比赛链接

补题情况

题目	蒋贤蒙	王赵安	王智彪
A	2	2	0
B	0	0	0
C	2	2	0
D	2	1	0
E	0	0	0
G	2	1	0
I	0	0	0
J	2	0	2
K	2	0	0

题解

A. Browser Games

题意

给定 n 个字符串，对 $i=1\sim n$找出一个最小的前缀集合，满足：

对前 i 个字符串都至少有一个前缀位于该集合且对于后面的 $n-i$ 个字符串都不存在前缀属于这个集合。

数据保证不存在一个字符串是另一个字符串前缀的情况，且内存限制为 32 megabytes

题解

首先不考虑内存限制，可以对所有串建立字典树，并用叶子结点代表每个字符串。

然后问题转化为从树上选择最少的结点集合，使得前 i 个字符串至少有一个祖先结点被选中，且后 $n-i$ 个字符串不存在祖先结点被选中。

不难发现，如果一个结点的子树中叶子结点都属于前 i 个字符串，则以该结点为根的子树答案为 1。否则该结点答案等于所有儿子结点答案之和。

建立字典树后依次处理 $i=1\sim n$ 的询问，动态更新每个结点的子树中的后 $n-i$ 个字符串个数以及以该结点为根的子树答案。

每次询问的答案记为字典树根节点的答案，注意特判 $i=n$ 的询问，因为前缀不能是空串。

然后考虑内存限制，注意到只有一个儿子结点的结点都是可以压缩的，于是树上的关键结点个数可以卡到 $O(n)$

最坏的情况是完全二叉树，这时节点个数是 $2n-1$于是开两倍空间即可。

关于建树，可以递归构建，如果当前结点的字符串个数为 1 则直接返回。

否则找到最小的当前结点的所有字符串的非公共前缀长度，然后划分字符串，继续递归，同时记录非公共前缀的位置用于后续更新操作比较。

```
const int MAXN=1e5+5,MAXS=MAXN<<1,MAXL=105;
char s[MAXN][MAXL],suf[MAXS];
int head[MAXS],nxt[MAXS],dep[MAXS],s1[MAXS],s2[MAXS],node_cnt;
vector<int> c[MAXS];
void build(int k,int d){
    s1[k]=c[k].size();
    if(s1[k]==1){
        c[k].clear();
        return;
    }
    while(true){
        int p1=c[k][0];
        bool flag=true;
        for(int p2:c[k]){
            if(s[p2][d]!=s[p1][d]){
                flag=false;
                break;
            }
        }
        if(flag)
        d++;
        else
        break;
    }
    dep[k]=d;
    for(int t:c[k]){
        int i=head[k];
        for(;i;i=nxt[i]){
            if(suf[i]==s[t][d])
            break;
        }
        if(!i){
            i=++node_cnt;
            nxt[i]=head[k];
            suf[i]=s[t][d];
            head[k]=i;
        }
        c[i].push_back(t);
    }
    c[k].clear();
    for(int i=head[k];i;i=nxt[i])
    build(i,d+1);
}
void update(int k,char *t){
    s1[k]--;
```

```
    if(s1[k]==0){
        s2[k]=1;
        return;
    }
    for(int i=head[k];i;i=nxt[i]){
        if(suf[i]==t[dep[k]]){
            s2[k]-=s2[i];
            update(i,t);
            s2[k]+=s2[i];
            break;
        }
    }
}
int main()
{
    int n=read_int();
    if(n==1){
        puts("1");
        return 0;
    }
    _rep(i,1,n){
        scanf("%s",s[i]);
        c[0].push_back(i);
    }
    build(0,0);
    _for(i,1,n){
        update(0,s[i]);
        enter(s2[0]);
    }
    int ans=0;
    for(int i=head[0];i;i=nxt[i])
    ans++;
    enter(dep[0]==0?ans:1);
    return 0;
}
```

C. Dance Party

题意

给定 $n\times 2$ 的二分图。对左部每个点，仅和右部 k_i 个点不连边。求二分图最大匹配。

题解

设 $k=\max_{i=1}^n k_i$

先进行预匹配，每个左部点任选一个还未被匹配且有连边的右部点匹配，可以用 set 维护所有未

匹配的右部点，时间复杂度 $O(nk\log n)$

接下来剩下的未匹配的点一定不超过 k 个，对每个点考虑匈牙利算法匹配， 总时间复杂度为 $O(km)$

$O(m)\sim O(n^2)$考虑优化。假定现在需要对点 i 进行匈牙利算法，将右部与点 i 不相邻的点染黑，其余右部点染白。

对除点 i 以外的左部点，仅保留与黑点相关的连边，这样 $O(m)\sim O(nk)$总时间复杂度 $O\left(nk^2\right)$ 足以通过此题。

关于算法的正确性，假设在原图上存在一条从 i 出发的增广路，且增广路上除了 i 以外有其他点的失配边指向白点。

找到增广路上的最后一个白点，直接将 i 的失配边指向该点然后保留原增广路的剩余部分也可以一条增广路。

同时该增广路上除了 i 其他点的失配边都指向黑点。因此只要原图存在一条从 i 出发的增广路则只保留与黑点相关的连边也可以得到一条增广路。

```
const int MAXN=3e4+5,MAXK=105;
struct Edge{
    int to,next;
}edge[MAXN*MAXK];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
bitset<MAXN> bt[MAXN];
vector<int> g[MAXN];
namespace KM{
    set<int> s;
    int match[MAXN],vis[MAXN];
    bool dfs(int u,int k){
        if(vis[u]==k)
        return false;
        vis[u]=k;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(!match[v]||dfs(match[v],k))
            return match[v]=u,true;
        }
        return false;
    }
    bool get_pair(int n){
        _rep(u,1,n)
        s.insert(u);
        vector<int> vec;
        _rep(u,1,n){
            bool flag=true;
            for(int v:s){
```

```
                if(!bt[u][v]){
                    match[v]=u;
                    s.erase(v);
                    flag=false;
                    break;
                }
            }
            if(flag)
            vec.push_back(u);
        }
        for(int i:vec){
            mem(head,0);
            edge_cnt=0;
            _rep(u,1,n){
                for(int v:g[i]){
                    if(!bt[u][v])
                    Insert(u,v);
                }
            }
            _rep(v,1,n){
                if(!bt[i][v])
                Insert(i,v);
            }
            if(!dfs(i,i))
            return false;
        }
        return true;
    }
}
int ans[MAXN];
int main()
{
    int n=read_int();
    _rep(u,1,n){
        int k=read_int();
        while(k--){
            int v=read_int();
            g[u].push_back(v);
            bt[u][v]=1;
        }
    }
    if(KM::get_pair(n)){
        _rep(i,1,n)
        ans[KM::match[i]]=i;
        _rep(i,1,n)
        space(ans[i]);
    }
    else
    puts("-1");
    return 0;
```

```
}
```

