

离散数学在信息学竞赛中的运用

目录

- ◆ 重集全排列
- ◆ Catalan数
- ◆ 简单数论
- ◆ 矩阵的简单运用
- ◆ 棋盘多项式与任务分配
- ◆ 置换群与pólya定理

重集全排列

- ◆ 一般我们认为排列或组合的元素必须两两相异，现取消这一限制。例如一个排列可以是aaaaabbbbbbaaaabbbb等等，一个组合可以只{a, a, b, b}。对于重集 $S = \{p_1a_1, p_2a_2, \dots, p_ka_k\}$ ，其中 p_i 表示 a_i 在集合中的个数。那么这个重集的全排列个数是 $(\sum P_i)! / \prod (P_i!)$ ，其中 $(i=1 \text{ to } k)$ 。

字符串序号

- ◆ 字符串 `acab` 含有两个 `a` ,一个 `b` ,一个 `c` ,和 `acab` 含的字母和每个字母的个数都相等的字符串还有: `aacb`,`baca`等, 因为他们也是含有两个 `a` ,一个 `b` ,一个 `c` 。所有满足这个性质的字符串按字典顺序排列后, `acab` 是第 5 个, 我们就说 `acab` 的序号是 5 .再如: `ba` 的序号是 2,`aa` 的序号是 1. 编程求出给定字符串 `S` (长度 ≤ 100) 的序号 `P` (保证 ≤ 30000) 注意: 字符串只含小写字母。
- ◆ 本题可以逐位利用重集全排列的公式, 计算出解。

飞行问题

- ◆ 在 $n \times m \times k$ 一只飞蛾从空间上的 $(0,0,0)$ 点飞到 (n,m,k) 点，规定飞蛾每次只能向右，向前，向下飞行一单位距离，问飞蛾有多少种不同的路径可以飞到终点。
- ◆ 向右飞行计为 a ，向前计为 b ，向下计为 c 。那么每一个飞行过程就对应着一个多个 a 多个 b 多个 c 的排列，反之也对应。而 a 有 n 个， b 有 m 个， c 有 k 个。所以这些字母全排列的个数对应着飞行路径的个数 $-(m+n+k)!/m!n!k!$ 。

Catalan数

- ◆ 堆栈问题
- ◆ 栈有两种最重要的操作，即pop（从栈顶弹出一个元素）和push（将一个元素进栈）。
- ◆ 考虑这样一个问题：一个操作数序列，从1，2，一直到 n （图示为1到3的情况），栈A的深度大于 n 。

Catalan数

- ◆ 现在可以进行两种操作：
- ◆ 1. 将一个数，从操作数序列的头端移到栈的头端（对应数据结构栈的push操作）
- ◆ 2. 将一个数，从栈的头端移到输出序列的尾端（对应数据结构栈的pop操作）
- ◆ 使用这两种操作，由一个操作数序列就可以得到一系列的输出序列。
- ◆ 例如1 2 3 4 5为初始序列，经过进栈，出栈，进栈，进栈，进栈，出栈，进栈，出栈，出栈操作后，序列变为了1 4 5 3 2

Catalan数

- ◆ 试问一个序列 $1\dots n$ 经过堆栈处理之后，有多少种出栈序列。
- ◆ 要知道这个问题如何解决，先将这个问题的模型转换一下。进栈时，我们在纸上画一个左括号，出栈时，画一个右括号。这样就把整个进栈出栈的过程转化成了一个括号序列。而且任何一个进栈出栈的过程都对应着唯一的括号序列。

Catalan数

- ◆ 好括号列的定义：（1）空列是好括号列。（2）若A和B是好括号列，则AB是好括号列。（3）若A是好括号列， (A) 是好括号列。
- ◆ 由好括号的定义可知，任何一个进栈出栈过程转化成的括号序列都是好括号列。那么任何一个好括号列也对应着一个唯一的进栈出栈过程。计算出长度为 n （ n 为完整括号个数，一个左括号加上一个右括号算作一个完整括号）的好括号列的个数也就是计算出了堆栈问题的答案。
- ◆ 经过数学证明，长度为 n 的好括号列的个数是 $C(n) = C(2n, n) - C(2n, n-1) = C(2n, n) / (n+1)$ 。这个数被称为Catalan数(卡特兰数)。

Catalan数

- ◆ 考虑新 $X_1 \times X_2 \times X_3 \dots \times X_n$ ，对这个乘式加括号搞结合律，则共加 $n-1$ 个括号，那么添加括号的方案数是 $C(n-1)$ 种。若最后一次乘法是 $(X_1, X_2 \dots X_k) \times (X_{k+1}, X_{k+2} \dots X_n)$ ，那么组合方式也是相乘的(乘法原理)，设前 k 个元素的填括号方式有 $a(k)$ 种，后面的 $n-k$ 个元素有 $a(n-k)$ 种，则 $a(n) = \sum a(k) \times a(n-k)$ ，其中 $k=1$ to $n-1$ ， $a_1=1$ 。这就是Catalan数的递推公式。
- ◆ 由上述的递推公式可以知道，包含 n 个结点的二叉树有 $C(n)$ 棵。

小结

- ◆ 可以运用Catalan数的地方：
- ◆ 设圆周上 $2n$ 个点，用 n 条彼此在圆内部无公共交点的弦连接这些点，共有 $C(n)$ 种连接方式。
- ◆ 在 $n \times n$ 的矩阵中，从 $(0, 0)$ 点走到 (n, n) 点，规定只能向下或向右走，且不能穿过左上到右下的对角线，共有 $C(n)$ 种走的方式。
- ◆ 填括号问题
- ◆ 进栈出栈问题
- ◆ 结点数是 n 的二叉树的个数

数论

- ◆ 这里简要介绍一些实用的数论算法。
- ◆ 求最大公约数(欧几里德算法)
- ◆ Function gcd(a,b:longint):longint;
- ◆ Begin
- ◆ if $a \bmod b = 0$ then gcd:=b
- ◆ else gcd:=gcd(b, $a \bmod b$);
- ◆ End;
- ◆ 这里，调用gcd(a,b)函数可以求出a和b 的最大公约数。

欧几里德算法

- ◆ 狼找兔子
- ◆ 一座山周围有 n 个洞，顺时针编号为 $0, 1, 2, \dots, n-1$ 。而一只狼从0号洞开始，顺时针方向计数，每遇到 m 个洞就进洞找兔子。例如 $n=5$ ， $m=3$ ，狼经过的洞依次为 $0, 3, 1, 4, 2, 0$ 。输入 n 和 m 。试问兔子有没有幸免的机会，如果有，该藏在哪儿？

