1 Reverse

1 Reverse

1.1 Description

小G有一个长度为n的01串T，其中只有$T_S = 1$，其余位置都是0。现在小G可以进行若干次以下操作：

- 选择一个长度为K的连续子串（K是给定的常数），翻转这个子串。

对于每个$i, i \in [1, n]$，小G想知道最少要进行多少次操作使得$T_i = 1$. 特别的，有m个"禁止位置"，你需要保证在操作过程中1始终不在任何一个禁止位置上。

1.2 Input

从文件***reverse.in***中读入数据.
第一行四个整数n, K, m, S.
接下来一行m个整数表示禁止位置。

1.3 Output

输出到文件***reverse.out***中.
输出一行n个整数，对于第i个整数，如果可以通过若干次操作使得$T_i = 1$，输出最小操作次数，否则输出-1.

来，跟我一起念:

N方过百万，暴力碾标算！

这道题我没有用任何特殊的算法，仅仅是用了一个以SPFA（对，这道题能用SPFA）为蓝本的宽搜就直接卡常卡过去了（甚至在一测时就过了）。基本原理是从原点开始遍历原点所在位置的所有转移情况并将没计算过的点推入队列中（dis数组初始值设为-1，禁止区域设为已遍历过），依次计算直到队列为空为止。由于所有边的边权皆为正（1），这个算法其实就是名为SPFA的宽搜，很轻松就过了。

```
#pragma GCC optimize(1)
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<iostream>
#include<algorithm>
#include<queue>
#define dlong long long
#define rlong register dlong
#define IL inline
#define For(k,st,ed) for(rlong k=st;k<=ed;k++)
#define Rof(k,st,ed) for(rlong k=st;k>=ed;k--)
using namespace std;

IL dlong read(){
    rlong _1=0ll;
    register char _3=getchar();
    while(_3<'0'||_3>'9'){
        _3=getchar();
    }
```

```
    while(_3>='0'&&_3<='9'){
        _1=(_1<<1ll)+(_1<<3ll)+(_3^48ll);
        _3=getchar();
    }
    return _1;
}

IL void write(dlong _1){
    if(_1<0ll){
        putchar('-'),putchar('1');
        return;
    }
    if(_1>9ll) write(_1/10ll);
    putchar(_1%10ll+'0');
}

bool is[100005];

signed main(){
    freopen("reverse.in","r",stdin);
    freopen("reverse.out","w",stdout);
    rlong n,k,m,s,un;
    n=read(),k=read(),m=read(),s=read();
    For(phigros,1ll,m){
        un=read();
        is[un]=true;
    }
    rlong dis[n+1];
    fill(dis,dis+n+1ll,-1ll);
    register queue<dlong> spfa;
    spfa.push(s);
    dis[s]=0ll;
    is[s]=true;
    rlong ccl1,ccl2,wh;
    while(!spfa.empty()){
        wh=spfa.front();
        spfa.pop();
        ccl1=max(wh-k+1ll,1ll);
        Rof(i,min(wh,n-k+1ll),ccl1){
            if(!is[(i<<1ll)+k-wh-1ll]){
                ccl2=(i<<1ll)+k-wh-1ll;
                is[ccl2]=true;
                dis[ccl2]=dis[wh]+1ll;
                spfa.push(ccl2);
            }
        }
    }
    For(i,1ll,n){
        write(dis[i]),putchar(' ');
    }
    return 0;
}
```

2 Silhouette 学霸题

2 Silhouette

2.1 Description

有一个$n \times n$的网格，在每个格子上堆叠了一些边长为1的立方体。

现在给出这个三维几何体的正视图和左视图，求有多少种与之符合的堆叠立方体的方案。两种方案被认为是不同的，当且仅当某个格子上立方体的数量不同。

输出答案对$10^9 + 7$取模的结果。

2.2 Input

从文件***silhouette.in***中读入数据.

第一行一个整数n.

