# Pólya原理及其应用

华东师大二附中 符文杰

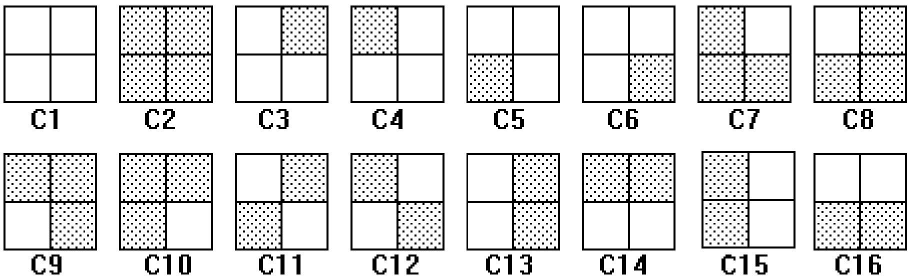
Pólya原理是组合数学中，用来计算全部互异的组合状态的个数的一个十分高效、简便的工具。下面，我就向大家介绍一下什么是Pólya原理以及它的应用。请先看下面这道例题：

**【例题1】**

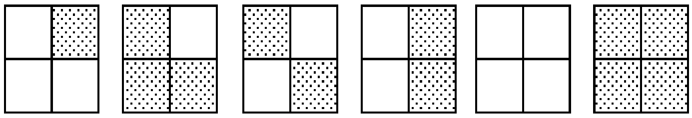
对2*2的方阵用黑白两种颜色涂色，问能得到多少种不同的图像？经过旋转使之吻合的两种方案，算是同一种方案。

**【问题分析】**

由于该问题规模很小，我们可以先把所有的涂色方案列举出来。




一个2*2的方阵的旋转方法一共有4种：旋转0度、旋转90度、旋转180度和旋转270度。(注：本文中默认旋转即为顺时针旋转) 我们经过尝试，发现其中互异的一共只有6种：C3、C4、C5、C6是可以通过旋转相互变化而得，算作同一种；C7、C8、C9、C10是同一种；C11、C12是同一种；C13、C14、C15、C16也是同一种； C1和C2是各自独立的两种。于是，我们得到了下列6种不同的方案。



但是，一旦这个问题由2*2的方阵变成20*20甚至200*200的方阵，我们就不能再一一枚举了，利用Pólya原理成了一个很好的解题方法。在接触Pólya原理之前，首先简单介绍Pólya原理中要用到的一些概念。

**群：**给定一个集合$G=\{a,b,c,\ldots\}$和集合$G$上的二元运算，并满足:

(a) 封闭性：$\forall a,b\in G, \exists c\in G, a * b=c$。

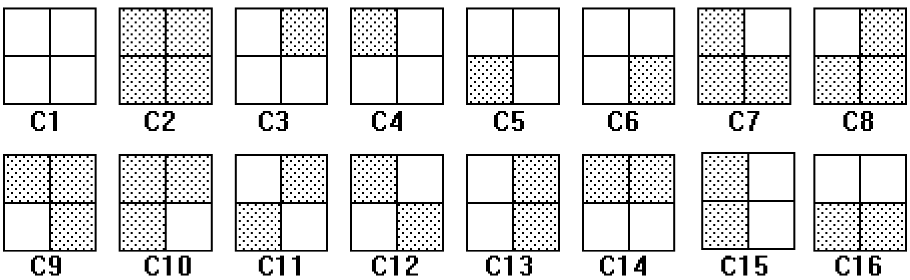
(b) 结合律：$\forall a,b,c\in G, (a * b) * c=a * (b * c)$。

(c) 单位元：$\exists e\in G, \forall a\in G, a * e=e * a=a$。

(d) 逆元：$\forall a\in G, \exists b\in G, a * b=b * a=e$，记$b=a^{-1}$。

则称集合$G$在运算 * 之下是一个群，简称$G$是群。一般$a * b$简写为$ab$。

**置换：**$n$个元素$1,2,\ldots,n$之间的一个置换$\begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix}$表示1被1到$n$中的某个数$a_1$取代，2被1到$n$中的某个数$a_2$取代，直到$n$被1到$n$中的某个数$a_n$取代，且$a_1,a_2,\ldots,a_n$互不相同。本例中有4个置换：




