2022.11.15

题目	期望得分	实际得分	差值
reverse	84	43	-41
silhouette	25	5	-20
seat	8	4	-4
ancient	0	0	0

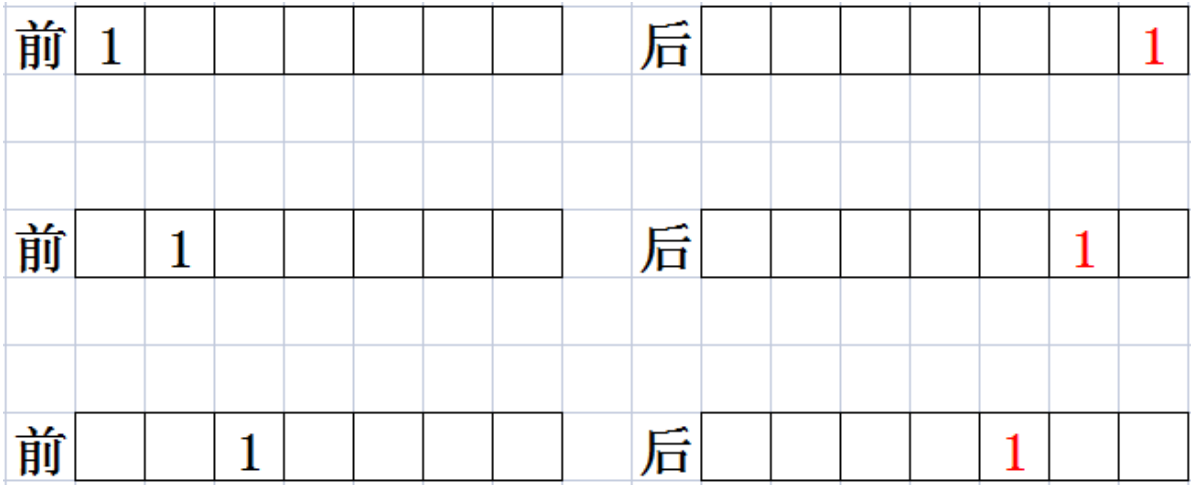
Reverse

题意：有一个01串，其中只有一个1，现要用“翻转特定长度的子串”的方式对1进行移动，求1移动到每个位置的最小步数。注意有些地方始终不能为1.

题意转化：作为1的移动方式，“翻转”是怎样的？由于我们的目的是移动1，所以翻转的子串肯定包含1，这样我们的操作才是具有建设性的。

$e.g.1$ 翻转长度为4的含1子串

$e.b.2$ 翻转长度为7的含1子串



所以我们可以将“翻转”这种抽象的变换转化为“平移”

翻转长度为 n 的含1子串

n 为奇数：可以平移的长度为：$n-1$, $n-3$,..., 2 .

n 为偶数：可以平移的长度为：$n-1$, $n-3$,..., 1 .

注意，当1的位置临近两端以至于无法进行完整的翻转时，平移规律也可能会发生变化，需要特殊考虑。

方法分析

我的第一个想法是：在所有可以到达的点之间建边，然后跑单源最短路。

但由于边权都是1，在这种情况下最短路算法的效率不如 bfs ，所以理想的算法是 bfs ，并在此基础进行优化。

直接 bfs 的效率是 $O(n^2)$ 的，我们发现可以转移到的点一定是一段区间的奇数或者偶数点，于是一种简单的优化方法是在 bfs 是开两个 set 维护当前有哪些奇数点和偶数点还未被 bfs 到，转移时直接在 set 上 $lower\ bound$ ，就不会访问已经 bfs 到过的点了。效率为 $O(nlogn)$.

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<set>
#include<queue>
using namespace std;
const int N=1e5+5;
int n,K,m,S;
bool ban[N];
int dis[N];
set<int> odd,even;
int minn(int a,int b)
{
    return a<b?a:b;
}
int maxx(int a,int b)
{
    return a>b?a:b;
}
void bfs()
{
    memset(dis,-1,sizeof(dis));
    queue<int> q;
    dis[S]=0;
    q.push(S);
    while(q.size())
    {
        int x=q.front();q.pop();
        int l=maxx(1,x-K+1);
        int r=minn(n,x+K-1);
        l=l+(l+K-1)-x;
        r=r+(r-K+1)-x;
        set<int> &p=l&1?odd:even;
        for(auto i=p.lower_bound(l);i!=p.end()&& *i<=r;p.erase(i++))
        {
            dis[*i]=dis[x]+1;
            q.push(*i);
        }
    }
}
int main()
```