D. Diameter Counting

题意

求所有 n 标号树的直径和。

题解1

考虑求树的直径的过程，可以先删去所有叶子结点，得到一棵新树，称为一次操作。然后再不断对新树进行操作，知道最后剩下一个或两个结点。

此时如果只剩下一个结点，则树的直径为操作次数 $\times 2$，如果剩下两个结点，则树的直径为操作次数 $\times 2+1$。

考虑通过逆算法反向构建树。设 $f(i,j)$ 表示有 j 个叶子结点的 i 标号树个数，假设上一步操作删除了 $k(k\ge j)$ 个叶子。

于是问题等价于给这 k 个叶子找一个父结点，使得原来的 j 个叶子结点至少有一个儿子。

同时对于这 $i+k$ 个结点，标号是任意的，对所有 $i+k$ 标号树而言，删去 k 叶子结点得到的树的标号方式实际上有 ${i+k\choose i}f(i,j)$ 种。

设 $g(i,j,k)$ 表示长度为 k 且每个位置有 i 种可选取值且特定的 j 个值至少出现一次的序列个数，于是有

$$ f(i+k,k)\gets g(i,j,k)f(i,j){i+k\choose i} $$

接下来考虑求 $g(i,j,k)$，可以考虑序列前 $k-1$ 位，如果此时 j 个特定值都出现了至少一次，则第 k 位可以任取，于是有

$$ g(i,j,k)\gets i\times g(i,j,k-1) $$

如果前 $k-1$ 位只有 $j-1$ 个特殊值出现了至少一次，则显然前 $k-1$ 位的取值只有 $i-1$ 种，同时要从 j 个特殊值中确定一个放在第 k 位，有

$$ g(i,j,k)\gets j\times g(i-1,j-1,k-1) $$

最后设 $h(i,j)$ 表示有 j 个叶子的 i 标号树的直径之和。同样假设上一步操作删除了 $k(k\ge j)$ 个叶子。

考虑原有的树的直径和操作带来的直径 $+2$ 的新贡献，于是有

$$ h(i+k,k)\gets g(i,j,k){i+k\choose i}(h(i,j)+2f(i,j)) $$

另外所有 $h(i+k,k)\gets 2f(i,j)g(i,j,k){i+k\choose i}$ 也可以等价于 $h(i,j)\gets 2f(i,j)$，时空间复杂度 $O\left(n^3\right)$。

```
const int MAXN=505;
int mod;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
int frac[MAXN],invf[MAXN];
int C(int n,int m){
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int f[MAXN][MAXN],g[MAXN][MAXN][MAXN],h[MAXN][MAXN];
int main()
{
    int n=read_int();
    mod=read_int();
    frac[0]=1;
    _for(i,1,MAXN)frac[i]=1LL*frac[i-1]*i%mod;
    invf[MAXN-1]=quick_pow(frac[MAXN-1],mod-2);
    for(int i=MAXN-1;i;i--)
    invf[i-1]=1LL*invf[i]*i%mod;
    g[0][0][0]=1;
    _rep(i,1,n){
        g[i][0][0]=1;
        _rep(k,1,n)
        g[i][0][k]=1LL*g[i][0][k-1]*i%mod;
        _rep(j,1,i)_rep(k,j,n)
        g[i][j][k]=(1LL*g[i][j][k-1]*i+1LL*g[i-1][j-1][k-1]*j)%mod;
    }
    f[1][1]=f[2][2]=1;
    _rep(i,1,n)_rep(j,1,i)_rep(k,max(j,2),n-i)
    f[i+k][k]=(f[i+k][k]+1LL*f[i][j]*g[i][j][k]%mod*C(i+k,i))%mod;
    h[2][2]=1;
    _rep(i,3,n)_rep(j,1,i)
    h[i][j]=2LL*f[i][j]%mod;
    _rep(i,1,n)_rep(j,1,i)_rep(k,max(j,2),n-i)
    h[i+k][k]=(h[i+k][k]+1LL*h[i][j]*g[i][j][k]%mod*C(i+k,i))%mod;
    int ans=0;
    _rep(i,1,n)
    ans=(ans+h[n][i])%mod;
    enter(ans);
    return 0;
}
```

题解2

设 $f(i,j)$ 表示深度为 j 的 i 标号树个数，$g(i,j)$ 表示深度不超过 j 的 i 标号树个数。

对一个直径为 $2d+1$ 的 n 标号无根树，可以沿着直径的中心边切开，得到两个深度为 d 的有根树。

设 1 号点所在的有根树大小为 i，考虑为他分配 $i-1$ 个编号，于是直径为 $2d+1$ 的 n 标号无根树个数为

$$ \sum_{i=1}^{n-1}f(i,d)f(n-i,d){n-1\choose i-1} $$

对一个直径为 $2d$ 的 n 标号无根树，可以认为是根节点连接至少两个深度等于 $d-1$ 的有根树。

利用容斥，用深度为 d 的 n 标号有根树个数减去根节点仅连接一个深度为 $d-1$ 的有根树个数的情况。

根节点仅连接一个深度为 $d-1$ 的有根树可以认为是由一个深度不超过 $d-1$ 的有根树根节点连接一个深度等于 $d-1$ 的有根树得到的。

于是直径为 $2d$ 的 n 标号无根树个数为

$$ f(n,d)-\sum_{i=1}^{n-1}f(i,d-1)g(n-i,d-1){n\choose i} $$

接下来考虑计算 $f(i,j),g(i,j)$，$f(i,j)$ 难以直接计算，但显然有 $f(i,j)=g(i,j)-g(i,j-1)$，于是只需要计算 $g(i,j)$。