第二行n个整数，第i个表示正视图中从左到右第i个位置的高度A_i。

第三行n个整数，第i个表示左视图中从左到右第i个位置的高度B_i。

2.3 Output

输出到文件***silhouette.out***中.

输出一行表示答案。

这道题主要难在对转移方程的推导（说实话这部分真的复杂，我题解都没看太懂，只是把那个公式套上去过了而已）。首先明显可知A、B数组的顺序对答案没有任何影响，因此我们可以先把数组按从大到小顺序排序。

接下来考虑输入数据中出现的所有不同数组成的数组S，则排序后我们能得到以下图片：

那么接下来干啥已经很明显了：把每个部分的情况分别求出来再相乘就可以了。

好耶！那怎么求呢？

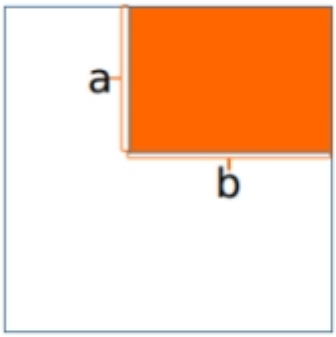
看我干嘛？我也推不出来！

还是请教大佬吧。

类似分成了一层一层的。

然后我们枚举每一层。先说S_1这一层，对于S_1有一个特殊性质，S_1是所有S中最大的，先设S_1所涉及到的这个矩形的长宽分别为a,b，

如下图：



我们在设$f[i]$表示a行中，**至少**有i行一定不合法的方案数，这里说的不合法是指高度没有到达S，也就是说不能对投影做贡献；之所以我们要设为至少，是因为这样每两行之间就可以不互相影响了。

给出状态转移方程：$f[i]=C_a^i\times(S^i\times((S+1)^{a-i}-S^{a-i}))^b$

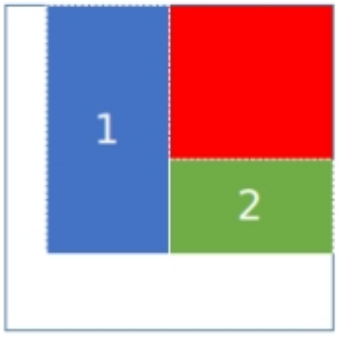
来解释一下状态转移方程：首先，组合数必不可少，因为我们要从a行里选出i行；再解释后面的b次方，因为有b列，这b列互不影响（前面有解释）；

再来解释S^i，因为有i个位置一定不合法，这i个位置可以选$0\sim S-1$这S个高度；

最后来理解$(S+1)^{a-i}-S^{a-i}$，因为还剩下$a-i$个位置，每一个位置选多高都可以，所以有$(S+1)^{a-i}$，但是我们要保证这一列一定合法，所以还要减去S^{a-i}。

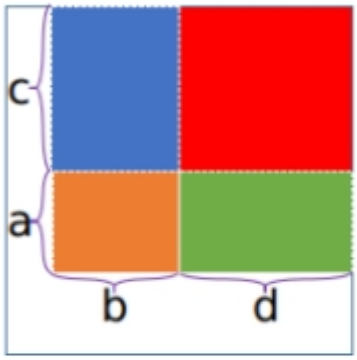
因为我们$f[i]$的定义为至少有i行不合法的方案数，所以我们还需要求得恰好有i行不合法的方案数，而我们只需要知道恰好有0行不合法的方案数（设为res），也就是都合法的方案数，考虑容斥，则有：$res=\sum_{i=0}^{a}(-1)^i\times f[i]$ 那么，我们在来考虑一般情况，也就是上图中S_2以后的所有S，因为每当我们处理完一个S之后，下一个区域的形状只有L行或矩形，如下图（红色为上一次处理的区域，紫色为这一次处理的区域）：

而无论对于哪种情况，我们都按如下划分方式将其划分为两部分：



对于矩形的两种情况，无非就是没有了那一部分，可以认为举行是一种特殊的L行。

为方便，我们不妨将L行区域按如下图方式标号a,b,c,d：



那么，现在我给出状态转移方程：

$$f[i]=C_a^i\times(S^i\times((S+1)^{a+c-i}-S^{a+c-i}))^b\times(S^i\times(S+1)^{a-i})^d$$

