置换群快速幂运算 研究与探讨

江苏省苏州中学 潘震皓

[关键词] 置换 循环 分裂 合并

[摘要]

群是一个古老的数学分支，近几年来在程序设计中置换群得到了一定的应用。本文针对置换群的特点提出了线性时间的幂运算算法，并举例说明了优化后算法的效果。

[正文]

一、引言

置换群是一种优秀的结构，在程序设计中，它的大部分基本操作，时间和空间复杂度都是线性的，甚至有的还是常数的。所以一个问题如果能够抽象归结为一个置换群模型的话，往往能够在程序设计中轻松地解决。但是对于整幂运算来说，似乎只能通过反复做乘法来获得 O(k*乘法)或是 O(logk*乘法)的算法；而对于分数幂运算，则找不到较好的方法实现。

二、置换群的整幂运算

2.1 整幂运算的一个转化

在置换群中有一个定理：设$T^k = e$，(T 为一置换，e 为单位置换（映射函数为 $f(x) = x$ 的置换))，那么 k 的最小正整数解是 T 的拆分的所有循环长度的最小公倍数。

或者有个更一般的结论：设$T^k = e$，(T 为一循环，e 为单位置换)，那么 k 的最小正整数解为 T 的长度。

我们知道，单位置换就是若干个只含单个元素的循环的并。也就是说，长度为1的循环，1 次的幂，把所有元素都完全分裂了。这是为什么呢？

批注: 在这里，第一次出现了“分裂”。在后面的叙述中，将会多次出现。它的意思，通常是指 将一个循环按照 mod k 的值平均地分成 k 个循环。

我们来做一个试验：(下面的置换均以循环的连接表示)

设 n=6，那么$T^6 = (T^2)^3$。任取一 T=(1 3 5 2 4 6)，来做一遍乘法：

$$T^2 = \begin{pmatrix}1&2&3&4&5&6\\3&4&5&6&2&1\end{pmatrix}\begin{pmatrix}1&2&3&4&5&6\\3&4&5&6&2&1\end{pmatrix}$$
$$= \begin{pmatrix}1&2&3&4&5&6\\3&4&5&6&2&1\end{pmatrix}\begin{pmatrix}3&4&5&6&2&1\\5&6&2&1&4&3\end{pmatrix}$$
$$= \begin{pmatrix}1&2&3&4&5&6\\5&6&2&1&4&3\end{pmatrix}$$
$$= (1\ \ 5\ \ 4)(2\ \ 6\ \ 3)$$

分裂成了 2 份！而且这 2 份恰好是 T 的奇数项和偶数项！(注意可以写成(1 5 4)(3 2 6))

再来看看T^3：

$$
\begin{aligned}
T^3 &= T^2T \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 2 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix}\begin{pmatrix} 5 & 6 & 2 & 1 & 4 & 3 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{pmatrix} \\
&= (1\ \ 2)(3\ \ 4)(5\ \ 6)
\end{aligned}
$$

不出所料，分裂成了 3 份，每份分别是在原来循环中的位置 mod 3=0, 1, 2 的项。

继续看T^4：

$$
\begin{aligned}
T^4 &= T^2T^2 \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix}\begin{pmatrix} 5 & 6 & 2 & 1 & 4 & 3 \\ 4 & 3 & 6 & 5 & 1 & 2 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 3 & 6 & 5 & 1 & 2 \end{pmatrix} \\
&= (1\ \ 4\ \ 5)(2\ \ 3\ \ 6)
\end{aligned}
$$

与前面不同的是，循环只分裂成了 2 份。并且每一份的循环看起来都是杂乱无章的，只知道是在循环中的奇数项和偶数项。

再来拿T^5做个试验：

$$
\begin{aligned}
T^5 &= T^2T^3 \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 3 & 6 & 5 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 2 & 1 & 4 & 3 \end{pmatrix}\begin{pmatrix} 5 & 6 & 2 & 1 & 4 & 3 \\ 6 & 5 & 1 & 2 & 3 & 4 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 5 & 1 & 2 & 3 & 4 \end{pmatrix} \\
&= (1\ \ 6\ \ 4\ \ 2\ \ 5\ \ 3)
\end{aligned}
$$