狼找兔子

- ◆ 不妨让兔子躲在1号洞，因为若狼能从0号洞到达1号洞，则必能从1号洞到达2号洞，.....，兔子难逃厄运。反过来说，若有安全洞，则1号洞就是其中一个。
- ◆ 再来看狼的运动。狼的第 i 次运动后的洞址应该是 $(m \times i) \bmod n$ 。若 $(m \times i) \bmod n = 1$ ，即狼第 i 次运动后到达1号洞，则 $\gcd(m, n) = 1$ 。因为若 $\gcd(m, n) = k > 1$ ，则由 $(m' \times k \times i) \bmod (n' \times k) = 1$ ，设 $m' \times k = m$ ， $n' \times k = n$ ， $m' \times k \times i - [(m' \times k \times i) / (n' \times k)] \times n' \times k = 1$ 。得出 $1/k$ 是整数，与 $k > 1$ 矛盾。所以当 $\gcd(n, m) = 1$ 时，兔子无法幸免。反之，兔子应该选择除了 $\{i \times \gcd(m, n) | i = 0 \dots (n/\gcd(m, n) - 1)\}$ 外的洞躲藏。

扩展欧几里德算法

- ◆ 下面，我们对欧几里德算法进行推广，使得该算法不仅能得出任意两个正整数 a 和 b 的最大公约数 d ，而且还能计算出满足下式的整系数 x 和 y 。
- ◆ $D=\gcd(a,b)=ax+by$ (x 和 y 可能为0或负数)

扩展欧几里德

- ◆ Function euclid(a,b:longint; var x,y:longint):longint;
- ◆ Var t:longint;
- ◆ Begin
- ◆ if b=0 then
- ◆ begin
- ◆ euclid:=a; x:=1; y:=0;
- ◆ end
- ◆ Else
- ◆ begin
- ◆ euclid:=euclid(b,a mod b,x,y);
- ◆ t:=x; x:=y; y:=t-(a div b)*y;
- ◆ end;
- ◆ End;

扩展欧几里德算法

- ◆ 上述函数是从欧几里德算法中衍生出来的。算法一开始，它首先测试 $b=0$ ，若 $b=0$ ，返回当前最大公约数 a ， $x=1$ ， $y=0$ ，以使 $a=\gcd(a,0)=a*1+b*0$ ；若 $b\neq 0$ ，则欧几里德算法首先计算出 $d'=\gcd(b,a \bmod b)=bx'+(a \bmod b)y'$ 中的 x' 和 y' 。在这种情况下， $d=\gcd(a,b)=d'=\gcd(b,a \bmod b)$ 。为了得到 $d=ax+by$ 的 x 和 y ，我们利用等式 $d=d'=bx'+(a \bmod b)y'$ 得出 $d=bx'+(a-(a \operatorname{div} b)b)y'=ay'+b(x'-(a \operatorname{div} b)y')$ 。因此， $x=y'$ ； $y:=x'-(a \operatorname{div} b)*y'$ 。

解的个数

- ◆ 已知 x, y 满足如下条件：
- ◆ $ax + by + c = 0$; $x_1 \leq x$; $x \leq x_2$; $y_1 \leq y$; $y \leq y_2$; x, y 均为整数。
- ◆ 其中： $a, b, c, x_1, x_2, y_1, y_2$ 都是绝对值不超过 10^8 的整数。
- ◆ 求 (x, y) 的解的个数。
- ◆ 输入： $a, b, c, x_1, x_2, y_1, y_2$
- ◆ 输出： 输出解的个数
- ◆ 样例输入
- ◆ 1 1 1 -10 10 -9 9
- ◆ 样例输出
- ◆ 19

解的个数

- ◆ 直接的枚举显然是不符合时间要求的。由于方程所有解之间的关系是可以互相推导的，所以如果我们已知了方程的一个解，那么用数学方法推导出其他的解，继而统计解的个数。
- ◆ 所以在求解方程的任意一个解的时候，我们可以用到扩展的欧几里德算法。

补充材料

令 a 为整数， d 为正整数，那么有惟一的整数 q 和 r ，其中 $0 \leq r < d$ ，使得 $a = dq + r$ 。可以用这个定理来定义除法： d 叫除数， a 叫被除数， q 叫商， r 叫余数。如果两个数 a, b 除以一个数 c 的余数相等，说 a 和 b 关于模 c 同余，记作 $a \equiv b \pmod{c}$ 。

素数的测试

- ◆ 刚刚接触信息学的时候，可能就做过关于素数测试的题目，当时的方法是 $2\text{-}\sqrt{n}$ 的枚举，这样的效率并不高。
- ◆ 欧拉定理： $a^{\psi(n)} \equiv 1 \pmod{n}$ ，其中 $\psi(n)$ 是 n 的欧拉函数， $\psi(n)$ = 不大于 n 的但与 n 互质的正整数个数。 a 可以取任意值。
- ◆ 易知， $\psi(\text{素数}n) = n - 1$

费尔马小定理

- ◆ 从而，当 n 是质数的时候， $a^{n-1} \equiv 1 \pmod{n}$ ，这是欧拉定理的特殊情况，也称为费尔马小定理。
- ◆ 那么，费尔马小定理的逆定理是否成立呢？答案是否定的，但是这种情况很少。也就是说，对于 a 取 $1 \sim n-1$ 的任意一个值的时候，都有 $a^{n-1} \equiv 1 \pmod{n}$ 成立的话，则几乎可以肯定地确认 n 是素数。

费尔马小定理

- ◆ 当 $a=2$ 时，小于 10^4 的 n 值中，只会产生22个不成立的 n 。当 $a=3$ 时，在小于 10^8 的 n 值中，只会产生255个不成立的 n 。
- ◆ 显然可以看出这个出错的概率是很小的。
- ◆ 我们用分治法，可以在 $k\log N$ 的复杂度内判断 N 是否为素数。其中 k 是 a 的取值次数。一般情况下，我们只需要让 $a=2,3,4$ 判断一下就可以了。
- ◆ 于是费尔马小定理的逆定理成为了我们进行素数测试的有效算法之一。

选数

- ◆ 选数(Noip2002普及组)
- ◆ 已知 n 个整数 x_1, x_2, \dots, x_n ，以及一个整数 $k(k < n)$ 。从 n 个整数中任选 k 个整数相加，可分别得到一系列的 $和$ 。例如当 $n=4, k=3$ ，4个整数分别为3，7，12，19时，可得全部的组合与它们的和为：
- ◆ $3+7+12=22$ $3+7+19=29$
- ◆ $7+12+19=38$ $3+12+19=34$

选数

- ◆ 现在，要求你计算出和为素数共有多少种。
- ◆ 输入格式为：
- ◆ n, k ($1 \leq n \leq 20$, $k < n$)
- ◆ x_1, x_2, \dots, x_n ($1 \leq x_i \leq 5000000$)
- ◆ 一个整数(满足条件的种数)。
- ◆ 样例输入：
4 3
3 7 12 19
- ◆ 样例输出： 1

