疯狂的馒头

中山纪念中学高二(19)班 陈启峰

January 25, 2007

题目

CQF十分喜欢吃馒头。兴奋之下他一下子买了N个馒头请所有认识他的人吃。

但是**CQF**不喜欢白色，而喜欢红色、黄色、绿色等鲜艳的颜色。于是他把所有白色的馒头排成一列。然后进行M次染色操作。每个染色操作都是用一个神奇的刷子把连续的多个馒头染成特定的某种颜色。一个馒头最终的颜色是最后一次染它的颜色。如果一个馒头没有被染过色，那么它的颜色就是白色。现在**CQF**已经定好了染色计划：在第i次染色操作中，把第$(i\times p+q) \bmod N+1$个馒头和第$(i\times q+p) \bmod N+1$个馒头之间的馒头染成颜色 i，其中p、q是特定的两个正整数。他想立即知道最后每个馒头的颜色。你能帮他吗？

数据范围

在**20%**的数据中，$1\leqslant N\leqslant 1000$，$1\leqslant M\leqslant 10000$

在**40%**的数据中，$1\leqslant N\leqslant 10000$，$1\leqslant M\leqslant 100000$

在**60%**的数据中，$1\leqslant N\leqslant 50000$，$1\leqslant M\leqslant 500000$

在**80%**的数据中，**1≤*N*≤300000**，**1≤*M*≤3000000**

在**100%**的数据中，**1≤*N*≤1000000**，**1≤*M*≤10000000**

保证所以输入数据中 **1≤*M*×*p* + *q*、*M*×*q* + *p*≤ $2^{31}-1$**

输入格式

第 1 行:

四个正整数***N***，***M***，***p***，***q***。

输出格式

一共输出***N***行:

第***i***行表示第***i***个馒头的最终颜色（如果最终颜色是白色就输出**0**）。

输入样例

4 3 2 4

输出样例

2

2

3

0

算法分析:

题意相当简单，就是模拟一个序列的染色操作：每次把长度为 N 的序列的$(i \times q+p) \bmod N+1$ 与$(i \times p+q) \bmod N+1$ 之间的元素都变成颜色 i。

通过 **20%**数据的算法**:**

最简易的方法就是简单地模拟每项操作。对每个染色操作用 For 循环对序列进行赋值。

时间复杂度：O(MN)

通过 **40%**数据的算法**:**

对上述的算法进行常数优化。假设要对序列 a 至 b 之间的元素都赋值为 c，在 Pascal 里我们可以使用 filldword(x[a],b－a＋1,c)来加速。实践中发现这种方法比用 For 循环的快 5 倍。

时间复杂度：O(MN)

通过 **60%**数据的算法**:**

算法 1:

注意到第 i 个操作和第 i+N 个操作对应的序列区间的一样。而由题意可知前者是无意义。因此我们只要执行最后 min(N,M)次操作。再运用上述 filldword 的方法就可以通过 60%的数据。

时间复杂度：O(min(N,M)N)

算法 2:

运用线段树加快染色操作。

时间复杂度：O(MlogN)

通过 **80%**数据的算法**:**

*运用线段树只对最后 **min(M,N)** 次染色操作进行模拟。*

时间复杂度：O(min(M,N)logN)

通过 **100%**数据的算法**:**

算法梗概：结合类似于并查集的指针进行逆向操作。

从后往前想，不难发现：最后一次的染色区间是一定出现在最终序列里；倒数第二次的染色区间则是避开最后一次的染色区间进行染色；倒数第三次的染色操作则是避开最后两次的染色区间进行染色……

*因此我们可以从后往前去只对最后 **min(N,M)** 次进行染色。我们设定后继指针 **next[i]** 表示 **i** 到 **next[i]** －1 之间都已被染色，**next[i]** 可能是未染色的。如果 **i** 到 **N** 之间都已被染色，我们初始化 **next[i]=i**，并保证 **next[i]**≥*i*。*

假设当前已经进行了最后 k-1 次操作，现在要进行倒数第 k 次操作，染色区间为[a, b]，颜色为 c。我们如果要找以 a 为左边界最左边的颜色为 0（白色）的元素，应该怎么样做呢？