对于深度不超过 j 的 i 标号树个数，可以先分配一个编号给根结点，然后从与根节点相连的子树中找到编号最小的结点所在的子树。

设子树大小为 k，对该子树，他深度不超过 $j-1$，显然有 $g(k,j-1)$ 种。另外需要从剩余 $i-2$ 个编号再分配 $k-1$ 个编号给子树。

对于余下的 $n-k$ 个点，深度仍然不超过 j，但根节点编号已经分配，所以总数为 $\frac {g(i-k,j)}{i-k}$，于是有

$$ g(i,j)=\sum_{k=1}^{i-1}i{i-2\choose k-1}g(k,j-1)\frac {g(i-k,j)}{i-k} $$

时间复杂度 $O\left(n^3\right)$，空间复杂度 $O\left(n^2\right)$。

```
const int MAXN=505;
int mod;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
int frac[MAXN],invf[MAXN],inv[MAXN];
int C(int n,int m){
```

```
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int f[MAXN][MAXN],g[MAXN][MAXN];
int main()
{
    int n=read_int();
    mod=read_int();
    frac[0]=1;
    _for(i,1,MAXN)frac[i]=1LL*frac[i-1]*i%mod;
    invf[MAXN-1]=quick_pow(frac[MAXN-1],mod-2);
    for(int i=MAXN-1;i;i--){
        invf[i-1]=1LL*invf[i]*i%mod;
        inv[i]=1LL*invf[i]*frac[i-1]%mod;
    }
    g[1][0]=1;
    _rep(i,1,n){
        _for(j,1,i)_for(k,1,i)
        g[i][j]=(g[i][j]+1LL*g[k][j-1]*g[i-
k][j]%mod*C(i-2,k-1)%mod*i%mod*inv[i-k])%mod;
        _rep(j,i,n)
        g[i][j]=g[i][i-1];
    }
    f[1][0]=1;
    _rep(i,2,n)_for(j,1,i)
    f[i][j]=(g[i][j]-g[i][j-1])%mod;
    int ans=0;
    _for(i,1,n){
        int cnt=0,d=i/2;
        if(i&1){
            _for(j,1,n)
            cnt=(cnt+1LL*f[j][d]*f[n-j][d]%mod*C(n-1,j-1))%mod;
        }
        else{
            cnt=f[n][d];
            _for(j,1,n)
            cnt=(cnt-1LL*f[j][d-1]*g[n-j][d-1]%mod*C(n,j))%mod;
        }
        ans=(ans+1LL*cnt*i)%mod;
    }
    if(ans<0)ans+=mod;
    enter(ans);
    return 0;
}
```

G. Game of Death

题意

一个游戏，有 n 名玩家。每个玩家等概率选择一名除自己以为的人进行射击，射击的命中率为 p

对每个 $k=1\sim n$ 询问最后存活 k 人的概率。

题解

设 $f(k)$ 表示固定 k 个人的集合，这 k 个人全部死亡的概率$g(k)$ 表示固定 k 个人的集合，死亡的人是这个集合的子集的概率。

首先考虑计算 $g(k)$事实上可以把所有人分成两种人，一种是在这个 k 人集合中的人，记为 A 类人。另一种记为 B 类人。

于是只需要保证不杀死 B 类人即可。于是对 A 类人，这个概率为 $1-\frac {p(n-k)}{n-1}$而对于 B 类人，这个概率为 $1-\frac {p(n-1-k)}{n-1}$

于是有

$$ g(k)=\left(1-\frac {p(n-k)}{n-1}\right)^k\left(1-\frac {p(n-1-k)}{n-1}\right)^{n-k} $$

同时有 $g(k)=\sum_{i=0}^k {k\choose i}f(i)$根据二项式反演，得 $$ f(k)=\sum_{i=0}^k (-1)^{k-i}{k\choose i}g(i)=k!\sum_{i=0}^k\frac{(-1)^{k-i}}{(k-i)!}\frac{g(i)}{i!} $$ 卷积计算即可，最终 k 人死亡的答案为 ${n\choose k}f(k)$时间复杂度 $O(n\log n)$

```
const int MAXN=3e5+5,mod=998244353;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
namespace Poly{
    const int G=3,Mod=998244353;
    int rev[MAXN<<2],Wn[30][2];
    void init(){
        int m=Mod-1,lg2=0;
        while(m%2==0)m>>=1,lg2++;
        Wn[lg2][1]=quick_pow(G,m);
        Wn[lg2][0]=quick_pow(Wn[lg2][1],Mod-2);
        while(lg2){
            m<<=1,lg2--;
            Wn[lg2][0]=1LL*Wn[lg2+1][0]*Wn[lg2+1][0]%Mod;
            Wn[lg2][1]=1LL*Wn[lg2+1][1]*Wn[lg2+1][1]%Mod;
        }
    }
    int build(int k){
        int n,pos=0;
        while((1<<pos)<=k)pos++;
        n=1<<pos;
```

```
        _for(i,0,n)rev[i]=(rev[i>>1]>>1)|((i&1)<<(pos-1));
        return n;
    }
    void NTT(int *f,int n,bool type){
        _for(i,0,n)if(i<rev[i])
        swap(f[i],f[rev[i]]);
        int t1,t2;
        for(int i=1,lg2=0;i<n;i<<=1,lg2++){
            int w=Wn[lg2+1][type];
            for(int j=0;j<n;j+=(i<<1)){
                int cur=1;
                _for(k,j,j+i){
                    t1=f[k],t2=1LL*cur*f[k+i]%Mod;
                    f[k]=(t1+t2)%Mod,f[k+i]=(t1-t2)%Mod;
                    cur=1LL*cur*w%Mod;
                }
            }
        }
        if(!type){
            int div=quick_pow(n,Mod-2);
            _for(i,0,n)f[i]=(1LL*f[i]*div%Mod+Mod)%Mod;
        }
    }
    void mul(int *f,int _n,int *g,int _m){
        int n=build(_n+_m-2);
        _for(i,_n,n)f[i]=0;_for(i,_m,n)g[i]=0;
        NTT(f,n,1);NTT(g,n,1);
        _for(i,0,n)f[i]=1LL*f[i]*g[i]%Mod;
        NTT(f,n,0);
    }
}
int frac[MAXN],invf[MAXN];
int C(int n,int m){
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int f[MAXN<<2],g[MAXN<<2];
int main()
{
    Poly::init();
    int n=read_int(),a=read_int(),b=read_int();
    int p=1LL*a*quick_pow(b,mod-2)%mod;
    frac[0]=1;
    _for(i,1,MAXN)frac[i]=1LL*frac[i-1]*i%mod;
    invf[MAXN-1]=quick_pow(frac[MAXN-1],mod-2);
    for(int i=MAXN-1;i;i--)
    invf[i-1]=1LL*invf[i]*i%mod;
    int div=quick_pow(n-1,mod-2);
    _rep(i,0,n){
        LL t1=1-1LL*p*(n-i)%mod*div;
        LL t2=1-1LL*p*(n-1-i)%mod*div;
        g[i]=1LL*quick_pow(t1%mod,i)*quick_pow(t2%mod,n-i)%mod;
```

```
        g[i]=1LL*g[i]*invf[i]%mod;
        f[i]=(i&1)?-invf[i]:invf[i];
    }
    Poly::mul(f,n+1,g,n+1);
    _rep(i,0,n)
    f[i]=1LL*f[i]*frac[i]%mod*C(n,i)%mod;
    _rep(i,0,n)
    enter(f[n-i]);
    return 0;
}
```