这次的循环，根本没有分裂，只是顺序改变了一下。

这之间有什么共同点呢？对，那就是 gcd(k,1)。
因为 gcd(6,6)=6，所以循环会完全分裂，而 gcd(2,6)=2, gcd(3,6)=3, gcd(4,6)=2, gcd(5,6)=1 也相对应了上面的每一个试验的结果。

经过多次试验以后，我们得到三个结论：

结论一：一个长度为 l 的循环 T，l 是 k 的倍数，则T^k是 k 个循环的乘积，每个循环分别是循环 T 中下标 i mod k=0,1,2…的元素按顺序的连接。

结论二：一个长度为 l 的循环 T，gcd(l,k)=1，则T^k是一个循环，与循环 T 不一定相同。

结论三：一个长度为 l 的循环 T，T^k是 gcd(l,k)个循环的乘积，每个循环分别是循环 T 中下标 i mod gcd(l,k)=0,1,2…的元素的连接。

可以看出，结论三只不过是把 k 分成 gcd(l,k)*(l/gcd(l,k))，再运用结论一和结论二所得到的。如果这几个结论是正确的话，那显然只需要确定结论二中叙述的T^k，就能够在 O(n)内解决任意循环的任意整幂运算了。

2.2 循环长度与指数互质时的整幂运算

和上面一样，我们来做几个试验。
设 T=(1 2 5 3 4)，则：

$$T^2=\begin{pmatrix}1&2&3&4&5\\2&5&4&1&3\end{pmatrix}\begin{pmatrix}1&2&3&4&5\\2&5&4&1&3\end{pmatrix}$$
$$=\begin{pmatrix}1&2&3&4&5\\2&5&4&1&3\end{pmatrix}\begin{pmatrix}2&5&4&1&3\\5&3&1&2&4\end{pmatrix}$$
$$=\begin{pmatrix}1&2&3&4&5\\5&3&1&2&4\end{pmatrix}$$
$$=\begin{pmatrix}1&5&4&2&3\end{pmatrix}$$

不知大家有没有注意到，如果把循环T的奇数项和偶数项取出来，就是 1 5 4 和 2 3，如果两者并在一起，就是刚才求出的T^2了。再试一个：

$$T^3=T^2T$$
$$=\begin{pmatrix}1&2&3&4&5\\5&3&1&2&4\end{pmatrix}\begin{pmatrix}1&2&3&4&5\\2&5&4&1&3\end{pmatrix}$$
$$=\begin{pmatrix}1&2&3&4&5\\5&3&1&2&4\end{pmatrix}\begin{pmatrix}5&3&1&2&4\\3&4&2&5&1\end{pmatrix}$$
$$=\begin{pmatrix}1&2&3&4&5\\3&4&2&5&1\end{pmatrix}$$
$$=\begin{pmatrix}1&3&2&4&5\end{pmatrix}$$

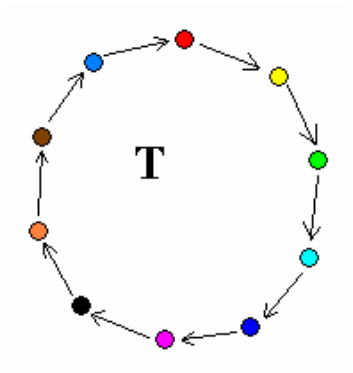
同样地，把 mod 3=1, 2, 0 的项取出来：1 3、2 4、5，连接在一起，就是所求得的新循环了。
把这一试验结果写成一个定理，就是：

设$a=T$，$a'=T^k$，且gcd(l,k)=1，则$a'[i]=a[(k+1)i \bmod l]$。

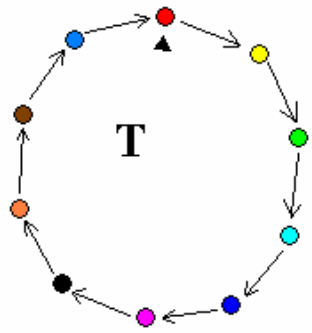
批注：我们预先定义，若数组$a=T$，则 a 按顺序包含了循环 T，下标范围$0 \le i \le l-1$，且 a[0]包含了循环中最小的元素