选数

- ◆ 这一题采取的是搜索的算法，这已经是非常低效的了，但是还得加上素数判定的复杂度，所以这里采取费尔马小定理的逆定理判定素数才能使得整个程序在短时间内出解。

小结

- ◆ 关于数论的一些知识本人还十分缺陷，所以只能做一些简单知识的介绍。

矩阵的简单运用 – by 韩文弢

- ◆ 由 m 行、 n 列的标量所构成的数组被称为一个 $m \times n$ 的矩阵(matrix)。
- ◆ 一般用大写字母表示矩阵，对应的小写字母表示矩阵中的项(entry)。
- ◆ 这里， a_{ij} 就是矩阵 A 中第 i 行第 j 列的项。

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \\ = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

矩阵的乘法

- ◆ 设 $A=(a_{ij})_{m \times n}, B=(b_{ij})_{n \times p}$ 是两个矩阵，矩阵 $C=AB$ ，则 C 是一个 $m \times p$ 的矩阵，且其中第 i 行第 j 列的项的值为 A 中第 i 个行向量与 B 中第 j 个列向量的内积。
- ◆ 也就是说，

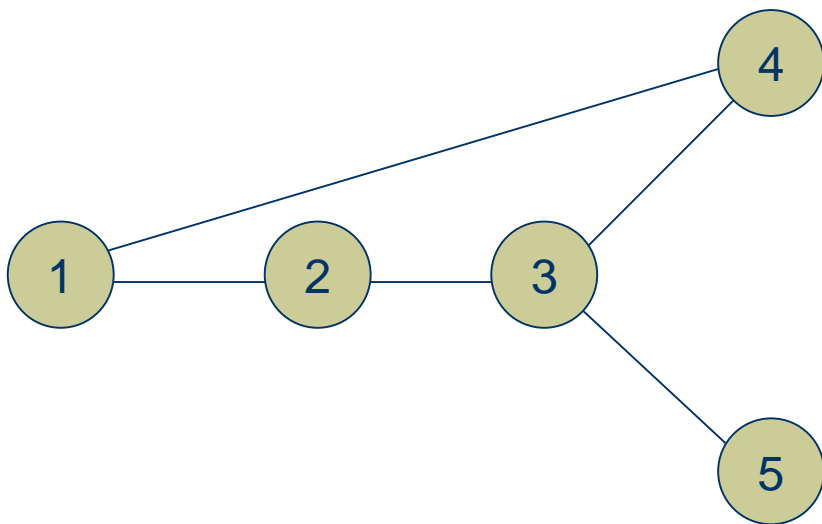
$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

矩阵乘法举例

$$A = \begin{pmatrix} -2 & 1 & 2 & 0 \\ 4 & -1 & 0 & 1 \\ 3 & 1 & -2 & 1 \end{pmatrix} \quad \begin{aligned} c_{11} &= (-2, 1, 2, 0) \cdot (1, 0, -2, 1) \\ &= -2 + 0 - 4 + 0 = -6 \\ c_{12} &= (-2, 1, 2, 0) \cdot (-4, -3, -1, 0) \\ &= 8 - 3 - 2 + 0 = 3 \\ c_{13} &= (-2, 1, 2, 0) \cdot (2, 1, 0, 1) \\ &= -4 + 1 + 0 + 0 = -3 \end{aligned}$$
$$B = \begin{pmatrix} 1 & -4 & 2 \\ 0 & -3 & 1 \\ -2 & -1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$
$$C = AB = (c_{ij}) \quad C = \begin{pmatrix} -6 & 3 & -3 \\ 5 & -13 & 8 \\ 8 & -13 & 8 \end{pmatrix}$$

矩阵乘法的应用

- ◆ 求无权图中长度为 k 的路径数



$$Adj = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$Adj^2 = \begin{pmatrix} 2 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 2 & 1 \\ 2 & 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 2 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

矩阵乘法运算的性质

- ◆ 不满足交换律
 - $AB=BA$ 不一定成立
- ◆ 满足结合律
 - $(AB)C=A(BC)$ 成立
- ◆ 满足分配律
 - $A(B+C)=AB+AC$ 成立（左分配律）
 - $(A+B)C=AC+BC$ 成立（右分配律）

例题1：递推数列(recurrence)

给出一个 k 阶齐次递推数列 $\{f_i\}$ 的递推公

$$\text{式 } f_i = a_1 f_{i-1} + a_2 f_{i-2} + \cdots + a_k f_{i-k} \ (i \geq k),$$

以及初始值 f_0, f_1, \dots, f_{k-1} ，求 f_n 。

$$1 \leq k \leq 100, \quad 0 \leq n \leq 1,000,000$$

例题1样例

◆ 输入样例

2 10

1 1

1 1

◆ 输出样例

89

递推数列解答

- ◆ Fibonacci数列递推公式: $f_i = f_{i-1} + f_{i-2}$
- ◆ 另外再设一个矩阵A, 使得
- ◆ 容易看出,
- ◆ 即,

$$F = \begin{pmatrix} f_{i-1} \\ f_{i-2} \end{pmatrix}, F' = \begin{pmatrix} f_i \\ f_{i-1} \end{pmatrix}$$

$$F' = AF$$

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} f_i \\ f_{i-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_{i-1} \\ f_{i-2} \end{pmatrix}$$

递推数列解答（续）

- ◆ 设
- ◆ 则有
- ◆ 如何降低复杂度？
- ◆ 时间复杂度：
 - $O(\log n)$

$$F_i = \begin{pmatrix} f_{i+1} \\ f_i \end{pmatrix}$$

$$F_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, F_i = A^i F_0$$

分治！

递推数列解答（续）

- ◆ 推广到一般情况，
- ◆ 其中，
- ◆ 时间复杂度：
 - $O(k^3 \log n)$

$$F_i = \begin{pmatrix} f_{i+k-1} \\ f_{i+k-2} \\ \vdots \\ f_i \end{pmatrix}, F_i = A^i F_0$$

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_k \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

例题2：图形变换(shape)

平面上有 n 个点需要依次进行 m 个变换处理。变换的规则有 4 种，分别为：

平移(M) 按照向量 $v = (x, y)$ 对点进行平移，即 $(x_0, y_0) \rightarrow (x_0 + x, y_0 + y)$ 。

缩放(Z) 以原点为参考，按照比例 λ 对点进行缩放，即 $(x_0, y_0) \rightarrow (\lambda x_0, \lambda y_0)$ 。

翻转(F) 把点按照 x 轴或者 y 轴进行翻转，即 $(x_0, y_0) \rightarrow (x_0, -y_0)$ 或 $(-x_0, y_0)$ 。

旋转(R) 使点绕原点旋转 a 度 (a 为正时逆时针旋转，为负时顺时针旋转)，离原点的距离不变。

其中括号中的字母为该变换的指令代码。编程序依次输出变换后各点的坐标。

$$1 \leq n \leq 10,000, \quad 1 \leq m \leq 10,000$$

例题2样例

◆ 输入样例

4 1

M 0 1

Z 0.5

R 45

F 0

2 1

◆ 输出样例

0.0000 -1.4142

图形变换解答

- ◆ 直观算法：直接模拟
 - 时间复杂度： $O(mn)$
 - 太慢了！
- ◆ 改进算法：矩阵运算
 - 时间复杂度： $O(m+n)$
 - 很好！
- ◆ 实际应用
 - 三维图形处理