J. Illuminations

题意

给一个凸多边形，和凸多边形外侧若干个点，每个点作为一盏灯，向四面八方发出光线，让用最少的点照亮平面除凸包内部外所有区域，如果不存在方案输出 -1 。

题解

这题可以分成两个部分，一个是求凸包切线部分，一个是求环区间最小覆盖部分。

我们注意到，一个点能照亮的最大区域是这个点对于这个凸包求左右两条切线，然后点亮所有区域的等价条件是所有边都被一个点的两条切线夹起来过，求切线就是一个二分的板子，然后这个问题就转化为环形结构内给若干个线段（可以跨过原点），求最少的线段的数量，覆盖 1 到 n 的所有点。

关于求环区间最小覆盖部分，首先肯定是断环成链，然后枚举 $i=1\sim n$ 区间 $[i,i+n)$ 的最小线段覆盖，但这是 $O\left(n^2\right)$ 的，有以下几种解法。

第一种，本人最初过题思路，设 $\text{dp}(l,r)$ 表示区间 $[l,r]$ 的最小线段覆盖。

固定 l，显然 $\text{dp}(l,r)$ 是分段的。然后对给定 l，显然策略为找到覆盖他的线段中的最大的右端点 k。

于是有

$$ \text{dp}(l,i) \begin{cases} 1, &i\le k\\ \text{dp}(k+1,i)+1, &i\gt k \end{cases} $$