```
{
    scanf("%d%d%d%d",&n,&K,&m,&S);
    for(int i=1,x;i<=m;i++)
    {
        scanf("%d",&x);
        ban[x]=1;
    }
    for(int i=1;i<=n;i++)
    {
        if(!ban[i]&&i!=S)
        {
            if(i&1) odd.insert(i);
            else even.insert(i);
        }
    }
    bfs();
    for(int i=1;i<=n;i++)
    {
        printf("%d ",dis[i]);
    }
    puts("");
    return 0;
}
```

set 中迭代器的使用比较抽象，在此一并复习。

```
for(auto i=p.lower_bound(l);i!=p.end()&& *i<=r;p.erase(i++))
{
    dis[*i]=dis[x]+1;
    q.push(*i);
}
```

因为 $p.lower\ bound(l)$ 的返回值是一个迭代器，所以 $auto\ i$ 也是一个迭代器。迭代器只支持"++"，"--"两个与算术相关的操作。在遍历时，不能用 $i<p.end()$ ，只能用 $i!=p.end()$.$*i$ 表示对 i 暂时解除引用，这时 i 就可以当成普通的数值型使用了。

Ancient

题意转化：给定整数 q , n ,计算 $q^{\sum_{d|n} C_n^d} mod\ 999911659$

若 $q=999911659$ 则上式结果为0。否则，因为 999911659 是质数，所以 q , n 互质。

引理1：欧拉定理推论

若正整数 a , n 互质，则对于任意正整数 b ，有 $a^b \equiv a^{b\ mod\ \phi(n)} mod\ n.$

引理2：欧拉函数

1~N中与 N 互质的数的个数被称为欧拉函数，记为$\phi(N)$.

*推论：当 N 为质数时，$\phi(N)=N-1.$

引理3：中国剩余定理

设 $m_1,m_2,\ldots,m_n$ 是两两互质的整数，$m=\Pi_{i=1}^n m_i$,$M_i=m/m_i$, t_i 是线性同余方程 $M_i t_i \equiv 1(mod\ m_i)$ 的一个解。对于任意的 n 个整数 $a_1,a_2,\ldots,a_n$ ，方程组

$$\begin{cases} x \equiv a_1 (mod\ m_1) \\ x \equiv a_2 (mod\ m_2) \\ \quad\quad ..... \\ x \equiv a_n (mod\ m_n) \end{cases}$$

有整数解，解为 $x = \sum_{i=1}^{n} a_i M_i t_i$.

引理4：费马小定理

若 p 是质数，则对于任意整数 a ，有 a^{p-2} 为 a 在 $mod\ p$ 意义下的逆元。

*由引理4可得$M_i t_i \equiv 1 (mod\ m_i)$ 中 t_i 解的情况。

引理5：$lucas$ 定理

若 p 是质数，则对于任意整数 $1 \leq m \leq n$,有：$C_n^m \equiv C_{n\ mod\ p}^{m\ mod\ p} \times C_{n/p}^{m/p} (mod\ p)$.

*也就是把 n 和 m 表示成 p 进制数，对 p 进制下每一位分别计算组合数，最后再乘起来。

引理6：算术基本定理

任何一个大于 1 的正整数都能唯一分解为有限个质数的乘积，可写作$N = p_1^{c_1} p_2^{c_2} \dots p_m^{c_m}$

其中 c_i 都是正整数，p_i 都是质数，且满足 $p_1 < p_2 < \dots < p_m$.

solution

由引理1、引理2可得：

$q^{\sum_{d|n} C_n^d} \equiv q^{\sum_{d|n} C_n^d\ mod\ 999911658} mod\ 999911659$.

因此，本题的关键是计算 $\sum_{d|n} C_n^d mod 999911658$.

由引理6，尝试分解质因数，可以发现 $999911658 = 2 \times 3 \times 4679 \times 35617$.

现考虑：

$m = m_1 \times m_2 \times m_3 \times m_4$.

$A = \sum_{d|n} C_n^d$

欲求：$A\%m$

先算：

$A\%m_1 = a_1$

$A\%m_2 = a_2$

$A\%m_3 = a_3$

$A\%m_4 = a_4$

由引理5，这个过程可以用 $lucas$ 定理进行优化。在计算过程中，对于一个质数 p ，预处理 p 以内的所有阶乘以及阶乘的模 p 乘法逆元，就能快速计算按照 p 进制表示之后，每一位对应的组合数。

再求解方程组：

$$\begin{cases} x\%m_1 = a1 \\ x\%m_2 = a2 \\ x\%m_3 = a3 \\ x\%m_4 = a4 \end{cases}$$

由引理3，x 的最小非负整数解可以由 x 对 m 取模，并让 x 落在 0~$m-1$ 的范围内即可。

则 $A\%m=x$.

回归本题，可得到 $\sum_{d|n} C_n^d mod 999911658$ 的最小非负整数解 x.再用快速幂求 q^x 即可得到原问题的答案。