转0° $a1=\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{pmatrix}$

转90° $a2=\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 2 & 6 & 3 & 4 & 5 & 10 & 7 & 8 & 9 & 12 & 11 & 16 & 13 & 14 & 15 \end{pmatrix}$

转180° $a3=\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 2 & 5 & 6 & 3 & 4 & 9 & 10 & 7 & 8 & 11 & 12 & 15 & 16 & 13 & 14 \end{pmatrix}$

转270° $a4=\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 1 & 2 & 4 & 5 & 6 & 3 & 8 & 9 & 10 & 7 & 12 & 11 & 14 & 15 & 16 & 13 \end{pmatrix}$

**置换群：**置换群的元素是置换，运算是置换的连接。例如：

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}\begin{pmatrix} 3 & 1 & 2 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}=\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}$$

可以验证置换群满足群的四个条件。

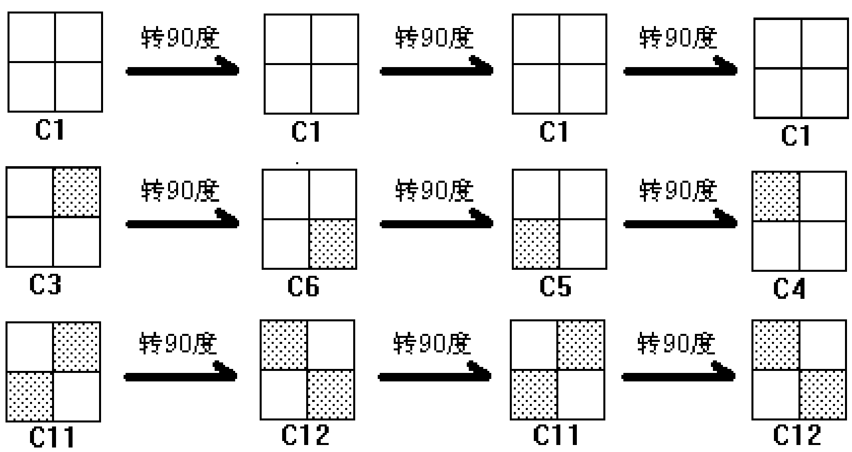
本题中置换群G={转0°、转90°、转180°、转270°}

我们再看一个公式：$|E_k|\cdot|Z_k|=|G|\quad k=1\ldots n$

该公式的一个很重要的研究对象是群的元素个数，有很大的用处。

**$Z_k$ (K不动置换类)：**设G是1…n的置换群。若K是1…n中某个元素，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。

*如本例中：*G是涂色方案1~16的置换群。对于方案1，四个置换都使方案1保持不变，所以$Z_1=\{a_1, a_2, a_3, a_4\}$；对于方案3，只有置换$a_1$使其不变，所以$Z_3=\{a_1\}$；对于方案11，置换$a_1$和$a_3$使方案其保持不变，




所以$Z_{11}=\{a_1, a_3\}$。

**$E_k$(等价类)：**设G是1…n的置换群。若K是1…n中某个元素，K在G作

用下的轨迹，记作$E_k$。即K在G的作用下所能变化成的所有元素的集合。

*如本例中：*方案1在四个置换作用下都是方案1，所以$E_1=\{1\}$；方案3，在$a_1$下是3，在$a_2$下变成6，在$a_3$下变成5，在$a_4$下变成4，所以$E_3=\{3,4,5,6\}$；方案11，在$a_1$、$a_3$下是11，在$a_2$、$a_4$下变成12，所以$E_{11}=\{11,12\}$。