于是可以可持久化线段树维护每个 $\text{dp}(i,\ast)$ 数组的所有分段点，然后 $O(n\log n)$ 查询 $\text{dp}(i,i+n-1)(i=1\sim n)$，得到答案。

```
const double eps=1e-11;
const double inf=1e20;
const double pi=acos(-1);
const int maxp=300020;
int n,m;
struct xd {
    int id,l,r;
```

```
} xx[maxp];
int xxs,maxv;
int sgn(double x) {
    if(fabs(x)<eps) return 0;
    if(x<0) return -1;
    return 1;
}

struct Point {
    double x,y;
    Point() {}
    Point(double _x,double _y) {
        x = _x;
        y = _y;
    }
    void input() {
        scanf("%lf%lf",&x,&y);
    }
    void output() {
        printf("%.2f %.2f\n",x,y);
    }
    bool operator == (Point b)const {
        return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;
    }
    bool operator < (Point b)const {
        return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;
    }
    Point operator -(const Point &b)const {
        return Point(x-b.x,y-b.y);
    }
    //叉积
    double operator ^(const Point &b)const {
        return x*b.y-y*b.x;
    }
    //点积
    double operator *(const Point &b)const {
        return x*b.x + y*b.y;
    }
    //返回长度
    double len() {
        return hypot(x,y);//库函数
    }
    //返回长度的平方
    double len2() {
        return x*x + y*y;
    }
    //返回两点的距离
    double distance(Point p) {
        return hypot(x-p.x,y-p.y);
    }
    Point operator +(const Point &b)const {
```

```
        return Point(x+b.x,y+b.y);
    }
    Point operator *(const double &k)const {
        return Point(x*k,y*k);
    }
    Point operator /(const double &k)const {
        return Point(x/k,y/k);
    }
    //计算 pa 和pb 的夹角
    //就是求这个点看 a,b 所成的夹角
    //测试 LightOJ1203
    double rad(Point a,Point b) {
        Point p = *this;
        return fabs(atan2(fabs((a-p)^(b-p)),(a-p)*(b-p)));
    }
    //计算 pa 和pb 的有向角
    double tmprad(Point a,Point b) {
        Point p = *this;
        return atan2(((a-p)^(b-p)),(a-p)*(b-p));
    }
    //化为长度为 r 的向量
    Point trunc(double r) {
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point(x*r,y*r);
    }
    //逆时针旋转90度
    Point rotleft() {
        return Point(-y,x);
    }
    //顺时针旋转90度
    Point rotright() {
        return Point(y,-x);
    }
    //绕着 p 点逆时针旋转 angle
    Point rotate(Point p,double angle) {
        Point v = (*this)-p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x*c-v.y*s,p.y + v.x*s + v.y*c);
    }
};

struct Line {
    Point s,e;
    Line() {

    }
    Line(Point _s,Point _e) {
        s=_s;
```

```
        e=_e;
    }

};
struct polygon {
    int n;
    Point p[maxp];
    Line l[maxp];
    void input(int _n) {
        n = _n;
        for(int i = 0; i < n; i++)
            p[i].input();
    }
    void add(Point q) {
        p[n++] = q;
    }
    void getline() {
        for(int i = 0; i < n; i++) {
            l[i] = Line(p[i],p[(i+1)%n]);
        }
    }
    struct cmp {
        Point p;
        cmp(const Point &p0) {
            p = p0;
        }
        bool operator()(const Point &aa,const Point &bb) {
            Point a = aa, b = bb;
            int d = sgn((a-p)^(b-p));
            if(d == 0) {
                return sgn(a.distance(p)-b.distance(p)) < 0;
            }
            return d > 0;
        }
    };
    //进行极角排序
    //首先需要找到最左下角的点
    //需要重载号好 Point 的<操作符 (min 函数要用)
    void norm() {
        Point mi = p[0];
        for(int i = 1; i < n; i++)mi = min(mi,p[i]);
        sort(p,p+n,cmp(mi));
    }
    //得到凸包的另外一种方法
    //测试 LightOJ1203 LightOJ1239
    void Graham(polygon &convex) {
        norm();
        int &top = convex.n;
        top = 0;
        if(n == 1) {
            top = 1;
```

```
            convex.p[0] = p[0];
            return;
        }
        if(n == 2) {
            top = 2;
            convex.p[0] = p[0];
            convex.p[1] = p[1];
            if(convex.p[0] == convex.p[1])top--;
            return;
        }
        convex.p[0] = p[0];
        convex.p[1] = p[1];
        top = 2;
        for(int i = 2; i < n; i++) {
            while( top > 1 && sgn((convex.p[top-1]-convex.p[top-2])^(p[i]-
convex.p[top-2])) <= 0 )top--;
            convex.p[top++] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;//特判
    }

    //下面是过凸包外一点求凸包左右切点的板子
    int getl(int l,int r,Point po) {
        int ans=l,mid;
        l++;
        while(l<=r) {
            mid=l+r>>1;
            if(sgn((p[mid]-po)^(p[(mid-1+n)%n]-p[mid]))<=0) {
                ans=mid;
                l=mid+1;
            } else r=mid-1;
        }
        return ans;
    }

    int getr(int l,int r,Point po) {
        int ans=r,mid;
        r--;
        while(l<=r) {
            mid=l+r>>1;
            if(sgn((p[mid]-po)^(p[(mid+1)%n]-p[mid]))>=0) {
                ans=mid;
                r=mid-1;
            } else l=mid+1;
        }
        return ans;
    }

    void work(Point po,int id) {
        int pos=0;
```

```
        if(sgn(po.x)>0) {
            int l=1,r=n-1,mid;
            while(l<=r) {
                mid=l+r>>1;
