Codeforces Round #706 (Div. 1)

比赛链接

C. Garden of the Sun

题意

给定一些黑格和一些白格，要求将一些白格转化为黑格，使得所有黑格连通但不出现环路。

题目保证以起始时以每个黑格为中心的 3×3 范围内没有其他黑格。

题解

当 $n\equiv 1\bmod 3$ 时，考虑将第 $1,4,7\cdots$ 行染成黑色，然后对第 $3k+2,3k+3$ 行的每列，最多只有一个黑格。

如果第 $3k+2,3k+3$ 行间存在黑格，直接将两行的任意一个黑格所在列全染黑，否则将两行的第一列染黑。易知这样即可完成构造。

当 $n\not\equiv 1\bmod 3$ 时，将 $1,4,7\cdots$ 行换成第 $2,5,8\cdots$ 行处理即可。

```
const int MAXN=505;
char buf[MAXN][MAXN];
int main()
{
    int T=read_int();
    while(T--){
        int n=read_int(),m=read_int();
        _for(i,0,n)scanf("%s",buf[i]);
        int s1,s2;
        if(n%3==1){
            s1=0;
            s2=2;
        }
        else{
            s1=1;
            s2=3;
        }
        for(int i=s1;i<n;i+=3)_for(j,0,m)buf[i][j]='X';
        for(int i=s2;i<n;i+=3){
            bool flag=false;
            _for(j,0,m){
                if(buf[i-1][j]=='X'||buf[i][j]=='X'){
                    buf[i-1][j]=buf[i][j]='X';
                    flag=true;
```

```
                    break;
                }
            }
            if(!flag)
            buf[i-1][0]=buf[i][0]='X';
        }
        _for(i,0,n)puts(buf[i]);
    }
    return 0;
}
```

D. BFS Trees

题意

给定一个连通图，定义以点 x 为根的 BFS 树是生成树且树上所有点到 x 的距离等于连通图该点到 x 的距离。

定义 $f(x,y)$ 表示既满足是以点 x 为根的 BFS 树同时也是以点 y 为根的 BFS 树的生成树的个数。

题解

首先给出结论：假定 x,y 之间有超过一条最短路径，则 $f(x,y)=0$□

因为对于 x,y 之间最短路上的点 u□必有 $\text{dis}(x,u)+\text{dis}(u,y)=\text{dis}(x,y)$□易知树上满足该条件的点仅 $\text{dis}(x,y)+1$ 个。

假如 x,y 之间有超过一条最短路径，则图中满足 $\text{dis}(x,u)+\text{dis}(u,y)=\text{dis}(x,y)$ 的点必然超过 $\text{dis}(x,y)+1$ 个，矛盾。

接下来仅考虑 x,y 之间仅有一条最短路的情况，首先易知生成树一定包含 x,y 之间的最短路。

接下来对除最短路外原图中的每条边 $u\to v$□假如保留该边，则必有 $\text{dis}(x,u)=\text{dis}(x,v)\pm 1,\text{dis}(y,u)=\text{dis}(y,v)\pm 1$□

假如 $\text{dis}(x,u)+\text{dis}(u,y)=\text{dis}(x,v)+\text{dis}(v,y)$□则 u,v 必然在 $x\to y$ 的路径上，矛盾。

于是为每个不在最短路上的结点指定一个父结点即可，答案即为所有点的所有可选父结点的个数的乘积。

设当前结点为 u□父结点为 v□则父结点应该满足 $\text{dis}(x,u)=\text{dis}(x,v)+1$ 且 $\text{dis}(y,u)=\text{dis}(y,v)+1$□

于是可以 $O(m)$ 计算出每个 $f(x,y)$□总时间复杂度 $O(n^2m)$□

```
const int MAXN=405,MAXM=605,Mod=998244353;
```

```
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt,dis[MAXN][MAXN];
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int main()
{
    int n=read_int(),m=read_int();
    mem(dis,127/3);
    _rep(i,1,n)dis[i][i]=0;
    _for(i,0,m){
        int u=read_int(),v=read_int();
        dis[u][v]=dis[v][u]=1;
        Insert(u,v);Insert(v,u);
    }
    _rep(k,1,n)_rep(i,1,n)_rep(j,1,n)
    dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j]);
    _rep(x,1,n){
        _rep(y,1,n){
            int ans=1,cnt=0;
            _rep(u,1,n){
                if(dis[x][u]+dis[u][y]==dis[x][y])cnt++;
                else{
                    int cnt2=0;
                    for(int i=head[u];i;i=edge[i].next){
                        int v=edge[i].to;
                        if(dis[x][u]==dis[x][v]+1&&dis[y][u]==dis[y][v]+1)
                        cnt2++;
                    }
                    ans=1LL*ans*cnt2%Mod;
                }
            }
            space(cnt==dis[x][y]+1?ans:0);
        }
        puts("");
    }
    return 0;
}
```