现在来解释这个更复杂的式子，先明确一下，因为上图中红色区域已经处理完毕，所以对于蓝色区域，其已经满足行，而没有满足列，对于绿色区域同理。

先来解释式子中的组合数C_a^i，你可能会存在疑问，为什么有$a+c$行，而不是C_{a+c}^i；那是因为上面我也明确过了，对于c行，行已经满足，我们不能让它不满足，而这i行的不满足我们只能从a行出；所以是C_a^i，而不是C_{a+c}^i。

再来解释$(S^i\times((S+1)^{a+c-i}-S^{a+c-i}))^b$，对于$b$次方和$S^i$，上面对于$S$是最大的值的情况已经做了解释，在此就不做过多的赘述；直接解释$(S+1)^{a+c-i}-S^{a+c-i}$，这个也很好理解，与上面情况类似，有i行不合法，那么还有$a+c-i$个位置可以合法，而我们要保证这一列一定合法，所以还要减去不合法的情况。

最后来看后面新填进来的这一部分，应该能看的出来，这部分是为了处理矩形$a\times d$的，也就是区域2，式子与前面大体类似，不再赘述，只来思考这样一个问题：网上有另一篇题解（在我写之前可能也只有那一篇，而我就是按照那一篇的思路调出来的），他的后面这部分是$S^i\times(S+1)^{a-i}-S^{a-i})^d$，但是我的式子里却没有$-S^{a-i}$，他的式子的漏洞就在这里，而为什么不用这部分呢？

因为我们减去S^{a-i}是为了让那一列合法，还是开始我明确的那个问题，因为我们在处理红色区域的时候已经保证了其合法，而红色区域的S要比矩形$a\times d$（绿色区域）的大，也就是高，所以无论如何其正视图都已经不变化了，变化的只有左视图，而我们需要做的就是让左视图合法，不用考虑列合不合法的问题，所以不用减去S^{a-i}。