小结

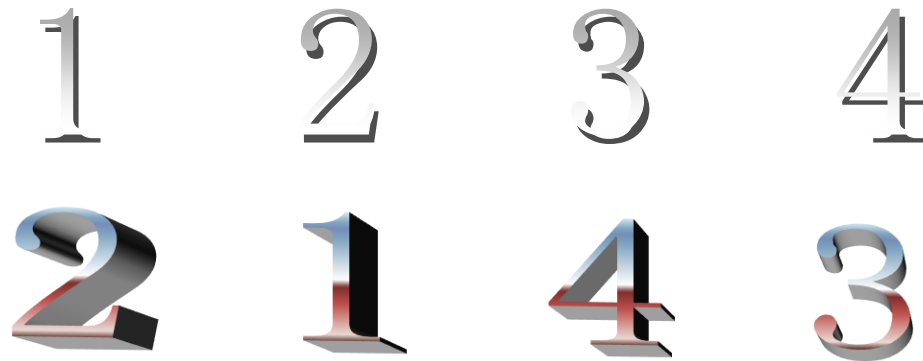
- ◆ 线性代数是一件十分有力的数学工具。
- ◆ 一般地，矩阵乘法可以用来解决大量线性关系的计算，能够把问题大大简化。
- ◆ 而线性方程组则是解决自然科学问题的常用工具，可以用矩阵的基本操作来求解。
- ◆ 涉及思想：
 - 分治思想
 - 转化思想
 - 消元思想

棋盘多项式和任务分配

- ◆ 任务分配
- ◆ 有 N 位工作人员，同时有 N 项任务，每人必须承担一项任务，若给出某人不能从事的某些任务，问要安排好工作，共有多少种方案？
- ◆ $N \leq 100$ ，所有人员不能从事的任务之和不大于20。

棋盘多项式和任务分配

- ◆ 先来了解一下错排问题。
- ◆ 如下图所示，每个元素不能对应自己，由容斥原理，所有的排列方法有 $n!(1 - 1/1! + 1/2! - 1/3! + \dots + (-1)^n * 1/n!)$



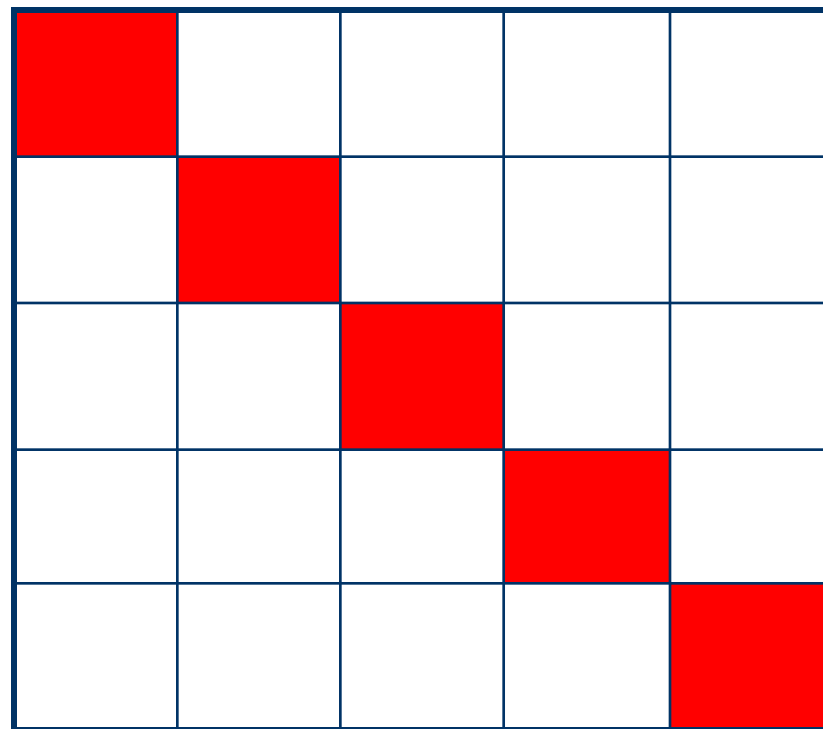
棋盘多项式和任务分配

- ◆ 其实这这也是一个任务分配问题，相当于1号工作人员不能做1号工作，2号工作人员不能做2号工作...以次类推。
- ◆ 可是现在的情况变得一般化了。那么应该怎样来解决这个问题呢？

容斥原理！

棋盘多项式和任务分配

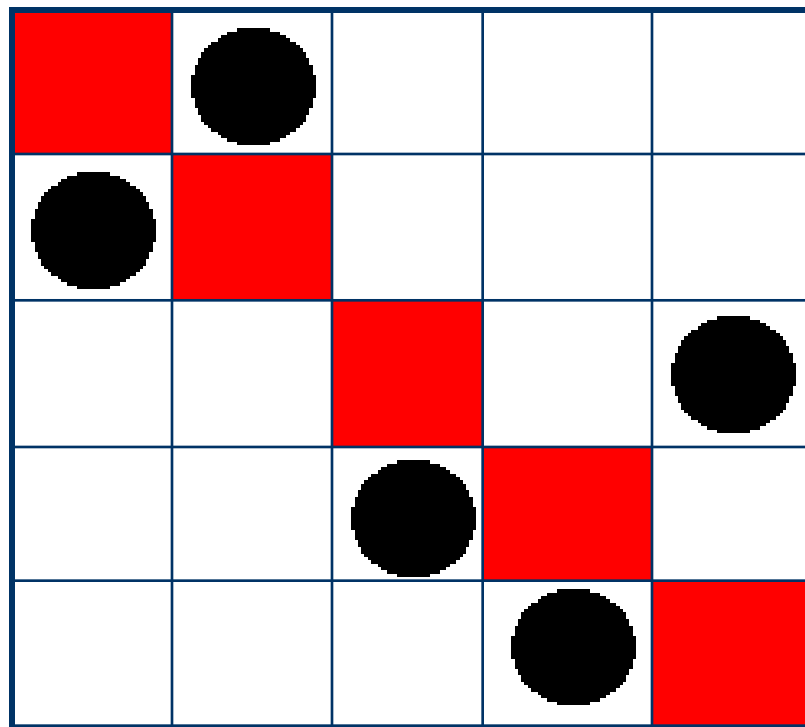
- ◆ 可以列出一个 $N \times N$ 的棋盘，如果 i 号工作人员不能从事 j 号工作，那么在 i, j 处的格子上图红。如错排问题的棋盘表示则是：