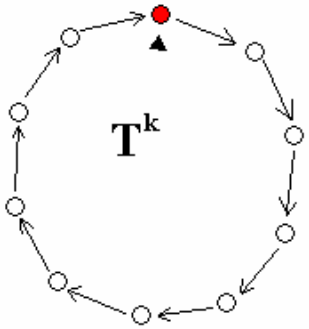
这个定理看起来似乎挺复杂，但如果画张图看，一点也不复杂：

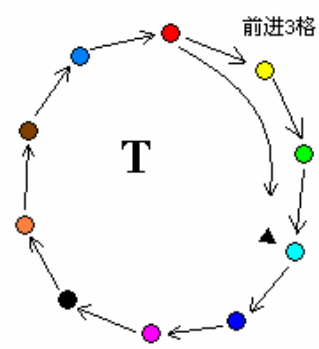
设 l=10，k=3：

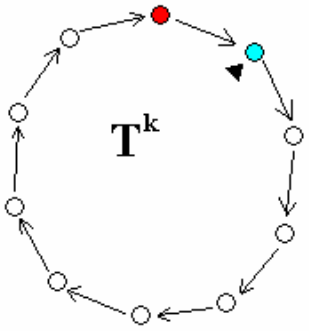


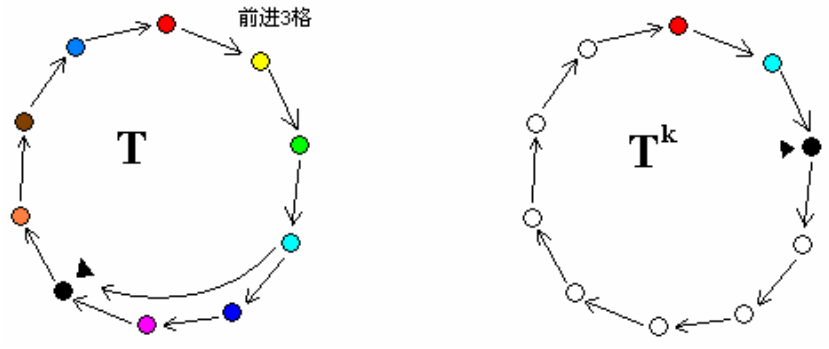
我们来一步一步构造T^k：



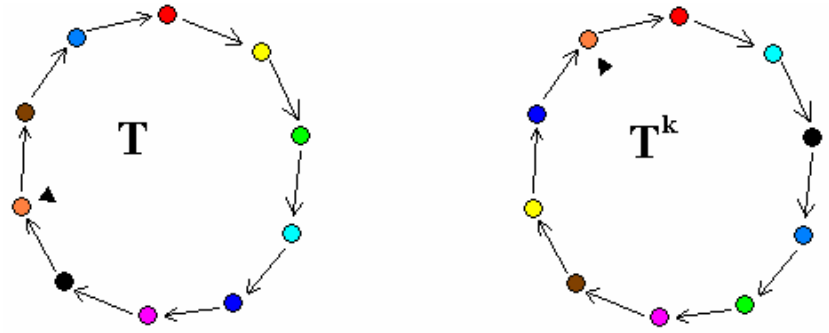








像这样反复前进三格，然后涂色...



最后，得到的循环就是所需要求的T^k了。

证明：设任意$0 \le j \le n-1$，能唯一地找到$a[t]=j$，$a'[s]=j$。

那么 j→T 显然等于$a[(t+1) \bmod n]$，$j(\rightarrow)^p T = a[(t+p) \bmod n]$（$(\rightarrow)^p T$表示连续执行 p 次→T）

由置换的连接的运算法则可知，$j(\rightarrow)^p T = j \rightarrow T^p$。所以$a'[s+1]=a[(t+k) \bmod n]$。由于循环的性质，我们令$a[0]=a'[0]=1$，就得到了上面的公式。

2.3 算法的实现

根据上一节的定理和再上一节的 3 个结论，我们可以很方便地得到求整幂运算的 O(n)算法，但是如果单纯地照着做，常数项是非常大的，有时甚至还不如 O(nlogk)的算法快。针对这一问题，可以使用一个简化的算法：

- For 源置换中每一个循环
  - For 环中每一个未标记元素
    - Do
      - 做上标记
      - 放入结果数组
      - 前进 k 格