本例中的数据，也完全符合这个定理。*如本例中：*

$|E_1|\cdot|Z_1| = 1\times4 = 4 = |G|$

$|E_3|\cdot|Z_3| = 4\times1 = 4 = |G|$

$|E_{11}|\cdot|Z_{11}| = 2\times2 = 4 = |G|$

限于篇幅，这里就不对这个定理进行证明。

接着就来研究每个元素在各个置换下不变的次数的总和。见下表：

| 置换\$S_{ij}$\元素j<br>$a_I$ | 1 | 2 | …… | 16 | $D(a_i)$ |
|---|---|---|---|---|---|
| a1 | $S_{1,1}$ | $S_{1,2}$ | …… | $S_{1,16}$ | $D(a_1)$ |
| a2 | $S_{2,1}$ | $S_{2,2}$ | …… | $S_{2,16}$ | $D(a_2)$ |
| a3 | $S_{3,1}$ | $S_{3,2}$ | …… | $S_{3,16}$ | $D(a_3)$<br>$\sum_{j=1}^{16}\lvert Z_j\rvert=\sum_{j=1}D(a_j)$ |
| a4 | $S_{4,1}$ | $S_{4,2}$ | …… | $S_{4,16}$ | $D(a_4)$ |
| $\lvert Z_j\rvert$ | $\lvert Z_1\rvert$ | $\lvert Z_2\rvert$ | …… | $\lvert Z_{16}\rvert$ | |

其中

$$S_{ij}=\begin{cases}0 & \text{当}a_i\notin Z_j,\text{即}j\text{在}a_i\text{的变化下变动了}\\ 1 & \text{当}a_i\in Z_j,\text{即}j\text{在}a_i\text{的变化下没有变}\end{cases}$$

$D(a_j)$ 表示在置换$a_j$下不变的元素的个数

*如本题中：*涂色方案1在$a_1$下没变动，$S_{1,1}=1$；方案3在$a_3$变动了，$S_{3,3}=0$；在置换$a_1$的变化下16种方案都没变动，$D(a_1)=16$；在置换$a_2$下只有1、2这两种方案没变动，$D(a_2)=2$。

一般情况下，我们也可以得出这样的结论： $\sum_{j=1}^{n}|Z_j|=\sum_{i=1}^{s}D(a_i)$

我们对左式进行研究。

不妨设N={1，......，n}中共有L个等价类，$N=E_1+E_2+\ldots\ldots+E_L$，则当j和k属于同一等价类时，有$|Z_j|=|Z_k|$。所以

$$\sum_{k=1}^{n}|Z_k|=\sum_{i=1}^{L}\sum_{k\in E_i}|Z_k|=\sum_{i=1}^{L}|E_i|\cdot|Z_i|=L\cdot|G|$$

这里的L就是我们要求的互异的组合状态的个数。于是我们得出：

$$L=\frac{1}{|G|}\sum_{k=1}^{n}|Z_k|=\frac{1}{|G|}\sum_{j=1}^{s}D(a_j)$$

利用这个式子我们可以得到本题的解 L=(16+2+4+2)/4=6 与前面枚举得到的结果相吻合。这个式子叫做Burnside引理。

但是，我们发现要计算$D(a_j)$的值不是很容易，如果采用搜索的方法，总的时间规模为$O(n\times s\times p)$。(n表示元素个数，s表示置换个数，p表示格子数，这里n的规模是很大的) 下一步就是要找到一种简便的$D(a_j)$的计算方法。先介绍一个循环的概念：

**循环：**记

$$(a_1a_2\cdots a_n)=\begin{pmatrix}a_1 & a_2 & \cdots & a_{n-1} & a_n\\ a_2 & a_3 & \cdots & a_n & a_1\end{pmatrix}$$

称为$n$阶循环。每个置换都可以写若干互不相交的循环的乘积，两个循环$(a_1a_2\ldots a_n)$和$(b_1b_2\ldots b_n)$互不相交是指$a_i \neq b_j$, $i,j=1,2,\ldots,n$。例如：