                if(sgn(p[mid]^po)>=0) {
                    pos=mid;
                    l=mid+1;
                } else {
                    r=mid-1;
                }
            }
        } else if(sgn(po.y)>0) {
            pos=n-1;
        }
        int l,r;
        if(sgn(po.x)>0) {
            l=getl(0,pos,po);
            r=getr(pos,n,po);
        } else {
            l=getl(maxv,n,po);
            r=getr(0,maxv,po);
        }
        if(r==0) r=n;
        xx[++xxs]= {id,l+1,r};
    }
    //到此结束
} PO,po,po1,PPO;

namespace Tree {
    const int MAXN=4e5+5;
    int lef[MAXN<<2],rig[MAXN<<2];
    pair<int,int> s[MAXN<<2],lazy[MAXN<<2];
    void build(int k,int L,int R) {
        lef[k]=L,rig[k]=R;
        if(lef[k]==rig[k])
            return;
        int M=L+R>>1;
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
    }
    void push_tag(int k,pair<int,int> v) {
        s[k]=max(s[k],v);
        lazy[k]=max(lazy[k],v);
    }
    void push_up(int k) {
        s[k]=max(s[k<<1],s[k<<1|1]);
    }
    void push_down(int k) {
        if(lazy[k].first) {
            push_tag(k<<1,lazy[k]);
            push_tag(k<<1|1,lazy[k]);
```

```
            lazy[k]=make_pair(0,0);
        }
    }
    void update(int k,int L,int R,pair<int,int> v) {
        if(L<=lef[k]&&rig[k]<=R) {
            push_tag(k,v);
            return;
        }
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=L)
            update(k<<1,L,R,v);
        if(mid<R)
            update(k<<1|1,L,R,v);
        push_up(k);
    }
    pair<int,int> query(int k,int pos) {
        if(lef[k]==rig[k])
            return s[k];
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=pos)
            return query(k<<1,pos);
        else
            return query(k<<1|1,pos);
    }
}
namespace JXM {
    const int MAXN=4e5+5;
    struct Node {
        int s,v,lch,rch;
    } node[MAXN*40];
    int root[MAXN],node_cnt;
    void update(int &k,int p,int vl,int vr,int pos,int v) {
        node[k=++node_cnt]=node[p];
        node[k].s++;
        if(vl==vr) {
            node[k].v=v;
            return;
        }
        int vm=vl+vr>>1;
        if(vm>=pos)
            update(node[k].lch,node[p].lch,vl,vm,pos,v);
        else
            update(node[k].rch,node[p].rch,vm+1,vr,pos,v);
    }
    int query_max(int k,int vl,int vr) {
        if(vl==vr)return vl;
        int vm=vl+vr>>1;
        if(node[k].rch)
```

```
            return query_max(node[k].rch,vm+1,vr);
        else
            return query_max(node[k].lch,vl,vm);
    }
    int query(int k,int vl,int vr,int pos) {
        if(!k)return 0;
        if(vl==vr)return 1;
        int vm=vl+vr>>1;
        if(vm>=pos)
            return query(node[k].lch,vl,vm,pos);
        else
            return node[node[k].lch].s+query(node[k].rch,vm+1,vr,pos);
    }
    vector<pair<int,int> > res;
    void dfs(int k,int vl,int vr) {
        if(!k)return;
        if(vl==vr) {
            res.push_back(make_pair(vl,node[k].v));
            return;
        }
        int vm=vl+vr>>1;
        dfs(node[k].lch,vl,vm);
        dfs(node[k].rch,vm+1,vr);
    }
    void pt(int v) {
        enter(v);
        dfs(root[v],1,6);
        for(pair<int,int> p:res) {
            space(p.second);
            enter(p.first);
        }
        puts("");
        res.clear();
    }
    void solve() {
        int m=xxs;
        int n=PO.n,n2=PO.n<<1;
        Tree::build(1,1,n2);
        _rep(i,1,m) {
            if(xx[i].r<xx[i].l)xx[i].r+=n;
            Tree::update(1,xx[i].l,xx[i].r,make_pair(xx[i].r,xx[i].id));
        }
        int ans=MAXN;
        for(int i=n2; i; i--) {
            pair<int,int> t=Tree::query(1,i);
            if(t.first!=0) {
                update(root[i],root[t.first+1],1,n2,t.first,t.second);
                if(i<=n&&query_max(root[i],1,n2)>=i+n-1)
                    ans=min(ans,query(root[i],1,n2,i+n-2)+1);
            }
        }
```

```
        if(ans==MAXN) {
            puts("-1");
            return;
        }
        enter(ans);
        _rep(i,1,n) {
            if(root[i]&&query_max(root[i],1,n2)>=i+n-1) {
                if(ans==query(root[i],1,n2,i+n-2)+1) {
                    dfs(root[i],1,n2);
                    for(pair<int,int> p:res) {
                        space(p.second);
                        if(p.first>=i+n-1)
                            break;
                    }
                    return;
                }
            }
        }
    }
}

int main() {
    cin>>n>>m;
    PPO.n=PO.n=n,po.n=m;
    for(int i=0; i<n; i++) {
        scanf("%lf %lf",&PPO.p[i].x,&PPO.p[i].y);
    }
    PPO.Graham(PO);
    Point pp=PO.p[0];
    for(int i=0; i<n; i++) PO.p[i]=PO.p[i]-pp;
    for(int i=0; i<n; i++) if(sgn(PO.p[i].x-PO.p[maxv].x)>=0) maxv=i;
    for(int i=0; i<m; i++) {
        scanf("%lf %lf",&po.p[i].x,&po.p[i].y);
        po.p[i]=po.p[i]-pp;
    }
    for(int i=0; i<m; i++) PO.work(po.p[i],i+1);
    JXM::solve();
    return 0;
}
```