E. Qingshan and Daniel

题意

给定一个圈，圈上有两个队伍的人，每个人有 a_i 张卡。

从第一个人开始弃一张卡，然后顺时针找到下一个不同队伍的且手牌不为零的人，继续弃卡，直到有一个

队伍没发弃卡为止。

问游戏结束时每个人手上的卡。

题解

不难发现当两个队伍卡牌总数相同时第一个人所在队伍败北，否则卡片总数少的队伍败北。

另外对于败北的队伍，显然每个人的卡牌数量为 0。

接下来考虑获胜队伍的每个人的卡牌数量，首先如果第一个人在获胜队伍，则他先弃一张卡，轮到失败队伍。

接下来仅考虑失败队伍先开始弃卡的情况。

不难发现失败队伍的每个人弃牌的顺序不影响最终结果，于是将每个人弃的卡转移到下一个人处理即可。

考虑到一次遍历圆环可能会仍有剩余卡牌，于是二重遍历圆环，时间复杂度 $O(n)$

```
const int MAXN=5e6+5,Mod=1e9+7;
int seed,base;
int rnd(){
    int ret=seed;
    seed=(1LL*seed*base+233)%Mod;
    return ret;
}
int a[MAXN],b[MAXN],t[MAXN];
LL s[2];
int main()
{
    int n=read_int(),m=read_int(),pos=0;
    _for(i,0,m){
        int p=read_int(),k=read_int();
        seed=read_int();
        base=read_int();
        while(pos<p){
            t[pos]=rnd()%2;
            a[pos]=b[pos]=rnd()%k+1;
            pos++;
        }
    }
    _for(i,0,n)
    s[t[i]]+=a[i];
    int flag=(s[0]^s[1])?(s[0]<s[1]):(!t[0]);
    if(t[0]==flag)
    b[0]--;
    LL pre=0;
    _for(j,0,2){
        _for(i,0,n){
            if(t[i]!=flag){
```

```
                pre+=b[i];
                b[i]=0;
            }
            else{
                int temp=b[i]<pre?b[i]:pre;
                b[i]-=temp;
                pre-=temp;
            }
        }
    }
    int ans=1;
    _for(i,0,n)
    ans=1LL*ans*(((a[i]-b[i])^(1LL*(i+1)*(i+1)))%Mod+1)%Mod;
    enter(ans);
    return 0;
}
```

F. Squares

题意

给定一个长度为 n 的序列，每个元素有属性 p_i,a_i,b_i，其中 p_i 为 $1\sim n$ 的排列。

每轮起点为第一个元素，假设当前位于位置 i，则可以花费 a_i 到达 $i+1$ 或花费 b_i 到达 i 右边第一个 $p_j\gt j$ 的位置。

当 $i+1\gt n$ 时花费 a_i 可以到达终点，当 j 不存在时花费 b_i 也可以到达终点。

给定一个集合 S，初始时为空。

每轮为集合加入或删除一个元素，问在遍历集合中所有元素的前提下到达终点的最小花费。

题解

令 $p_0=n+1$，将每个点向左边第一个 $p_j\gt j$ 的元素连一条边。

不难发现可以得到一棵树，且设当前处于结点 i，则如果花费 a_i 后接下来一定会遍历 i 的所有儿子结点。

如果花费 b_i 则将跳过 i 所在的子树。

设 $\text{dp}(i)$ 表示在结点 i 选择花费 a_i，离开 i 的整个子树的总费用。

设当前轮操作中选择在结点 i 花费 a_i 的结点集合为 T，$a_0=b_0=0$，此时总费用为

$$ \sum_{i\in T}a_i+\sum_{i\not\in T,f(i)\in T}b_i $$

设 $c_i=a_i-b_i+\sum_{j\in \text{child}(i)}b_j$，则上式化简为 $\sum_{i\in T}c_i$。

设 $\text{dp}(i)$ 表示 $i\in T$ 时 T 集合中 i 的子树结点的 c_i 和的最小值，则有状态转移方程

$$ \text{dp}(i)=c_i+\sum_{j\in \text{child}(i)}\min(\text{dp}(j),0) $$

对每个 S 或 T 中的元素 u该元素的父结点 p 一定属于 T否则不可能进入 p 的子树即不可能到达 u

于是 T 一定是连通集且 S 的父结点一定属于 T设 H 集合为满足该限制的最小集合，于是最小费用为

$$ \sum_{i\in H}(c_i+\sum_{j\not\in H,j\in \text{child}(i)}\min(\text{dp}(j),0))=\sum_{i\in H}(\text{dp}(i)-\sum_{j\in H,j\in \text{child}(i)}\min(\text{dp}(j),0))=\text{dp}(0)+\sum_{i\in H}(\text{dp}(i)-\min(\text{dp}(i),0)) $$