Red				
	Red			
		Red		
			Red	
				Red

棋盘多项式和任务分配

- ◆ 我们称红色的区域为禁区，红色的格子为禁位。我们的任务就是把N个棋子放到棋盘中不是禁区的地方，且使得每一行和每一列只有一颗棋子。我们这种排列叫做有限制排列（或者是禁位全排列）



棋盘多项式和任务分配

- 如果我们知道了将 k 科棋子放入禁区的所有放法 $r_k(C)$,其中 C 表示棋盘。那么,由容斥原理,整个禁位排列的数目为:

$$Q_n = n! - (n-1)!r_1 + (n-2)!r_2 - \dots + ((-1)^n)0!r_n$$

棋盘多项式和任务分配

- ◆ 下面的问题变成了如何求 r_k ?对于任何一张棋盘(以后所指的棋盘都是指原棋盘上的禁区部分), $r_0(C)=1$.构造棋盘多项式 $R(C)=r_0(C)x^0+r_1(C)x^1\dots r_k(C)x^k$.
- ◆ 如果我们知道了一张棋盘的棋盘多项式, 那么就知道了任意一个 r_k 。
- ◆ 由上式可知, $R(\Phi)=1$ 。

棋盘多项式和任务分配

- ◆ 那么如何求得任意一块棋盘的棋盘多项式呢？我们可以试着找找递推公式！
- ◆ 不难发现， k 枚棋子对棋盘 C 进行布局的方案，可按选定格子放子与不放子划分为两类：
 1. 对选定的格子放一枚棋子，有 $r_{k-1}(C_i)$ 种。 C_i 表示去掉该个选定格子所在行和列后剩下的棋盘。
 2. 对选定的格子不放棋子，有 $r_k(C_e)$ 种， C_e 表示去掉该个选定的格子以后的棋盘。

棋盘多项式和任务分配

- ◆ 由加法原理可以知道

$$r_k(C) = r_{k-1}(C_i) + r_k(C_e)$$

- ◆ 于是我们利用上面上面的式子对棋盘多项式 $R(C)$ 进行推导。

$$\begin{aligned} R(C) &= \sum_{k=0}^n r_k(C) x^k \\ &= \sum_{k=0}^n [r_{k-1}(C_i) + r_k(C_e)] x^k \\ &= x \sum_{k=0}^n r_k(C_i) x^k + \sum_{k=0}^n r_k(C_e) x^k \\ &= xR(C_i) + R(C_e) \end{aligned}$$

棋盘多项式和任务分配

$$R(\square) = 1 + x;$$

$$R(\begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \end{array}) = xR(\square) + R(\square) = 1 + 2x;$$

$$\begin{aligned} R(\begin{array}{|c|} \hline \square \\ \hline \square \quad \square \\ \hline \end{array}) &= xR(\square) + R(\square) \\ &= x(1 + x) + 1 + x \\ &= 1 + 2x + x^2 \end{aligned}$$

$$\begin{aligned} R(\begin{array}{|c|} \hline \square \\ \hline \square \quad \textcolor{red}{\square} \quad \square \\ \hline \quad \square \quad \square \\ \hline \quad \quad \square \\ \hline \end{array}) &= xR(\begin{array}{|c|} \hline \square \\ \hline \square \\ \hline \end{array}) + R(\begin{array}{|c|} \hline \square \\ \hline \square \quad \square \quad \square \\ \hline \quad \square \\ \hline \end{array}) \\ &= 1 + 6x + 10x^2 + 4x^3 \end{aligned}$$

棋盘多项式和任务分配

- ◆ 把整个棋盘作为参数传递，可以递推出整个棋盘的棋盘多项式。这里当然需要加入一些多项式乘法和加法的东西。
- ◆ 具体分析整个程序的复杂度，可以发现是与禁区的格子数目有着跟大的关系！如果直接这样做的话，复杂度是 $2^{20} \times$ 多项式运算的复杂度。由于是整个数组作为参数在传递，所以这个在时间和空间上都是不理想的。实际测试的时候会有一点超时。

棋盘多项式和任务分配

怎么办



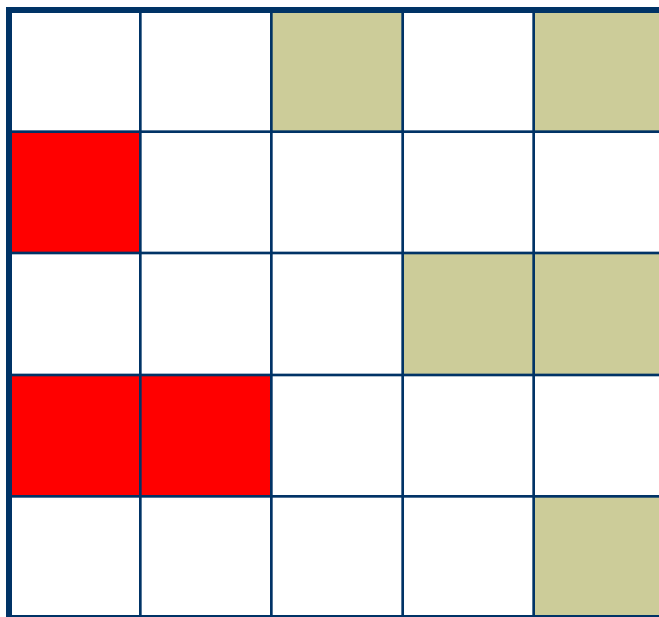
分治!

棋盘多项式和任务分配

- ◆ 先定义两个名称：独立格和独立块。
- ◆ 独立格：两个既不在同一行也不在同一列的格子。
- ◆ 独立块：两个块中的所有格子分别都是独立格。
- ◆ 显然一个独立块也是有他的多项式的，而对于两个独立块，他们放棋子的时候是不会互相影响的？！

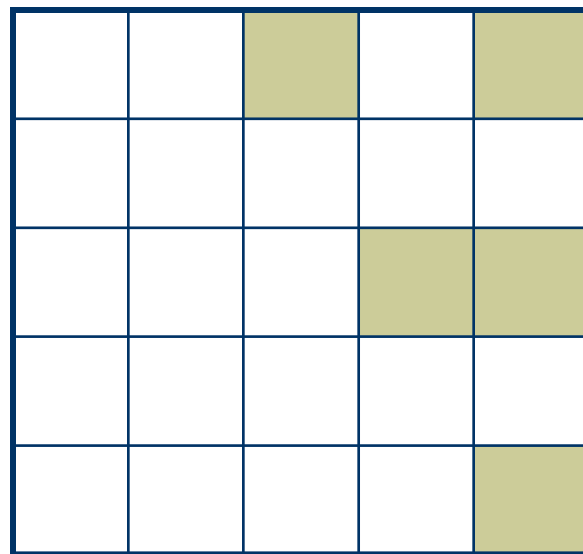
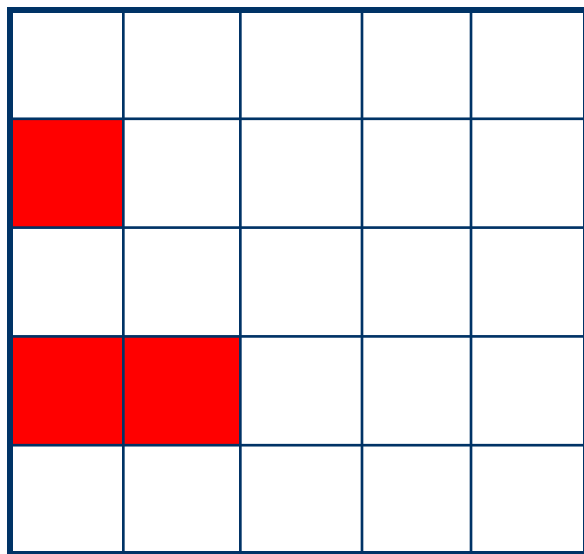
棋盘多项式和任务分配