- Until 回到这个元素
- 将结果数组中的元素取出，得到的环，便是目标置换包含的一个循环

可以分析出,这个算法是符合上一节的定理和再上一节的 3 个结论的,在这里就不再说明了。

循环的储存我们可以单纯地用 2 个数组来实现：一个是 data，把每一个循环按顺序放在里面；一个是 point，保存每个循环在 data 中的起始位置和长度。
显然，2 个数组都是 O(n)阶的。

所以，置换群的整幂运算可以用时空复杂度均为 O(n)的算法来实现。

2.4 优化的例子

庆典的日期 （国家集训队原创试题）

[问题描述]

古斯迪尔文明曾在约 10 亿年前在地球上辉煌一时，尤其在历法、数学、天文等方面的发展水平已经超过现代。在古城的众多庙宇中，考古人员都发现了一种奇特的建筑，该建筑包含一排独立的房间。

在每个房间的中央，挂有一个转盘，每个转盘分为p个格子，每个格子写着一个 1 到n的数字。转盘可以逆时针转动。转盘的红色标记始终指向上方的格子。每个房间的转盘都不相同。

CC 考古工作室近日成功地破译了当时的文字，对进一步研究古斯迪尔文明作出了重要贡献。首先，研究人员翻译了当时的宗教书籍，得知了建筑的用途。原来每个寺院都要在建成以后每隔若干年举行一次大型的庆典。由于“天机不可泄漏”，寺院方面并不直接说明庆典的日期，而是采用“暗示”的方法。奇特的建筑就是为了确定庆典的日期而专门建造的。

房间从左到右编号为 1，2，3，…n，同时寺院有n个祭司也从 1 到n编号，这些祭司每年到房间中祈祷一次。建寺那年祭司和自己编号相同的房间祈祷。同时，转盘上红色标记指示的格子的数字就是该祭司第二年祈祷的房间编号。在祭司祈祷完毕以后，将转盘逆时针旋转一格。转盘的设计使得在每年祈祷时，每个房间只有一个祭司。

从建寺以后，当某一年祈祷时，每个祭司的编号都和祈祷房间的编号相同时，就是举行庆典的日期。实际上，每隔若干年，就会有一次庆典。

作为 CC 考古工作室的首席软件顾问，你负责编程求出第一次举行庆典的确切日期。

[输入]

文件第一行是两个整数 n，p，n 表示房间的数目（也就是祭司的数目），p 表示转盘包含的格子的数目。(0 < n, p <= 200)

以下有n行，每行p个整数，表示每个房间转盘的格子上的数字。每行第一个数表示寺院建立时红色标记指向的数字，以下的数字按照顺时针方向给出。

[输出]

仅一行，表示第一次举行庆典是在建寺以后多少年。如果永远不会出现符合条件的情况或者第一次符合条件的年份超过 10^9（那时古斯迪尔文明已经衰落了），则输出’No one knows.’

[算法分析]

由于每个房间的转盘上的数字都是 p 个，而且每年每个祭司都在不同房间，所以我们可以把这些房间中安置的转盘，转化成 p 个长度为 n 的置换。而每一年祭司本身的位置，也可

以组成一个长度 n 的置换。

显然，第 i 年祭司的位置，就是第 i-1 年祭司的位置，与第 i 年转盘上数字代表的置换的连接。现在的问题是求一个最早的年份 y，使得那一年祭司的位置是单位置换。

由于转盘上的置换是以 p 为周期的，所以我们枚举 y mod p=k。那么也就是说：

$$T_1 \cdot T_2 \cdot ... T_p \cdot T_1 \cdot T_2 \cdot .... T_p \cdot .... T_{k-1} \cdot T_k = e$$

由于连接运算满足结合律，所以：

$$T_1 \cdot T_2 \cdot .... T_k \cdot (T_{k+1} \cdot T_{k+2} \cdot .... T_p \cdot T_1 \cdot T_2 \cdot .... T_k) \cdot .... \cdot (T_{k+1} \cdot T_{k+2} \cdot .... T_p \cdot T_1 \cdot T_2 \cdot .... T_k) = e$$

我们可以预先算出 $T_{Head} = T_1 \cdot T_2 \cdot .... T_k$ 和 $T_{Step} = T_{k+1} \cdot T_{k+2} \cdot .... T_p \cdot T_1 \cdot T_2 \cdot .... T_k$，那么上式就转为：

$$T_{Head} \cdot (T_{Step})^{(y-k)/p} = e$$

$$(T_{Step})^{(y-k)/p} = (T_{Head})^{-1}$$

所以，当我们枚举了 y mod p=k 以后，问题就转化成了：

已知 2 个置换 $T_{Deah} = (T_{Head})^{-1}, T_{Step}$，求一个最小的数 x=(y-k)/p 使 $T_{Step}{}^x = T_{Deah}$。