第二种，由于对给定 l最优策略为找到覆盖他的线段中的最大的右端点 r然后跳到 $r+1$于是每个状态的后继都是唯一的。

因此如果对每个 l建边 $l\to r+1$可以得到一棵树。

考虑树上倍增，可以 $O(n\log n)$ 查询每个结点 $i=1\sim n$ 跳到不小于 $i+n-1$ 的祖先的最小步数得到答案。

另外这也可以通过 dfs 维护根到当前结点的路径然后二分来实现。

第三种，首先淘汰掉被另一条线段完全包含的线段，然后选取余下的线段中最短的线段的每个位置作为起

点暴力跳查询答案。

设最短的线段长度为 L，则每次跳的长度不小于 L，于是最多跳 $O(\frac nL)$ 次，于是总复杂度 $O(L\times \frac nL)\sim O(n)$。

关于淘汰被另一条线段完全包含的线段，也可以利用桶排等技巧 $O(n)$ 实现。

关于为什么选最短的线段的每个位置作为起点可以保证得到最优解，本人无法证明。

```
const double eps=1e-11;
const double inf=1e20;
const double pi=acos(-1);
const int maxp=300020;
int n,m;
struct xd {
    int id,l,r;
} xx[maxp];
int xxs,maxv;
int sgn(double x) {
    if(fabs(x)<eps) return 0;
    if(x<0) return -1;
    return 1;
}

struct Point {
    double x,y;
    Point() {}
    Point(double _x,double _y) {
        x = _x;
        y = _y;
    }
    void input() {
        scanf("%lf%lf",&x,&y);
    }
    void output() {
        printf("%.2f %.2f\n",x,y);
    }
    bool operator == (Point b)const {
        return sgn(x-b.x) == 0 && sgn(y-b.y) == 0;
    }
    bool operator < (Point b)const {
        return sgn(x-b.x)== 0?sgn(y-b.y)<0:x<b.x;
    }
    Point operator -(const Point &b)const {
        return Point(x-b.x,y-b.y);
    }
    //叉积
    double operator ^(const Point &b)const {
        return x*b.y-y*b.x;
    }
    //点积
```

```
    double operator *(const Point &b)const {
        return x*b.x + y*b.y;
    }
    //返回长度
    double len() {
        return hypot(x,y);//库函数
    }
    //返回长度的平方
    double len2() {
        return x*x + y*y;
    }
    //返回两点的距离
    double distance(Point p) {
        return hypot(x-p.x,y-p.y);
    }
    Point operator +(const Point &b)const {
        return Point(x+b.x,y+b.y);
    }
    Point operator *(const double &k)const {
        return Point(x*k,y*k);
    }
    Point operator /(const double &k)const {
        return Point(x/k,y/k);
    }
    //计算 pa 和pb 的夹角
    //就是求这个点看 a,b 所成的夹角
    //测试 LightOJ1203
    double rad(Point a,Point b) {
        Point p = *this;
        return fabs(atan2(fabs((a-p)^(b-p)),(a-p)*(b-p)));
    }
    //计算 pa 和pb 的有向角
    double tmprad(Point a,Point b) {
        Point p = *this;
        return atan2(((a-p)^(b-p)),(a-p)*(b-p));
    }
    //化为长度为 r 的向量
    Point trunc(double r) {
        double l = len();
        if(!sgn(l))return *this;
        r /= l;
        return Point(x*r,y*r);
    }
    //逆时针旋转 90 度
    Point rotleft() {
        return Point(-y,x);
    }
    //顺时针旋转 90 度
    Point rotright() {
        return Point(y,-x);
```

```
    }
    //绕着 p 点逆时针旋转 angle
    Point rotate(Point p,double angle) {
        Point v = (*this)-p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x*c-v.y*s,p.y + v.x*s + v.y*c);
    }
};

struct Line {
    Point s,e;
    Line() {

    }
    Line(Point _s,Point _e) {
        s=_s;
        e=_e;
    }

};
struct polygon {
    int n;
    Point p[maxp];
    Line l[maxp];
    void input(int _n) {
        n = _n;
        for(int i = 0; i < n; i++)
            p[i].input();
    }
    void add(Point q) {
        p[n++] = q;
    }
    void getline() {
        for(int i = 0; i < n; i++) {
            l[i] = Line(p[i],p[(i+1)%n]);
        }
    }
    struct cmp {
        Point p;
        cmp(const Point &p0) {
            p = p0;
        }
        bool operator()(const Point &aa,const Point &bb) {
            Point a = aa, b = bb;
            int d = sgn((a-p)^(b-p));
            if(d == 0) {
                return sgn(a.distance(p)-b.distance(p)) < 0;
            }
            return d > 0;
        }
    };
```

```
    //进行极角排序
    //首先需要找到最左下角的点
    //需要重载号好 Point 的<操作符 (min 函数要用)
    void norm() {
        Point mi = p[0];
        for(int i = 1; i < n; i++)mi = min(mi,p[i]);
        sort(p,p+n,cmp(mi));
    }
    //得到凸包的另外一种方法
    //测试 LightOJ1203 LightOJ1239
    void Graham(polygon &convex) {
        norm();
        int &top = convex.n;
        top = 0;
        if(n == 1) {
            top = 1;
            convex.p[0] = p[0];
            return;
        }
        if(n == 2) {
            top = 2;
            convex.p[0] = p[0];
            convex.p[1] = p[1];
            if(convex.p[0] == convex.p[1])top--;
            return;
        }
        convex.p[0] = p[0];
        convex.p[1] = p[1];
        top = 2;
        for(int i = 2; i < n; i++) {
            while( top > 1 && sgn((convex.p[top-1]-convex.p[top-2])^(p[i]-
convex.p[top-2])) <= 0 )top--;
            convex.p[top++] = p[i];
        }
        if(convex.n == 2 && (convex.p[0] == convex.p[1]))convex.n--;//特判
    }

    //下面是过凸包外一点求凸包左右切点的板子
    int getl(int l,int r,Point po) {
        int ans=l,mid;
        l++;
        while(l<=r) {
            mid=l+r>>1;
            if(sgn((p[mid]-po)^(p[(mid-1+n)%n]-p[mid]))<=0) {
                ans=mid;
                l=mid+1;
            } else r=mid-1;
        }
        return ans;
    }
```

```
    int getr(int l,int r,Point po) {
        int ans=r,mid;
        r--;
        while(l<=r) {
            mid=l+r>>1;
            if(sgn((p[mid]-po)^(p[(mid+1)%n]-p[mid]))>=0) {
                ans=mid;
                r=mid-1;
            } else l=mid+1;
        }
        return ans;
    }

    void work(Point po,int id) {
        int pos=0;
        if(sgn(po.x)>0) {
            int l=1,r=n-1,mid;
            while(l<=r) {
                mid=l+r>>1;
                if(sgn(p[mid]^po)>=0) {
                    pos=mid;
                    l=mid+1;
                } else {
                    r=mid-1;
                }
            }
        } else if(sgn(po.y)>0) {
            pos=n-1;
        }
        int l,r;
        if(sgn(po.x)>0) {
            l=getl(0,pos,po);
            r=getr(pos,n,po);
        } else {
            l=getl(maxv,n,po);
            r=getr(0,maxv,po);
        }
        if(r==0) r=n;
        xx[++xxs]= {id,l+1,r};
    }
    //到此结束
} PO,po,po1,PPO;
namespace JXM {
    const int MAXN=4e5+5;
    vector<xd> c[MAXN];
    pair<int,int> dp[MAXN];
    void solve() {
        int n=PO.n,n2=PO.n<<1;
        int m=xxs;
        _rep(i,1,m){
```

```
            if(xx[i].r<xx[i].l)xx[i].r+=n;
            c[xx[i].r-xx[i].l].push_back(xx[i]);
        }
        m=0;
        _for(i,0,MAXN){
            for(xd t:c[i])
            xx[++m]=t;
        }
        _rep(i,1,m){
//              space(i);space(xx[i].l);enter(xx[i].r);
            dp[xx[i].l]=make_pair(xx[i].r,i);
        }
        _rep(i,1,n2)
        dp[i]=max(dp[i],dp[i-1]);
        _rep(i,1,n){
            pair<int,int> t1=dp[i],t2=dp[i+n];
            dp[i]=max(t1,make_pair(t2.first-n,t2.second));
            dp[i+n]=max(make_pair(t1.first+n,t1.second),t2);
        }
        int pos=0;
        _rep(i,1,n){
            if(dp[i].first<i){
                puts("-1");
                return;
            }
            if(!pos)
            pos=dp[i].second;
            else if(xx[pos].r-xx[pos].l>xx[dp[i].second].r-
xx[dp[i].second].l)
            pos=dp[i].second;
        }
        int ans=MAXN;
        _rep(i,xx[pos].l,xx[pos].r){
            int st=(i-1)%n+1,pos2=st,cnt=0;
            while(pos2<st+n){
                pos2=dp[pos2].first+1;
                cnt++;
            }
            ans=min(ans,cnt);
        }
        enter(ans);
        _rep(i,xx[pos].l,xx[pos].r){
            int st=(i-1)%n+1,pos2=st,cnt=0;
            while(pos2<st+n){
                pos2=dp[pos2].first+1;
                cnt++;
            }
            if(ans==cnt){
                int pos2=st;
                while(pos2<st+n){
```

```
                    space(xx[dp[pos2].second].id);
                    pos2=dp[pos2].first+1;
                }
                return;
            }
        }
    }
}

int main() {
    cin>>n>>m;
    PPO.n=PO.n=n,po.n=m;
    for(int i=0; i<n; i++) {
        scanf("%lf %lf",&PPO.p[i].x,&PPO.p[i].y);
    }
    PPO.Graham(PO);
    Point pp=PO.p[0];
    for(int i=0; i<n; i++) PO.p[i]=PO.p[i]-pp;
    for(int i=0; i<n; i++) if(sgn(PO.p[i].x-PO.p[maxv].x)>=0) maxv=i;
    for(int i=0; i<m; i++) {
        scanf("%lf %lf",&po.p[i].x,&po.p[i].y);
        po.p[i]=po.p[i]-pp;
    }
    for(int i=0; i<m; i++) PO.work(po.p[i],i+1);
    JXM::solve();
    return 0;
}
```

