = 1) Then Begin
      { 若城市 i 与城市 N 有直通航线,则两条航线上的城市数为 2,
        其中一条航线为城市 i←城市 N }
        List[i, j].Vt := 2;
        List[i, j].A := N; List[i, j].B := j;
      End;
      List[i,j].B := j; { 在两条至城市 N 的航线上增加被访问的城市 }
      For u := i+1 to N Do
        If ( u <> j ) And (Lk[i, u] = 1) And (List[u, j].Vt <> 0) And
           (List[u, j].Vt + 1 > List[i, j].Vt) Then Begin
          List[i, j].Vt := List[u, j].Vt + 1;
          List[i, j].A := u;
          List[i, j].B := j
        End;
      List[j, i].Vt := List[i, j].Vt;
      List[j, i].A := List[i, j].B; List[j, i].B := List[i, j].A
```

```
    End
End;

Procedure Out_Line(i, j : Byte);
{ 递归输出城市 i 至城市 N 和城市 N 至城市 j 的旅行路线 }
Begin
  If i = j Then { 往程航线输出完毕 }
    Writeln('——', N)
  Else Begin
    If (i <> N) And (List[i, j].A <> i) Then Write('——', i);
    { 输出往程路线上的城市 i }
    Out_Line(List[i, j].A, List[i, j].B);
    { 递归输出往程路线上的下一个城市 List[i,j].A }
    If (j <> N) And (List[i, j].B <> j) Then Write('——', j)
    { 回溯输出返程航线上的城市 }
  End
End;

Procedure Print; { 输出最佳旅行路线 }
Begin
  u := 2;
  For i := 3 to N Do
  { 选取某条含城市数最多的航线(u 至 N(u<>1)) 作为往程航线 }
    If (Lk[1, i] = 1) And (List[1, i].Vt > List[1, u].Vt) Then u := i;
  If List[1, u].Vt = 0 Then { 若无往程航线,返回"无解"信息 }
    Writeln('No Answer')
  Else Begin { 否则输出城市 1→城市 N→城市 u 的旅行路线 }
    Out_Line(1, u);
    Writeln('——1') { 最后返回城市 1 }
  End
End;

Begin
  Init;  { 输入航空图 }
  Main;  { 寻找最佳旅行路线 }
  Print  { 打印结果 }
End.
```

# 习 题 十

1. 某厂根据计划安排,拟将 $n$ 台高效率的设备,分配给 $m$ 个车间,各车间获得这种设备后,可以为国家提供的盈利为 $C_{ij}$($i$ 台设备提供给 $j$ 车间所获的利润,$i=1,2,\cdots,n$,$j=1,2,\cdots,m$)。问这 $n$ 台设备如何分配给各车间,才能使国家得到的盈利最大?

2. 某工厂要对一种产品制订今后 $n$ 个时期的生产计划,据估计在今后 $n$ 个时期内,市场对于该产品的需求量为 $C_k$($k=1,2,\cdots,n$)单位。假定该厂生产每批产品的固定成本为 $X$(千元),若不生产就为 0;每单位产品成本为 1(千元);每时期生产能力所允许的最大



生产批量为不超过 $Y$ 个单位；每个时期末未售出的产品，每单位需付存储费 0.5(千元)。还假定在第 1 个时期的初始库存量为 0，第 $n$ 个时期之末的库存量也为 0。试问该厂应如何安排各个时期的生产和库存，才能在满足市场需要的条件下，使总成本最小？

3. 设有 $n$ 个工件需要在机床 $A$，$B$ 上加工，每个工件都必须先 $A$ 而后 $B$ 的两道加工工序。以 $a_i$，$b_i$ 分别表示工件 $i(1\leqslant i\leqslant n)$ 在 $A$，$B$ 上的加工时间。问应如何在两机床上安排各工件加工顺序，使在机床 $A$ 上加工第 1 个工件到在机床 $B$ 上将第 $n$ 个工件加工完为止，所用的加工总时间最少？

4. 现有一个 $n$ 个城市间的距离矩阵。一个货郎，他从某个村庄出发，通过其它每个村庄一次且仅一次，最后仍回到原出发的村庄，问应如何选择行走路线，能使总的行程最短？