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 4 & 2 \end{pmatrix} = (13)(25)(4)$$

这样的表示是唯一的。置换的循环节数是上述表示中循环的个数。例如(13)(25)(4)的循环节数为3。

有了这些基础，就可以做进一步的研究，我们换一个角度来考虑这个问题。我们给2*2方阵的每个方块标号，如下图：

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

构造置换群$G'=\{g_1,g_2,g_3,g_4\}$, $|G'|=4$，令$g_i$的循环节数为$c(g_i)$ $(i=1,2,3,4)$

在G'的作用下，其中

$g_1$表示转0°，　即$g_1=(1)(2)(3)(4)$　　$c(g_1)=4$

$g_2$表示转90°，　即$g_2=(4\ 3\ 2\ 1)$　　$c(g_2)=1$

$g_3$表示转180°，即$g_3=(1\ 3)(2\ 4)$　　$c(g_3)=2$

$g_4$表示转270°，即$g_4=(1\ 2\ 3\ 4)$　　$c(g_4)=1$

我们可以发现，$g_i$的同一个循环节中的对象涂以相同的颜色所得的图像数$m^{c(g_i)}$正好对应G中置换$a_i$作用下不变的图象数，即

$2^{c(g_1)}=2^4=16=D(a_1)$　　$2^{c(g_2)}=2^1=2=D(a_2)$

$2^{c(g_3)}=2^2=4=D(a_3)$　　$2^{c(g_4)}=2^1=2=D(a_4)$

由此我们得出一个结论：

设G是p个对象的一个置换群，用m种颜色涂染p个对象，则不同

$$L=\frac{1}{|G|}\left(m^{c(g_1)}+m^{c(g_2)}+\cdots+m^{c(g_s)}\right)$$

染色方案为：

其中$G=\{g_1,\ldots g_s\}$　$c(g_i)$为置换$g_i$的循环节数$(i=1\ldots s)$

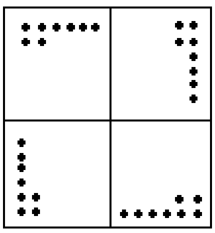
这就是所谓的Pólya定理。我们发现利用Pólya定理的时间复杂度为$O(s\times p)$ (这里s表示置换个数，p表示格子数)，与前面得到的Burnside引理相比之下，又有了很大的改进，其优越性就十分明显了。Pólya定理充分挖掘了研究对象的内在联系，总结了规律，省去了许多不必要的盲目搜索，把解决这类问题的时间规模降到了一个非常低的水平。

现在我们把问题改为：$n\times n$的方阵，每个小格可涂m种颜色，求在旋转操作下本质不同的解的总数。

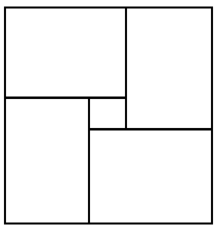
**【问题分析】**

先看一个很容易想到的搜索的方法。(见附录)

这样搜索的效率是极低的，它还有很大的改进的余地。前面，我们采用的方法是先搜后判，这样的盲目性极高。我们需要边搜边判，避免过多的不必要的枚举，我们更希望把判断条件完全融入到搜索的边界中去，消灭无效的枚举。这个美好的希望是可以实现的。



n为偶数时



n为奇数时

我们可以在方阵中分出互不重叠的长为$[(n+1)/2]$，宽为$[n/2]$的四个矩阵。当n为偶数时，恰好分完；当n为奇数时，剩下中心的一

个格子，它在所有的旋转下都不动，所以它涂任何颜色都对其它格子没有影响。令m种颜色为0~m-1，我们把矩阵中的每格的颜色所代表的数字顺次(左上角从左到右，从上到下；右上角从上到下，从右到左；......)排成m进制数，然后就可以表示为一个十进制数，其取值范围为$0\sim m^{[n_2/4]}-1$。(因为$[n/2]*[(n+1)/2]=[n^2/4]$) 这样，我们就把一个方阵简化为4个整数。我们只要找到每一个等价类中左上角的数最大的那个方案(如果左上角相同，就顺时针方向顺次比较) 这样，在枚举的时候其它三个数一定不大于左上角的数，效率应该是最高的。