K. Walking

题意

给定一个 $n\times m$ 的网格和初始点 (a,b)，求从初始点出发移动 t 步且始终不出界的情况下的所有走法。

题解

显然横轴坐标是独立的，可以分开考虑。

设 $f(s,n,a)$ 表示从一维坐标轴从 a 点出发走 s 步且始终处于 $[1,n]$ 范围内的情况下的所有走法。于是答案为

$$ \sum_{i=0}^t {t\choose i}f(i,n,a)f(t-i,m,b) $$

接下来考虑如何计算 $f(s,n,a)(s\in [0,t])$，$f(s,m,b)$ 的计算方式类同。

方案一：设 $\text{dp}(i,j)$ 表示走 i 步最后位于 j 点且始终为出界的方案数，不难得到一个 $O(nt)$ 的暴力解法。

方案二：不难发现，有

$$ \begin{equation}\begin{split} f(s,n,a)&=\sum_{i=1}^n \text{dp}(s,i) \\ &=\text{dp}(s-1,1)+\sum_{i=2}^{n-1} (\text{dp}(s-1,i-1)+\text{dp}(s-1,i+1))+\text{dp}(s-1,n)\\ &=2f(s-1,n,a)-\text{dp}(s-1,1)-\text{dp}(s-1,n) \end{split}\end{equation} $$ 于是问题转化为计算 $\text{dp}(s,1),\text{dp}(s,n)$。

对每个移动方案，定义非法序列，每当点进入 $(-\infty,0)$ 时非法序列末尾加上 L。每当点进入 $(n,+\infty)$ 时非法序列末尾加上 R。

对于 $\text{dp}(s,1)$，我们需要获得所有非法序列为空串的移动方案。设 $h(a,b,s)$ 表示从 a 移动 s 步到达 b 的方案数。

显然根据 a,b,s 奇偶性以及预处理组合数可以 $O(1)$ 计算 $h(a,b,s)$。

然后总移动方案为 $h(a,1,s)$，利用容斥，首先我们减去非法序列为 L+.* 和 R+.* (非法序列均用正则表达式表示)的移动方案。

设 $l(x)=-x,r(x)=2(n+1)-x$，根据简单组合数学知识，不难发现 L+.* 代表的方案为 $h(a,l(1),s)$，R+.* 代表的方案为 $h(a,r(1),s)$。

接下来我们需要补上减去非法序列为 L+R+.* 和 R+L+.* 的移动方案，分别为 $h(a,r(l(1)),s),h(a,l(r(1)),s)$，依次类推，有

$$ \text{dp}(s,1)=h(a,1,s)-h(a,l(1),s)-h(a,r(1),s)+h(a,r(l(1)),s)+h(a,l(r(1)),s)-h(a,l(r(l(1))),s)-h(a,r(l(r(1))),s)+\cdots $$

由于 $r(l(x))=2(n+1)+x$，且当 $\text{abs}(a-x)\gt s$ 时一定有 $h(a,x,s)=0$，所以上述容斥最多迭代 $O(\frac sn)$ 次。

于是方案二的总时间复杂度为 $O\left(\sum_{i=1}^t \frac in\right)\sim O\left(\frac {t^2}n\right)$。

考虑根号分治，当 $n\lt \sqrt t$ 时采用方案一，否则采用方案二。总时间复杂度 $O(t\sqrt t)$。

```
const int MAXN=5e5+5,MAXM=800,mod=998244353;
int quick_pow(int n,int k){
    int ans=1;
    while(k){
        if(k&1)ans=1LL*ans*n%mod;
        n=1LL*n*n%mod;
        k>>=1;
    }
    return ans;
}
int frac[MAXN],invf[MAXN];
int C(int n,int m){
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
void init(){
    frac[0]=1;
    _for(i,1,MAXN)frac[i]=1LL*frac[i-1]*i%mod;
    invf[MAXN-1]=quick_pow(frac[MAXN-1],mod-2);
```

```
    for(int i=MAXN-1;i;i--)
    invf[i-1]=1LL*invf[i]*i%mod;
}
int ans1[MAXN],ans2[MAXN];
int dp[2][MAXM];
void solve1(int s,int n,int a,int *ans){
    int pos=0;
    mem(dp[pos],0);
    dp[pos][a]=1;
    ans[0]=1;
    _rep(i,1,s){
        pos=!pos;
        mem(dp[pos],0);
        _rep(j,1,n){
            dp[pos][j]=(dp[!pos][j-1]+dp[!pos][j+1])%mod;
            ans[i]=(ans[i]+dp[pos][j])%mod;
        }
    }
}
int cal(int s,int n,int a,int pos){
    int pos1=pos,pos2=pos,ans=0,d=abs(pos-a);
    if(d<=s&&(s-d)%2==0)
    ans=C(s,(s+d)/2);
    for(int j=1;;j++){
        if(j&1){
            pos1=-pos1;
            pos2=2*(n+1)-pos2;
        }
        else{
            pos1=2*(n+1)-pos1;
            pos2=-pos2;
        }
        int d1=abs(pos1-a),d2=abs(pos2-a),det=0;
        if(d1>s&&d2>s)break;
        if(d1<=s&&(s-d1)%2==0)
        det=(det+C(s,(s+d1)/2));
        if(d2<=s&&(s-d2)%2==0)
        det=(det+C(s,(s+d2)/2))%mod;
        if(j&1)
        ans=(ans+mod-det)%mod;
        else
        ans=(ans+det)%mod;
    }
    return ans;
}
void solve2(int s,int n,int a,int *ans){
    ans[0]=1;
    _rep(i,1,s)
    ans[i]=(2LL*ans[i-1]+mod-cal(i-1,n,a,1)+mod-cal(i-1,n,a,n))%mod;
}
void solve(int s,int n,int a,int *ans){
```

```
    if(1LL*n*n<s)
    solve1(s,n,a,ans);
    else
    solve2(s,n,a,ans);
}
int main()
{
    init();
    int t=read_int(),n=read_int(),m=read_int(),a=read_int(),b=read_int();
    solve(t,n,a,ans1);
    solve(t,m,b,ans2);
    int ans=0;
    _rep(i,0,t)
    ans=(ans+1LL*C(t,i)*ans1[i]%mod*ans2[t-i])%mod;
    enter(ans);
    return 0;
}
```