- ◆ 如图所示，红色和绿色是两个对应的独立块。



棋盘多项式和任务分配

- ◆ 可以把刚才的棋盘分成两块：



分别对这两块棋盘递推出他们的多项式 $R(C_1)$ 和 $R(C_2)$

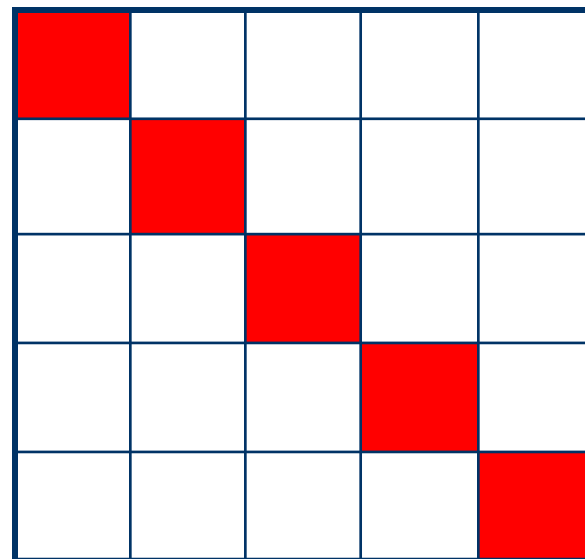
棋盘多项式和任务分配

- ◆ 因为这两块是相对独立的，所以依据乘法原理，整个棋盘的多项式是：

$$R(C) \equiv R(C1) \times R(C2)$$

棋盘多项式和任务分配

- ◆ 举一个简单的例子来说明分治以后的复杂度情况。
- ◆ 上图原本的复杂度是 2^5 ,分治以后的复杂度降为了 5×2^1 .也就是说,如果在禁区格子数为20的情况下,若是能分成两块格子数为10的独立块,那么复杂度由 2^{20} 降为了 2×2^{10} .



棋盘多项式和任务分配

- ◆ 在实际测试的时候，可以发现按上述方法做的程序虽然能运行出答案，但是会出现空间消耗太多错误提示，这是不够理想的。

~~我们把棋盘当作参数在传递！~~

棋盘多项式和任务分配

怎么处理这个问题呢？



从题目的一些特殊条件入手！

棋盘多项式和任务分配

- ◆ 题目中有这样的一句话：
- ◆ 所有人员不能从事的任务之和不大于**20**。
- ◆ 我们处理的只是棋盘的禁区而已，所以在传递参数的时候不用传递整张棋盘，传递每个禁格的位置就行了。空间的占用量似乎是下降了2500倍！！！！

不错!



棋盘多项式和任务分配

◆ 运行结果的比较:

程序 \ 时空	总时间消耗	空间占用量
原搜索标程	10s	小
程序1（原）	>20s	很大
程序2（分）	2s	大
程序3（参）	<0.5s	小

小结

- ◆ 棋盘多项式在信息学竞赛中的运用不是很广泛，似乎只能解决任务分配一类的问题。这里做一个了解即可。

置换群与pólya定理

- ◆ 先看一个例题：
- ◆ 有 n 个人，按 $1-n$ 编号。每个人手里都拿着一个球，这些球也按 $1-n$ 编号。但是1号选手手里拿的不一定是1号球，所以他们可以两两相互交换手里的球，使得最终 i 号选手手里的是 i 号球。请问如果交换可以同时进行，那么最少需要多少个单位时间（两两交换一次是一个单位时间），最少又需要多少次交换？

置换群

- ◆ 在解决这个问题之前，需要先学习一些群论，特别是置换群的知识。这里只介绍一些结论。
- ◆ 将每个人编号和所拿的球的编号一一对应起来写成如下形式：

$$\begin{pmatrix} 1 & 2 & 3 & \dots & n \\ a_1 & a_2 & a_3 & \dots & a_n \end{pmatrix}$$

- ◆ 这称为一个置换群。
- ◆ 如果把*i*和*j*下面的元素交换，称为一次置换。

置换群

- ◆ 例如 $(1\ 2\ 3\ 4)$
- ◆ $(3\ 4\ 1\ 2)$
- ◆ 这是一个置换群。下面进行一种操作，在第一行中找到1，然后看1下面的元素，是3，然后在第一行中找3，3下面的元素是1。这样我们找出了1—3找个循环节，写成： $(1\ 3)$
- ◆ $(3\ 1)$
- ◆ 然后找2，2下面的元素是4，接着4下面的元素又是2，所以又找出了一个循环节 $(2\ 4)$
- ◆ $(4\ 2)$

置换群

- ◆ 在这里，1—3的循环节和2—4的循环节被成为轮换。一个置换群可以写成若干个轮换的乘积：
- ◆ $(1\ 2\ 3\ 4) = (1\ 3) \times (2\ 4)$
- ◆ $(3\ 4\ 1\ 2) = (3\ 1) \times (4\ 2)$
- ◆ 再来看看题目，根据题目给出的信息，可以构造出一个置换群，在该置换群中，可以分解出若干个轮换。试想如果交换在不同的轮换内进行，也就是轮换1中的元素和轮换2中的元素交换。这样必定不会达到最优的交换次数（为什么？）。所以对于每一个轮换，我们可以分别进行处理。

置换群

- ◆ 由于每个轮换都是一个循环，他们的本质是相同的，也就是我们可以对轮换中的元素进行重新编号。于是一个长度为 n 的轮换可以写成如下形式：
- ◆ $(1\ 2\ 3\ \dots\ n-1\ n)$
- ◆ $(2\ 3\ 4\ \dots\ n\ 1)$
- ◆ 问题现在已经一般化了，长度为 n 的轮换，最少需要多少个单位时间的交换和最少需要交换几次呢？当 n 确定时，答案是肯定的。