如果 T_{Step} 和 T_{Deah} 都是一个循环，那根据上面的算法立即可以得到答案的最小值x，也可以知道，所有可行的答案的形式，都是x+kn的形式。那如果有多个循环的话，显然可以列出一系列形如x mod n_i=x_i的模线性方程组。解出这个方程组，就可以得到答案了。

这个算法相对于题目作者的算法而言，更优之处在于如何得到某个方程。题目作者的算法是对于每个置换中的元素，向前枚举出每个模线性方程的x。这样最坏情况下，时间复杂度是O(n^2)，是整个算法的瓶颈。并且由于在一个循环内的元素，所得到的模线性方程是一样的，就会产生大量重复计算。而这个算法求出模线性方程组的复杂度为O(n)，而使得瓶颈转化到了求解模线性方程组的O(nlogn)上，整个算法的时间复杂度也就是O(nlogn)了。

三、置换群的分数幂运算（开方）

3.1 单循环的分数幂运算

a. 循环长度与指数互质的情况

由 2.2 节得到的定理，我们可以类似地得到长度与指数互质时的分数幂运算的过程，不过就是目标循环每次指针向后移 k 位，源循环每次向后移 1 位罢了。

b. 循环长度与指数不互质的情况

如果循环长度与指数不互质，是一定不能开方的。因为：
假设可以开方，分两种情况：
1. 目标置换是一个循环
   根据 2.1 节的结论，目标置换的 k 次幂，会把自己分裂成 gcd(l,k)份，显然不可能等于源循环。
2. 目标置换由多个循环连接而成

目标置换的k次幂，只有可能把自己包含的循环分裂，而不可能把包含的置换合并，所以目标置换的k次幂也不可能是源循环。

所以，循环长度与指数不互质时，单个循环是不能开方的。

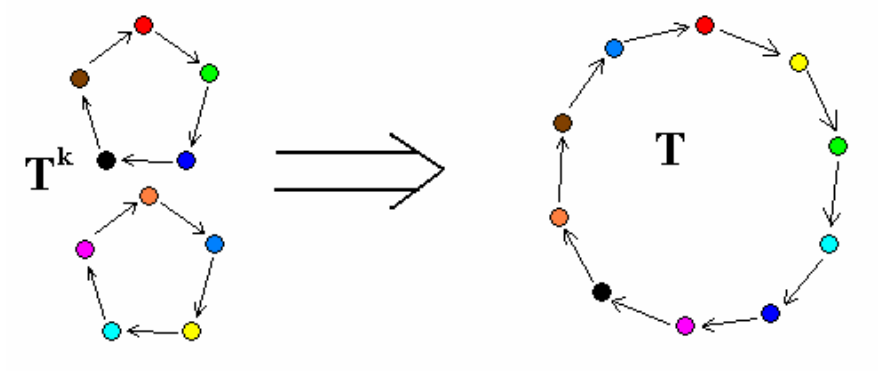
批注：在这里，第一次出现了“合并”。这将在下文中反复提及，含义均为 将一个或多个循环交错地并为一个循环。