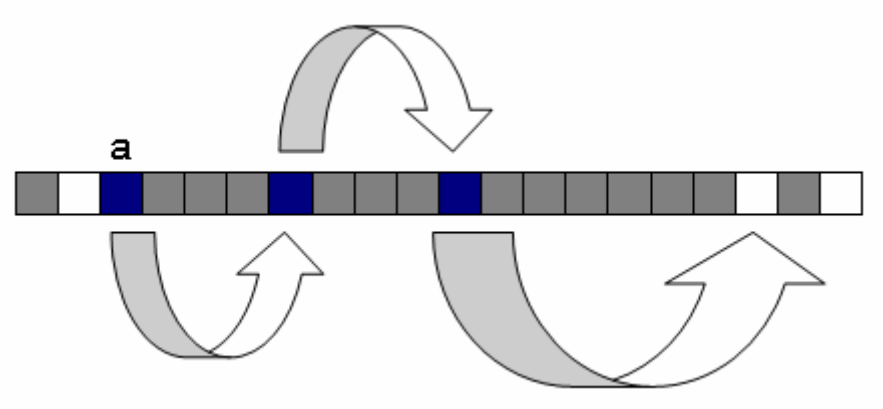

Figure 1:白格表示颜色为 0 的元素；黑格、蓝格表示已被染色的元素。

*我们只要沿着 **next** 指针不断地往后走，直到遇见一个颜色为 **0** 的元素就停止。这时的位置就是我们要找的答案。这时我们可以把蓝色部分的元素的 **next** 都指向我们找到的答案——这就相当于并查集中路径压缩。*

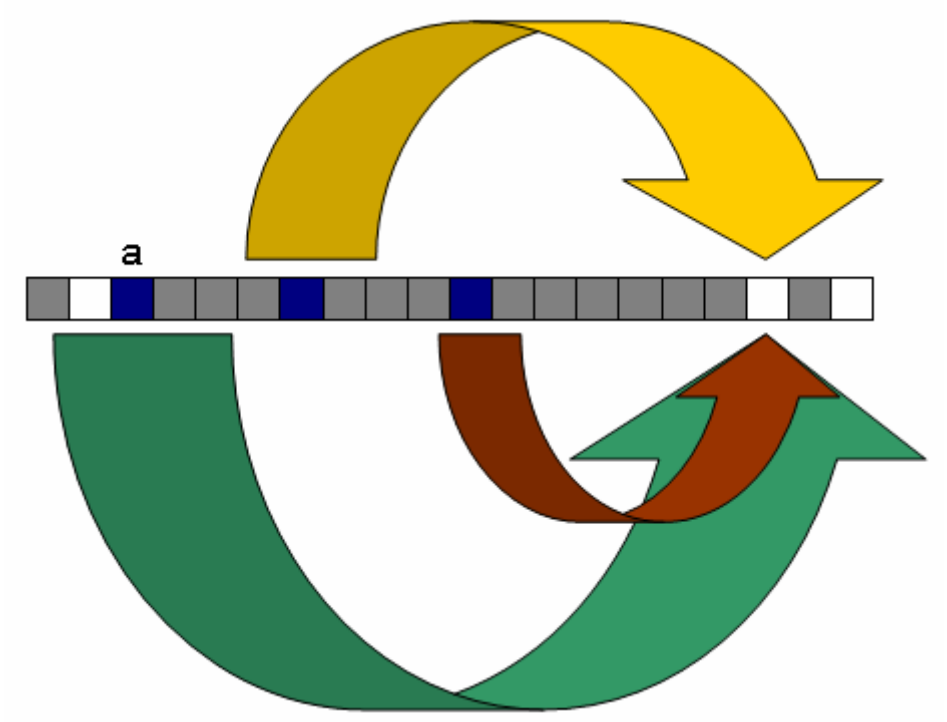

Figure 2:白格表示颜色为 0 的元素；黑格、蓝格表示已被染色的元素。

*这样，算法的流程就可以是这样的：不断地找以 **a** 为左边界最左边的颜色为 **0** 的元素，把它染成颜色 **c**，直到找到的元素位置大于 **b***

为止。

时间复杂度： $O(\min(N,M)(1+\log_{2+\frac{\min(N,M)}{N}}^{N}))=O(\min(N,M)\log N)$

程序代码:

```
{$M 100000000}
{$inline on}
program CQF_BREAD;
uses math;
var x,f:array[1..1000001] of longint;
      n,m,p,q:longint;
procedure init;
begin
    readln(n,m,p,q);
end;
procedure work;
var i,j,k:longint;
procedure find(var r:longint);inline;
begin
    if x[r]>0 then begin
        find(f[r]);
        r:=f[r];
    end;
end;
begin
    fillchar(x,sizeof(x),0);
    for i:=1 to n do
        f[i]:=i+1;
    for i:=m downto 1 do begin
        if i=m-n then
            break;
        j:=min((i*p+q)mod n+1,(i*q+p)mod n+1);
        k:=max((i*p+q)mod n+1,(i*q+p)mod n+1);
        find(j);
        while j<=k do begin
            x[j]:=i;
            find(j);
        end;
    end;
    for i:=1 to n do
        writeln(x[i]);
```

```
end;
begin
    assign(input,'bread.in');
    assign(output,'bread.out');
    reset(input);
    rewrite(output);
    init;
    work;
    close(input);
    close(output);
end.
```