对于统计合法方案数的容斥，其式子亦是：$res=\sum_{i=0}^{a}(-1)^i\times f[i]$ 对于$c=0$和$d=0$的情况，我们将其带入上式会发现对结果并无影响，于是我们可以将特殊情况的式子和普通情况的式子合并成一个。 这样我们从达到小不断枚举每一个S即可求出所有合法方案数。

```
#pragma GCC optimize(1)
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<iostream>
#include<vector>
#include<algorithm>
#define dlong long long
#define rlong register dlong
#define IL inline
```

```
#define For(k,st,ed) for(rlong k=(st);k<=(ed);k++)
#define rt return
using namespace std;
const dlong m=1000000007ll;

IL dlong read(){
    rlong _1=0,_2=1;
    register char _3=getchar();
    while(_3<'0'||_3>'9'){
        if(_3=='-'){
            _2=-1;
        }
        _3=getchar();
    }
    while(_3>='0'&&_3<='9'){
        _1=(_1<<1)+(_1<<3)+(_3^48);
        _3=getchar();
    }
    return _1*_2;
}

IL void write(dlong _1){
    if(_1<0) putchar('-'),_1=(-_1);
    if(_1>9) write(_1/10);
    putchar(_1%10+'0');
}

IL dlong ksm(dlong a,dlong b){
    rlong lc=a,sumn=1;
    while(b){
        if(b&1){
            sumn=(sumn*lc)%m;
        }
        lc=(lc*lc)%m;
        b>>=1;
    }
    rt sumn;
}

dlong a[100005],b[100005],n,s[200005],x[200005],y[200005],jc[100005],ans,zjz;

IL void jccsh(){
    jc[0]=1;
    For(i,1,100000){
        jc[i]=(jc[i-1]*i)%m;
    }
    rt;
}

IL dlong C(dlong fr,dlong to){
    rt (fr<to?0:(fr?(jc[fr]*ksm((jc[to]*jc[fr-to])%m,m-2))%m:1));
}

IL dlong ccl(dlong wh){
    rlong sumn=0,scnt=s[wh],acnt=x[wh]-x[wh-1],bcnt=y[wh]-y[wh-1],ccnt=x[wh-
1],dcnt=y[wh-1];
    For(i,0,acnt){
```

```
        zjz=((((C(acnt,i)*ksm(((ksm(scnt,i)*((ksm(scnt+1,acnt+ccnt-i)-
ksm(scnt,acnt+ccnt-i)+m)%m))%m),bcnt))%m)*ksm((ksm(scnt,i)*ksm(scnt+1,acnt-
i))%m,dcnt))%m;
        sumn=(i&1?(sumn-zjz+m)%m:(sumn+zjz)%m);
    }
    rt sumn;
}

signed main(){
    freopen("Silhouette.in","r",stdin);
    freopen("Silhouette.out","w",stdout);
    n=read();
    For(i,1,n){
        s[i]=a[i]=read();
    }
    For(i,1,n){
        s[n+i]=b[i]=read();
    }
    sort(a+1,a+n+1,greater<dlong>());
    sort(b+1,b+n+1,greater<dlong>());
    sort(s+1,s+(n<<1)+1,greater<dlong>());
    if(a[1]^b[1]){
        putchar('0');
        rt 0;
    }
    rlong xcnt=1,ycnt=1;
    For(i,1,n<<1){
        while(a[xcnt]>=s[i]){
            xcnt++;
        }
        while(b[ycnt]>=s[i]){
            ycnt++;
        }
        x[i]=xcnt-1;
        y[i]=ycnt-1;
    }
    jccsh();
    For(i,0,x[1]){
        zjz=(C(x[1],i)*ksm((ksm(s[1],i)*((ksm(s[1]+1,x[1]-i)-ksm(s[1],x[1]-
i)+m)%m))%m,y[1]))%m;
        ans=(i&1?(ans-zjz+m)%m:(ans+zjz)%m);
    }
    For(i,2,n<<1){
        ans=(ans*ccl(i))%m;
    }
    write(ans);
    rt 0;
}
```

3 Ancient

4 Ancient

4.1 Description

猪王国的文明源远流长，博大精深。

iPig 在大肥猪学校图书馆中查阅资料，得知远古时期猪文文字总个数为 N。当然，一种语言如果字数很多，字典也相应会很大。当时的猪王国国王考虑到如果修一本字典，规模有可能远远超过康熙字典，花费的猪力、物力将难以估量。故考虑再三没有进行这一项劳猪伤财之举。当然，猪王国的文字后来随着历史变迁逐渐进行了简化，去掉了一些不常用的字。

iPig 打算研究古时某个朝代的猪文文字。根据相关文献记载，那个朝代流传的猪文文字恰好为远古时期的 k 分之一，其中 k 是 N 的一个正约数（可以是 1 和 N）。不过具体是哪 k 分之一，以及 k 是多少，由于历史过于久远，已经无从考证了。

iPig 觉得只要符合文献，每一种能整除 N 的 k 都是有可能的。他打算考虑到所有可能的 k。显然当 k 等于某个定值时，该朝的猪文文字个数为 N/k 。然而从 N 个文字中保留下 N/k 个的情况也是相当多的。iPig 预计，如果所有可能的 k 的所有情况数加起来为 P 的话，那么他研究古代文字的代价将会是 G 的 P 次方。

现在他想知道猪王国研究古代文字的代价是多少。由于 iPig 觉得这个数字可能是天文数字，所以你只需要告诉他答案除以 999911659 的余数就可以了。

4.2 Input

从文件ancient.in中读入数据. 输入有且仅有一行：两数 N,G，用一个空格分开。

4.3 Output

输出到文件ancient.*out*中.