置换群

- ◆ 结论1：任何一个轮换的，使得所有元素归位的交换次数最少为 $n-1$ 。利用轮换中的任意一个元素和剩下的元素进行交换，可以使得所有元素归位，且交换次数最少。
- ◆ 结论2：交换可以同时进行时，最多需要两个单位的交换时间。做如下操作，将1和 n 下面的元素交换，2和 $n-1$ 下面的元素交换.....这样的交换可以在1个单位时间内进行，完成这次操作后，将2和 n 下面的元素交换，3和 $n-1$ 下面的元素交换.....这也只需要一个单位的交换时间。进行了上述操作后，可以惊奇地发现，所有的元素都已经归位了！交换次数是 $n-1$ 次，这是最少的交换次数。当然当 $n=1$ 时，无需交换时间； $n=2$ 时只要1个单位的交换时间，其他任何情况都只需要2个单位的交换时间就可以了。

书架

- ◆ 书架
- ◆ 书架上有 n 本书，顺序被打乱了。每次可以交换两本编号奇偶性不同的书。要求用最少的步骤，使所有书归位。 $N \leq 1000$ 。

书架

- ◆ 例如有4本书，最初，书架上从左到右的书的编号依次是(2, 4, 1, 3)，我们可以先交换(1, 2)得到(1, 4, 2, 3)，再交换(3, 4)得到(1, 3, 2, 4)，最后交换(3, 2)得到(1, 2, 3, 4)
- ◆ 题目似乎多了一个条件，就是需要奇位和偶位交换。其实道理还是一样的。在一个轮换中，如果奇位和偶位的元素都存在，可以利用任意一个奇位将偶位元素置换到位，也可以利用任意一个偶位将奇数元素置换到位，置换次数还是轮换长度-1。

书架

- ◆ 当轮换中只有奇位元素或者只有偶位元素的时候，这时，需要借助该轮换外的其他元素进行交换了。如果轮换中都是奇位，可以先将轮换中的某个元素和轮换外的一个偶位元素交换，然后利用偶位元素，把所有元素都置换归位。最后再和原来的那个奇位元素进行一次置换，这样，原来的那个奇位元素也归位了。这就是借助轮换外的元素进行交换操作。可以证明，这样的操作能够使得交换次数最少。

无聊的排序

- ◆ 无聊的排序
- ◆ 给出 n 个各不相同的数，每次操作是交换两个数，代价是这两个数的和。请你用最少的代价，使得所有元素从小到大排列。
- ◆ 例如 $(8\ 4\ 5\ 3\ 2\ 7) \Rightarrow (8\ 4\ 5\ 3\ 7\ 2) \Rightarrow (2\ 4\ 5\ 3\ 7\ 8) \Rightarrow (2\ 4\ 3\ 5\ 7\ 8) \Rightarrow (2\ 3\ 4\ 5\ 7\ 8)$ 。这样的交换可以使得代价值之和最小。

无聊的排序

- ◆ 仍然考虑轮换的问题。在一个轮换中，利用任意一个元素和其他元素交换，可以使得所有元素归位。那么我们选取轮换中最小的那个元素，然后和别的元素进行交换，这样可以使得交换的代价值之和“最小”。
- ◆ 但是这个代价值一定最小的吗？为什么要只能在轮换内进行交换呢？会不会和轮换外的元素交换之后，能使代价值更小呢？
- ◆ 答案是肯定的！可以利用轮换外的元素。

无聊的排序

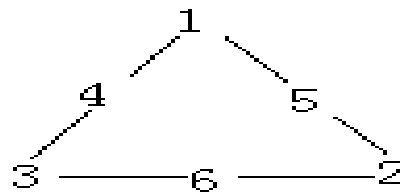
- ◆ 举个例子：
- ◆ (1 8 9 7 6)，轮换为(1)(8 9 7 6)。目标状态是(1 6 7 8 9)，先把轮换外最小元素1和轮换中最小元素6交换 \Rightarrow (6 8 9 7 1)，然后利用1使得轮换中所有元素归位 \Rightarrow (6 8 1 7 9) \Rightarrow (6 8 7 1 9) \Rightarrow (6 1 7 8 9)，最后再把1交换出去 \Rightarrow (1 6 7 8 9)。这样的操作可以使得代价值更加小。所以在对一个轮换进行处理的时候，应该考虑两种交换方式，选择更优秀的交换方式进行。

pólya定理

- ◆ pólya定理是用来解决一般染色问题的有效方法。
- ◆ 例如给出一个三角形，要求你对三角形的边和点进行染色，有 m 种颜色可以供你选择，每条边和每个点必须染色。问你经过翻转旋转等操作后，仍然本质不同的三角形个数有多少种。

pólya定理

- ◆ 对顶点和边进行标号，例如一个正三角形可以这样标号：



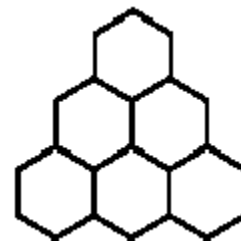
- ◆ 把本质不同的旋转翻转操作，用置换群表示。例如向右旋转 60° 的操作可以表示成
- ◆ $(1\ 2\ 3\ 4\ 5\ 6)$
- ◆ $(3\ 1\ 2\ 6\ 4\ 5)$
- ◆ 表示1的地方被3代替，2的地方被1代替。

pólya定理

- ◆ 假设可以列出 k 个置换群，且在第 i 个置换群中轮换的个数为 f_i ，伟大的数学家pólya研究出了一个让人兴奋的结论：用 m 种颜色对图形进行染色，本质不同的染色方案数为 $(m^{f_1} + m^{f_2} + \dots + m^{f_k})/k$ 。
- ◆ 这是一个伟大的结论，对于信息学竞赛来说，只需要记住这个结论即可。

pólya定理

- ◆ 黑白瓷砖
- ◆ 小可可在课余的时候受美术老师的委派从事一项漆绘瓷砖的任务。首先把 $(n+1) \times n/2$ 块正六边形瓷砖拼成三角形的形状下图给出了 $n=3$ 时拼成的“瓷砖三角形”。然后把每一块瓷砖漆成纯白色或者纯黑色，而且每块瓷砖的正、反两面都必须漆成同样的颜色。
- ◆ 有一天小可可突发奇想，觉得有必要试试看这些瓷砖究竟能漆成多少种本质不同的图案。所谓两种图案本质不同就是其中的一种图案无论如何旋转、或者翻转、或者同时旋转和翻转都不能得到另外一种图案。

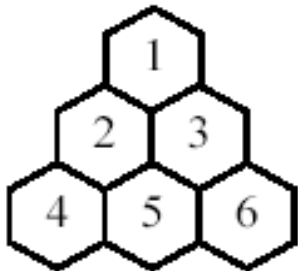
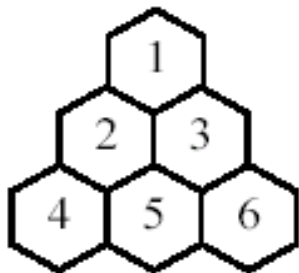


pólya定理

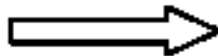
- ◆ 旋转是将瓷砖三角形整体顺时针旋转 120° 或 240° 。

以 $n=3$ 为例。为观察方便，将每块瓷砖都编上号。

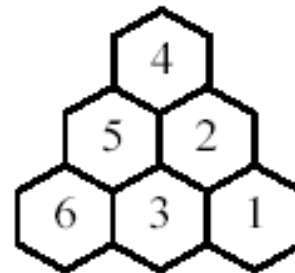
旋转前的图案



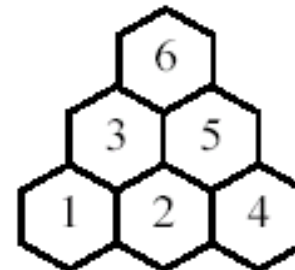
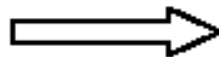
顺时针旋转
120 度后