```
#include<bits/stdc++.h>
#define ll long long
#define mod 999911658
#define N 40005
using namespace std;
ll n,g,m[5],a[5],b[5]={0ll,2ll,3ll,4679ll,35617ll},ans=0ll;
ll jie[40005];

void init(ll x)//预处理p以内的所有阶乘
{
    jie[0]=jie[1]=1ll;
    for(ll i=2;i<=x;i++)
    {
        jie[i]=(jie[i-1]*i)%x;//jie[]数组记录了预处理出的阶乘
    }
    return;
}

//快速幂
ll ksm(ll x,ll y,ll p)
{
    ll res=1ll;x%=p;
    for(;y;y>>=1)
    {
        if(y&1) res=(res*x)%p;
        x=(x*x)%p;
    }
    return res;
}

//lucas定理优化求组合数的过程
ll C(ll x,ll y,ll p)
{
    if(x<y) return 0;
    ll up=jie[x],down=jie[y]*jie[x-y]%p;
    return up*ksm(down,p-2,p)%p;
}
ll lucas(ll x,ll y,ll p)
{
    if(x<y) return 0ll;
    if(!x) return 1ll;
    return lucas(x/p,y/p,p)*C(x%p,y%p,p)%p;
}

//中国剩余定理计算出最小非负整数解x
void crt()
{
    for(int i=1;i<=4;i++)
    {
```

```
        m[i]=mod/b[i];
        ans+=(a[i]*m[i]%mod)*ksm(m[i],b[i]-2,b[i])%mod;
    }
    return;
}
int main()
{
    scanf("%lld%lld",&n,&g);
    if(!(g%(mod+1))) {printf("0");return 0;}//特判 q=999911659
    for(int i=1;i<=4;i++)
    {
        init(b[i]);
        for(ll j=1;j*j<=n;j++)//枚举n的约数
        {
            if(!(n%j))
            {
                a[i]=(a[i]+lucas(n,j,b[i]))%b[i];
                if(j*j!=n) a[i]=(a[i]+lucas(n,n/j,b[i]))%b[i];
            }
        }
    }
    crt();//使用中国剩余定理
    printf("%lld",ksm(g,ans,mod+1));
    return 0;
}
```

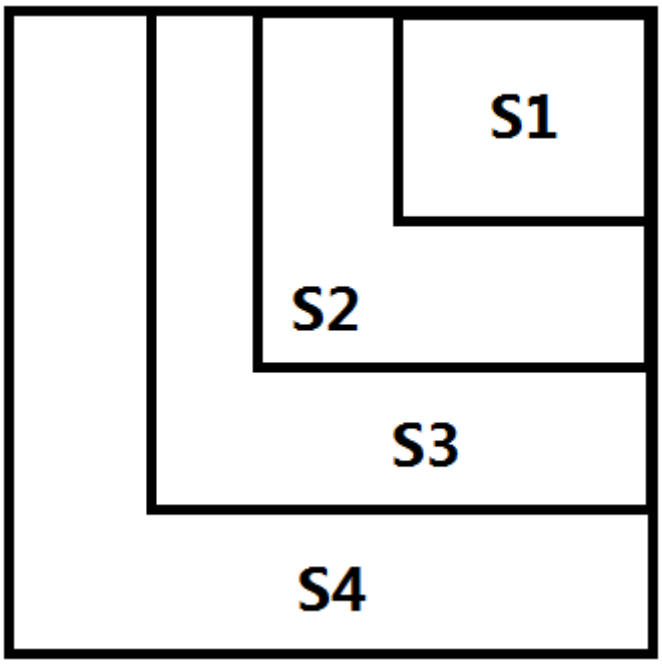
silhouette

题意理解

使第 i 列的最大数小于 $B[i]$,使第 j 行的最大数小于 $A[i]$.求满足这样条件的方案数。

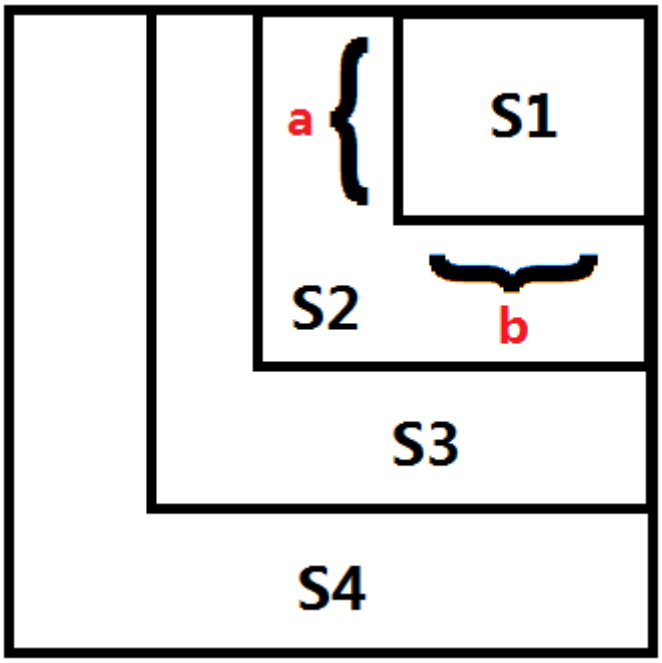
预处理

将 A, B 两个数组进行排序，可以证明这对结果是没有影响的。将 A, B中的每个值放进一个新的数组 S（那么 S 的大小就是 $n \times 2$ ）也对 S 进行排序。S 排好序后呈现这个状态。



可以直观地看到分层的形式。

然后枚举每一层。其中 S_1 这一层有一个特殊性质，S_1 是所有 S 中最大的，先设 S_1 这个矩形的长宽分别为a, b.



状态转移

设 $f[i]$ 为至少有 i 行的限制不满足条件时的方案（需要保证每一列都满足条件），那么有转移方程：
$f[i] = C_i^a \times (s^i \times ((s+1)^{a-i} - s^{a-i}))^b$