输出有且仅有一行：一个数，表示答案除以 999911659 的余数。

这道题简直是数论全家桶，把中国剩余定理、欧拉定理、Lucas算法等等东西一起揉了进去，没几年数论研究都不敢说做透了这道题（所以很显然我没做透）。

首先一个$O\left(\sqrt{n}\right)$的因数分解糊脸，然后再让你用欧拉定理推论（

$a^b \bmod p = a^{b \bmod \varphi(p)} \bmod p \quad (\gcd(a,p) = 1)$）求指数的模，再来个Lucas定理（$\binom{a}{b} \ mod\ p = \binom{\lfloor a/p \rfloor}{\lfloor b/p \rfloor} \cdot \binom{a \bmod p}{b \bmod p} \ mod\ p$）求组合数取模，最后还让你来个中国剩余定理（$\begin{cases} x \equiv a_1 \ (mod\ n_1) \\ x \equiv a_2 \ (mod\ n_2) \\ \vdots \\ x \equiv a_p \ (mod\ n_p) \end{cases}$ $n=\prod_{i=1}^{p} n_i$ $m_i=\frac{n}{n_i}$ $m_i \cdot m_i^{-1} \equiv 1 \ (mod\ n_i)$ $x=\sum_{i=1}^{p} a_i m_i m_i^{-1} \ \ (mod\ n)$）确定指数，快速幂什么的都只能算是饭后甜点，简直离谱到家。好在弄清楚步骤后还是很好理解的，就是步骤稍微繁琐了那么亿点点。

```
#pragma GCC optimize(1)
#pragma GCC optimize(2)
#pragma GCC optimize(3,"Ofast","inline")
#include<iostream>
#include<cmath>
#define dlong long long
#define rlong register dlong
#define reint register int
```

```
#define IL inline
using namespace std;
const dlong fj[4]={2,3,4679,35617},mod=999911659;

IL dlong read(){
    rlong _1=0,_2=1;
    register char _3=getchar();
    while(_3<'0'||_3>'9'){
        if(_3=='-'){
            _2=-1;
        }
        _3=getchar();
    }
    while(_3>='0'&&_3<='9'){
        _1=(_1<<1)+(_1<<3)+(_3^48);
        _3=getchar();
    }
    return _1*_2;
}

IL dlong ksm(dlong a,dlong b,dlong m){
    rlong lc=a,sumn=1;
    while(b){
        if(b&1){
            sumn=(sumn*lc)%m;
        }
        lc=(lc*lc)%m;
        b>>=1;
    }
    return sumn;
}

IL void write(dlong _1){
    if(_1<0) putchar('-'),_1=(-_1);
    if(_1>9) write(_1/10);
    putchar(_1%10+'0');
}

dlong n,g,jc[35700],a[4],m[4],ans,c[4];

IL void jccsh(dlong md){
    jc[0]=1;
    for(rlong i=1;i<=md;i++){
        jc[i]=(jc[i-1]*i)%md;
    }
}

IL dlong C(dlong x,dlong y,dlong md){
    return (x<y?0:(jc[x]*ksm(jc[y]*jc[x-y]%md,md-2,md))%md);
}

IL dlong Lucas(dlong x,dlong y,dlong md){
    return (x<y?0:(x?(Lucas(x/md,y/md,md)*C(x%md,y%md,md))%md:1));
}

signed main(){
    freopen("Ancient.in","r",stdin);
    freopen("Ancient.out","w",stdout);
```

```
    n=read(),g=read();
    for(rlong i=0;i<4;i++){
        jccsh(fj[i]);
        for(rlong j=1;j*j<n;j++){
            if(!(n%j)){
                a[i]=(a[i]+Lucas(n,j,fj[i])+Lucas(n,n/j,fj[i]))%fj[i];
            }
        }
        if(sqrt(n)-floor(sqrt(n))==0){
            a[i]=(a[i]+Lucas(n,(dlong)sqrt(n),fj[i]))%fj[i];
        }
        m[i]=(mod-1)/fj[i];
        c[i]=m[i]*ksm(m[i],fj[i]-2,fj[i]);
        ans=(ans+(a[i]*c[i])%(mod-1))%(mod-1);
    }
    write(ksm(g,ans,mod));
    return 0;
}
```