旋转后的图案



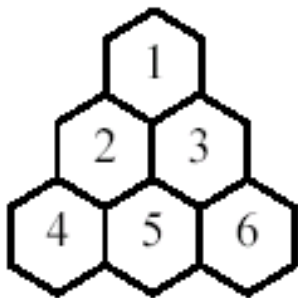
顺时针旋转
240 度后



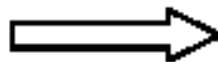
pólya定理

- ♦ 翻转是将瓷砖三角形整体左右翻转 180° 。

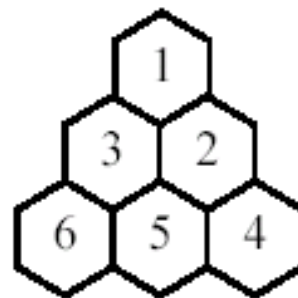
以 $n=3$ 为例。
翻转前的图案



翻转后

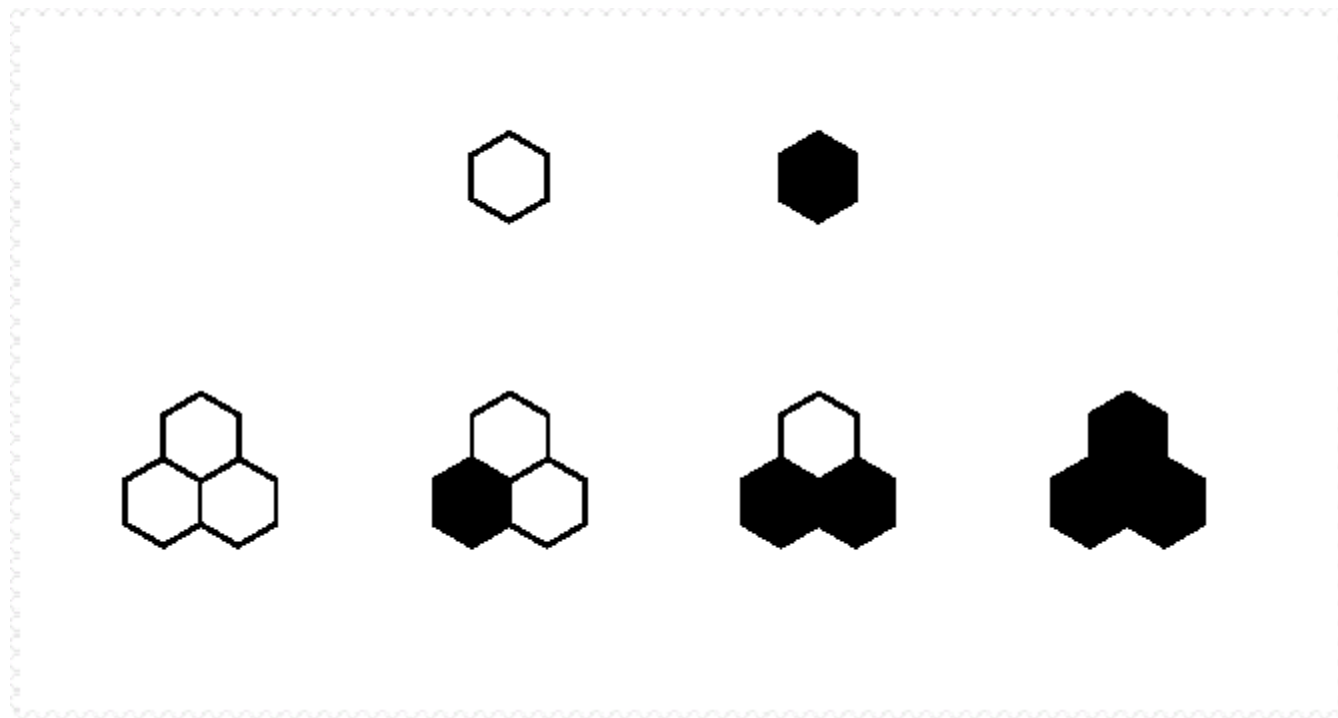


翻转后的图案



pólya定理

- 一开始，小可可觉得这项实验很有意思，他知道 $n=1$ 时有两种不同的漆绘方案， $n=2$ 时也只有4种不同的漆绘方案，小可可还把这些方案画了出来。



pólya定理

- ◆ 但是后来小可可发现 n 在变大的过程中，漆绘方案增长的速度很快，在 $n=14$ 的时候，居然有6760803201217259503457555972096种不同的漆绘方案。这果然是一项非常艰巨的实验。因此他决定请你编写程序帮他求解本质不同的漆绘方案数 s 。

pólya定理

- ◆ 这一题显然可以直接套用pólya定理。
- ◆ 首先需要做一件比较麻烦的事情，就是找出每一个置换群。题目给出的信息告诉我们，所有的重复方案都是由旋转和翻转的组合得到的。这里定义两个过程，一个是左右翻转，一个是顺时针旋转 120° 。这样调用这两个过程可以得出所有的重复方案。共6种。

pólya定理

- ◆ Procedure turn(Var a:SetType);//旋转
- ◆ var i,j:integer;
- ◆ tmp:SetType;
- ◆ begin
- ◆ tmp:=a;
- ◆ for i:=1 to n do
- ◆ for j:=1 to i do
- ◆ a[i,j]:=tmp[n-j+1,i-j+1];
- ◆ end;

pólya定理

- ◆ Procedure turnover(Var a:SetType);//翻转
- ◆ var i,j:integer;
- ◆ tmp:SetType;
- ◆ begin
- ◆ tmp:=a;
- ◆ for i:=1 to n do
- ◆ for j:=1 to i do
- ◆ a[i,j]:=tmp[i,i-j+1];
- ◆ end;

pólya定理

- ◆ 剩下的任务只要知道pólya定理就可以容易地解决。
- ◆ 用DFS求出每个置换群中的轮换个数，然后进行高精度运算。

小结

- ◆ 置换群和pólya定理在信息学竞赛中的运用比较广泛，近年来很多题目都涉及到此类解法。这两个知识的程序实现并不复杂，但是理论性很强，这里也只是做了一个结论的介绍。具体的证明请参照离散数学的专业书籍。