方案数就是 $\sum_{i=0}^{a} \times (-1)^i \times f[i]$

容斥

感性理解，本题的容斥是一个这样的过程：

从一个集合中挖掉一块，而那块中有一部分是我们需要的;然而我们需要的那一部分中又有一块是应该被挖去的。以此类推。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
typedef long long ll;
const ll mod=1e9+7;
const ll N=1e5+5;
ll n,ans=1;
ll A[N],B[N],S[N<<1];
ll jie[N],inv[N];
ll ksm(ll a,ll b)
{
    ll res=1;
    for(;b;b>>=1)
    {
        if(b&1) res=1ll*res*a%mod;
        a=1ll*a*a%mod;
    }
    return res;
}
void pre()
{
    jie[0]=1;
    for(ll i=1;i<=N-5;i++) jie[i]=1ll*jie[i-1]*i%mod;
    inv[N-5]=ksm(jie[N-5],mod-2);
    for(ll i=N-5;i>=1;i--) inv[i-1]=1ll*inv[i]*i%mod;
}
ll C(ll x,ll y)
{
    return jie[x]*inv[y]%mod*inv[x-y]%mod;
}
ll lucas(ll x,ll y)
{
    if(!y) return 1;
    return C(x%mod,y%mod)*lucas(x/mod,y/mod)%mod;
}
ll solve(ll a,ll b,ll c,ll d,ll s)
{
    ll res=0;
    for(ll i=0;i<=a;i++)
    {
        ll now=lucas(a,i)*ksm(ksm(s,i)*((ksm(s+1,a+c-i)-ksm(s,a+c-
i)+mod)%mod)%mod,b)%mod*ksm(ksm(s,i)*ksm(s+1,a-i)%mod,d)%mod;
        if(i&1) res=(res-now+mod)%mod;
        else res=(res+now)%mod;
    }
    return res;
```

```
}
signed main()
{
    freopen("silhouette.in","r",stdin);
    freopen("silhouette.out","w",stdout);
    pre();
    scanf("%lld",&n);
    for(ll i=1;i<=n;i++)
    {
        scanf("%lld",&A[i]);
        S[i]=A[i];
    }
    for(ll i=1;i<=n;i++)
    {
        scanf("%lld",&B[i]);
        S[n+i]=B[i];
    }
    sort(A+1,A+n+1);
    sort(B+1,B+n+1);
    if(A[n]!=B[n])
    {
        puts("0");
        return 0;
    }
    sort(S+1,S+2*n+1);
    S[0]=unique(S+1,S+2*n+1)-S-1;
    ll prea=n+1,preb=n+1,nowa=n,nowb=n;
    for(ll i=S[0];i;i--)
    {
        while(A[nowa-1]==S[i]&&nowa-1) nowa--;
        while(B[nowb-1]==S[i]&&nowb-1) nowb--;
        ans=ans*solve(prea-nowa,preb-nowb,n-prea+1,n-preb+1,S[i])%mod;
        prea=nowa;
        preb=nowb;
    }
    printf("%lld\n",ans);
    return 0;
}
```