进一步考虑，当左上角数为i时,$(0\le i\le R-1)$ 令$R=m^{[n_2/4]}$
可分为下列的4类：

① 其它三个整数均小于i，共$i^3$个。

② 右上角为i，其它两个整数均小于i，共$i^2$个。

③ 右上角、右下角为i，左下角不大于i，共i+1个。

④ 右下角为i，其它两个整数均小于i，且右上角的数不小于左下角的，共i(i+1)/2个。

$$
\begin{aligned}
L &= \sum_{i=0}^{R-1}(\mathrm{i}^3+\mathrm{i}^2+i+1+\frac{1}{2}i(i+1)) = \sum_{i=0}^{R-1}(\mathrm{i}^3+\frac{3}{2}\mathrm{i}^2+\frac{3}{2}i+1) \\
&= \sum_{i=1}^{R}((i-1)^3+\frac{3}{2}(\mathrm{i}-1)^2+\frac{3}{2}(i-1)+1) = \sum_{i=1}^{R}(\mathrm{i}^3-\frac{3}{2}\mathrm{i}^2+\frac{3}{2}i) \\
&= \frac{1}{4}R^2(R+1)^2-\frac{3}{2}\times\frac{1}{6}R(R+1)(2R+1)+\frac{3}{2}\times\frac{1}{2}R(R+1) \\
&= \frac{1}{4}(R^4+R^2+2R)
\end{aligned}
$$

因此，

当n为奇数时，还要乘一个m。

由此我们就巧妙地得到了一个公式。但是，我们应该看到要想到这个公式需要很高的智能和付出不少的时间。另一方面，这种方法只能对这道题有用而不能广泛地应用于一类试题，具有很大的不定性因素。因此，如果能掌握一种适用面广的原理，就会对解这一类题有很大的帮助。

下面我们就采用Pólya定理。我们可以分三步来解决这个问题。

1．确定置换群

在这里很明显只有4个置换：转0°、转90°、转180°、转270°。所以，置换群G={转0°、转90°、转180°、转270°}。

2．计算循环节个数

首先，给每个格子顺次编号（$1\sim n^2$），再开一个二维数组记录置换后的状态。最后通过搜索计算每个置换下的循环节个数，效率为一次方级。

3．代入公式

即利用Pólya定理得到最后结果。

$$L=\frac{1}{|G|}(m^{c(g_1)}+m^{c(g_2)}+\cdots+m^{c(g_s)})$$

【程序题解】

```
const
  maxn=10;
var
  a,b:array[1..maxn,1..maxn] of integer;{记录方阵的状态}
  i,j,m,n:integer;{m颜色数;n方阵大小}
  l,l1:longint;
```

```
procedure xz;{将方阵旋转90°}
var
  i,j:integer;
begin
  for i:=1 to n do
    for j:=1 to n do
      a[j,n+1-i]:=b[i,j];
  b:=a
end;
procedure xhj;{计算当前状态的循环节个数}
var
  i,j,i1,j1,k,p:integer;
begin
  k:=0;{用来记录循环节个数，清零}
  for i:=1 to n do
    for j:=1 to n do
      if a[i,j]>0 then{搜索当前尚未访问过的格子}
        begin
          inc(k);{循环节个数加1}
          i1:=(a[i,j]-1) div n;
          j1:=(a[i,j]-1) mod n+1;{得到这个循环的下一个格子}
          a[i,j]:=0;{表示该格已访问}
          while a[i1,j1]>0 do begin
            p:=a[i1,j1];{暂存当前格的信息}
            a[i1,j1]:=0;{置已访问标志}
            i1:=(p-1) div n+1;
            j1:=(p-1) mod n+1{得到这个循环的下一个格子}
          end{直到完整地访问过这个循环后退出}
        end;
  l1:=1;
  for i:=1 to k do l1:=l1*m;{计算m的k次方的值}
```

```
  l:=l+l1{进行累加}
end;
begin
  writeln('Input m,n=');
  readln(m,n);{输入数据}
  for i:=1 to n do
    for j:=1 to n do a[i,j]:=(i-1)*n+j;{对方阵的状态进行初始化}
  b:=a;
  xhj;{计算转0°状态下的循环节个数}
  xz;{转90°}
  xhj; {计算转90°状态下的循环节个数}
  xz;{再转90°}
  xhj; {计算转180°状态下的循环节个数}
  xz;{再转90°}
  xhj; {计算转270°状态下的循环节个数}
  l:=l div 4;
  writeln(l);{输出结果}
  readln
end.
```