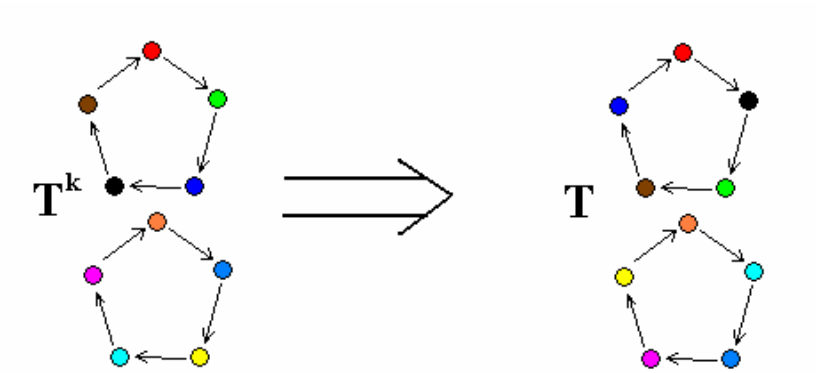
3.2 多个循环的分数幂运算

在 2.1 节的结论中，我们看到当循环长度与指数不互质时，会分裂成 gcd(l,k)份循环。那就可以联想到，如果相似地，将分出来的循环合并，就是它的逆过程了。

事实上，在开方运算中，我们可以将 k 份相同长度 l 的循环依次交错地合并，作为一次开方的过程。因为这样一个长度 k*l 的循环，作 k 次幂运算，势必会分裂成这样的 k 份每份长度 l 的循环，所以这个做法是正确的。

我们来画一张图解释一下：





对于一个由 2 个长度 5 的循环连接而成的置换 T，将它开平方就有 2 种方法：

1. 可以将两个循环交错地合并，得到一个长度 10 的大循环，正如第一张图上显示的那样；
2. 也可以将两个循环分别按照 3.1 节中的做法分别开方，得到的仍然是 2 个小循环，正如第二张图显示的那样。

可以验证，这两种方法都是正确的。

是不是一定要选择 k 个相同长度的循环合并呢？不是。

如果我们选择 m 个相同长度 l 的循环合并而作为一次运算过程的话，只要保证长度 m*l 的循环的 k 次方，会把自己分裂为 m 份，就可以保证这个做法是正确的了。

所以，对于这个m的值，我们有这样一个等式：gcd(m*l,k)=m。

可以看出，m是k的因数，并且gcd(l,k/m)=1。显然，这个式子的充分条件是m是gcd(l,k)的倍数，那么最小的m就是gcd(l,k)了。

回到 3.1 节，如果长度与指数不互质，单个循环就没有办法用 3.1 节的做法来开方。不过，我们可以选择相应 m 个长度相同的循环交错合并来完成开方的过程。可在这种情况下，如果找不到 m 个长度相同的循环，那就一定不能开方。(因为整幂运算的结果是唯一的，并且我们已经把整幂运算的所有情况都找到了，所以不存在一个置换 T'，它的 k 次方等于这种情况下的源置换 T)

这样我们可以将上面两种情况并在一起了，那就是：

1. m=gcd(l,k)
2. 选择 n*m 份长度为 l 的循环交错合并，n 为正整数，且 gcd(n*m,k/m)=1（保证第三步可行咯）
3. 将大循环开 k/m 次方

3.3 算法的实现

较整幂运算来说，分数幂运算（开方）就比较复杂了，需要分好几种情况，解也不是唯一的。在这里，我提供一个能求出一个正确解的 O(n)算法：

- 将源置换中的循环按照环的长度排序（可以使用桶排序咯）
- For 源置换中每一个环
  - m=gcd(l,k)
  - 循环判断，有 m 个相同长度的循环
    - 将 m 个循环交错地输出到目标数组，并保存为一个循环
    - 将得到的大循环仿造 2.3 节的算法开 k/m 次方并保存结果
  - Else
    - 跳出，输出无解

由于排序使用了桶排序，对每个环的操作都是 O(l)的（合并时候可以使用 m 个指针，依次下移并输出，也是 O(m*l)的），所以总的时间复杂度为 O(n)。虽然常数项的确大了点，但相比没有办法来说好多了:P

3.4 优化的例子

洗牌机 (CEOI 1998)

[问题描述]

凯凯和凡凡有 N 张牌（依次标号为 1，2，……，N）和一台洗牌机。假设 N 是奇数。洗牌机的功能是进行如下的操作：对所有位置 I（1≤I≤N），如果位置 I 上的牌是 J，而且位置 J 上的牌是 K，那么通过洗牌机后位置 I 上的牌将是 K。

凯凯首先写下一个 1~N的排列a_i，在位置a_i处放上数值a_{i+1}的牌，得到的顺序$x_1, x_2, ..., x_N$作为初始顺序。他把这种顺序排列的牌放入洗牌机洗牌S次，得到牌的顺序为$p_1, p_2, ..., p_N$。现在，凯凯把牌的最后顺序和洗牌次数告诉凡凡，要凡凡猜出牌的最初顺序$x_1, x_2, ..., x_N$。

[输入]