$$ \begin{equation}\begin{split} \sum_{i\in T}c_i&=\sum_{i\in H}\left(c_i+\sum_{j\not\in H,j\in \text{child}(i)}\min(\text{dp}(j),0)\right) \\ &=\sum_{i\in H}\left(\text{dp}(i)-\sum_{j\in H,j\in \text{child}(i)}\min(\text{dp}(j),0)\right)\\ &=\text{dp}(0)+\sum_{i\in H}(\text{dp}(i)-\min(\text{dp}(i),0))\\ &=\text{dp}(0)+\sum_{i\in H}\max(\text{dp}(i),0) \end{split}\end{equation} $$

接下来考虑如何维护 H 集合即可，将点权转移为到父结点的边权，发现可以类似虚树 dp 的方式维护最小连通集合。

另外由于本题建图的特殊性，导致 dfs 序可以等于结点编号。

时间复杂度 $O(n\log n)$

```
typedef set<int>::iterator iter;
const int MAXN=2e5+5;
struct Edge{
    int to,next;
}edge[MAXN];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int st[MAXN],tp,p[MAXN],a[MAXN],b[MAXN],cnt[MAXN];
LL c[MAXN],dp[MAXN];
bool vis[MAXN];
namespace LCA{
    int d[MAXN],sz[MAXN],f[MAXN];
    int h_son[MAXN],mson[MAXN],p[MAXN];
    LL dis[MAXN];
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        dp[u]=c[u];
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            dfs_1(v,u,depth+1);
            dp[u]+=min(dp[v],0LL);
```

```
            sz[u]+=sz[v];
            if(sz[v]>mson[u])
            h_son[u]=v,mson[u]=sz[v];
        }
    }
    void dfs_2(int u,int top){
        p[u]=top;
        if(mson[u])dfs_2(h_son[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==h_son[u])
            continue;
            dfs_2(v,v);
        }
    }
    void dfs_3(int u){
        dis[u]+=max(dp[u],0LL);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            dis[v]+=dis[u];
            dfs_3(v);
        }
    }
    void init(int root){
        dfs_1(root,-1,0);
        dfs_2(root,root);
        for(int i=head[root];i;i=edge[i].next)
        dfs_3(edge[i].to);
    }
    int query_lca(int u,int v){
        while(p[u]!=p[v]){
            if(d[p[u]]<d[p[v]])swap(u,v);
            u=f[p[u]];
        }
        return d[u]<d[v]?u:v;
    }
    LL query_dis(int u,int v){
        return dis[u]+dis[v]-dis[query_lca(u,v)]*2;
    }
};
int main()
{
    int n=read_int(),q=read_int();
    _rep(i,1,n)p[i]=read_int();
    _rep(i,1,n)a[i]=read_int();
    _rep(i,1,n)b[i]=read_int();
    p[0]=n+1;
    _rep(i,1,n){
        while(p[st[tp]]<p[i])tp--;
        Insert(st[tp],i);
        st[++tp]=i;
```

```
    }
    _rep(u,0,n){
        c[u]=a[u]-b[u];
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            c[u]+=b[v];
        }
    }
    LCA::init(0);
    set<int> s;
    s.insert(0);
    cnt[0]++;
    LL ans=0;
    while(q--){
        int u=read_int(),p=LCA::f[u];
        if(!vis[u]){
            cnt[p]++;
            if(cnt[p]==1){
                iter rig=s.upper_bound(p);
                iter lef=--rig;
                rig++;
                rig=(rig==s.end())?s.begin():rig;
                ans+=LCA::query_dis(p,*rig);
                ans-=LCA::query_dis(*lef,*rig);
                ans+=LCA::query_dis(*lef,p);
                s.insert(p);
            }
        }
        else{
            cnt[p]--;
            if(cnt[p]==0){
                iter rig=s.find(p);
                iter lef=--rig;
                rig++;rig++;
                rig=(rig==s.end())?s.begin():rig;
                ans-=LCA::query_dis(p,*rig);
                ans+=LCA::query_dis(*lef,*rig);
                ans-=LCA::query_dis(*lef,p);
                s.erase(p);
            }
        }
        vis[u]=!vis[u];
        enter(dp[0]+ans/2);
    }
    return 0;
}
```