在上面的程序中，我暂时回避了高精度计算，因为这和我讲的内容关系不大。

如果大家再仔细地考虑一下，就会发现这个题解还可以继续优化。对n分情况讨论：

① n为偶数：在转0°时，循环节为$n^2$个，转180°时，循环节为$n^2/2$个，转90°和转270°时，循环节为$n^2/4$个。

② n为偶数：在转0°时，循环节为$n^2$个，转180°时，循环节为$(n^2+1)/2$个，转90°和转270°时，循环节为$(n^2+3)/4$个。

把这些综合一下就得到：在转0°时，循环节为$n^2$个，转180°时，循环节为$[(n^2+1)/2]$个，转90°和转270°时，循环节为$[(n^2+3)/4]$个。(其中，方括号表示取整)于是就得到：

$$L=\frac{1}{4}\left(m^{n^2}+m^{[\frac{n^2+3}{4}]}+m^{[\frac{n^2+1}{2}]}+m^{[\frac{n^2+3}{4}]}\right)$$

这和前面得到的结果完全吻合。

经过上述一番分析，使得一道看似很棘手的问题得以巧妙的解决，剩下的只要做一点高精度计算即可。

通过这几个例子，大家一定对Pólya原理有了八九成的了解，通过和搜索方法的对比，它的优越性就一目了然了。它不仅极大地提高了程序的时间效率，甚至在编程难度上也有减无增。所以，我们在智能和经验不断增长的同时，也不能忽视了原理性的知识。智能和经验固然重要，但是掌握了原理就更加踏实。因此，我们在解题之余，也要不忘对原理性知识的学习，不停给自己充电，使自己的水平有更大的飞跃。

**附录 (搜索方法的程序)**

```
const
  maxn=10;
type
  sqtype=array[1..maxn,1..maxn] of byte;
var
  n,total,m:integer;
  sq:sqtype;
```

```
function big:boolean;(检验当前方案是否为同一等价类中最大的)
var
  units:array[2..4] of sqtype;(记录三种旋转后的状态)
  i,j,k:integer;
begin
  for i:=1 to n do
    for j:=1 to n do begin
      units[2,j,n+1-i]:=sq[i,j];
      units[3,n+1-i,n+1-j]:=sq[i,j];
      units[4,n+1-j,i]:=sq[i,j]
    end;
  big:=false;
  for k:=2 to 4 do (进行比较)
    for i:=1 to n do begin
      j:=1;
      while (j<=n)and(sq[i,j]=units[k,i,j]) do inc(j);
      if j<=n then
        if sq[i,j]<units[k,i,j]
          then exit
          else break
    end;
  big:=true
end;
procedure make(x,y:byte);(枚举每个格子中涂的颜色)
var i:integer;
begin
  if x>n then begin
    if big then inc(total);
    exit
  end;
  for i:=1 to m do begin
    sq[x,y]:=i;
    if y=n then make(x+1,1) else make(x,y+1)
  end
end;
begin
  writeln('Input m,n=');readln(m,n);
  total:=0;
  make(1,1);
  writeln(total);readln
```

end.