第一行为整数N和S。1≤N≤1000，1≤S≤1000。第二行为牌的最终顺序$p_1, p_2, ..., p_N$。

[输出]

为一行，即牌的最初顺序$x_1, x_2, ..., x_N$。

[算法分析]

很显然，这题的一副扑克牌就是一个置换，而每一次洗牌就是这个置换的平方运算。由于牌的数量是奇数，并且一开始是一个大循环，所以做平方运算时候不会分裂。所以，在任意时间，牌的顺序所表示的置换一定是一个大循环。

那么根据文章开头提到的定理：设$T^k = e$，（T 为一循环，e 为单位置换），那么 k 的最小正整数解为 T 的长度。

可以知道，这个循环的n次方是单位循环，换句话说，如果k mod n=1，那么这个循环的k次方，就是它本身。我们知道，每一次洗牌是一次简单的平方运算，洗x次就是原循环的2^x次方。

因为n是奇数，2^x mod n=1 一定有一个<n的整数解，假设这个解是a；那也就是说，一幅牌，洗a次，就会回到原来的顺序。使用最终顺序不停地洗，直到回到原始顺序，求出循环节长度a以后，再单纯地向前模拟a-s次，就可以得到原始顺序了。

上面的算法是出题方给出的标准算法。显然，时间复杂度为$O(n^2+logs)$。

换一个方向：给定了结果和 s 以后，可以简单地将这个目标置换用 3.1 节的方法开方 s 次得到结果。时间复杂度为 O(n*s)。

或者可以更简单地，算出2^s，将目标置换直接开2^s次方。这里有一个技巧，因为在开方时只需要在循环中前进2^s次，所以我们只关心(2^s) mod n，也就免去了大数字的运算。所以，计算2^s需要O(logs)，而开方需要O(n)。整个时间复杂度为O(n+logs)。

三、总结

置换群的幂运算这一问题是从最后一个例子洗牌机想到的，这一切都是对问题的深入研究带来的结果；分裂是自然而然的，而合并却是我们自己捏出来的，这一切又都是思想逆转所造成的结果；通过分裂和合并，置换群的幂运算被完美地解决了，这一切又都是多举例子多作猜想而得到的结果。

每当发现问题，探寻问题，解决问题的时候，人就会找到进步的道路。而完成这一切时，人就进步了。

[参考文献]

《算法艺术与信息学竞赛》
刘汝佳

《Graduate Texts in Mathematics》 No.163:
Permutation Groups
John D.Dixon
Brian Mortimer

[附录]

1. 《洗牌机》原题

Alice and Bob have a set of N cards labelled with numbers 1 ... N (so that no two cards have the same label) and a shuffle machine. We assume that N is an odd integer.

The shuffle machine accepts the set of cards arranged in an arbitrary order and performs the following operation of **double shuffle** : for all positions i, $1 \le i \le N$, if the card at the position i is j and the card at the position j is k, then after the completion of the operation of double shuffle, position i will hold the card k.

Alice and Bob play a game. Alice first writes down all the numbers from 1 to N in some random order: $a_1, a_2, ..., a_N$. Then she arranges the cards so that the position a_i holds the card numbered a_{i+1}, for every $1 \le i \le N-1$, while the position a_N holds the card numbered a_1.

This way, cards are put in some order $x_1, x_2, ..., x_N$, where x_i is the card at the i_{th} position.

Now she sequentially performs S double shuffles using the shuffle machine described above. After that, the cards are arranged in some final order $p_1, p_2, ..., p_N$ which Alice reveals to Bob, together with the number S. Bob's task is to guess the order $x_1, x_2, ..., x_N$ in which Alice originally put the cards just before giving them to the shuffle machine.

Input data

The first line of the input file **CARDS.IN** contains two integers separated by a single blank character : the odd integer N, $1 \le N \le 1000$, the number of cards, and the integer S, $1 \le S \le 1000$, the number of double shuffle operations.

The following N lines describe the final order of cards after all the double shuffles have been performed such that for each i, $1 \le i \le N$, the $(i+1)_{st}$ line of the input file contains p_i (the card at the position i after all double shuffles).

Output data

The output file **CARDS.OUT** should contain N lines which describe the order of cards just before they were given to the shuffle machine.

For each i, $1 \le i \le N$, the i_{th} line of the output file should contain x_i (the card at the position i before the double shuffles).