Codeforces Round #740 (Div. 1)

比赛链接

D. Top-Notch Insertions

题意

定义插入排序过程：遍历 $i=2\sim n$，当且仅当 $a_i\lt a_{i-1}$ 时找到最小的 j 满足 $a_i\lt a_j$ 然后将 a_i 插入位置 j，用二元组 (i,j) 记录。

多组数据，每组数据给定序列长度 n 和排序过程的所有二元组(共 m 个)，假设原序列每个值 $\in [1,n]$，求原序列的所有可能情况。

数据保证 $\sum m\le 2\times 10^5$。

题解

固定 n 和二元组序列，不难发现最终结果的下标序列唯一。假定最终结果下标序列为 p，于是有 $a_{p_1}\le a_{p_2}\le\cdots \le a_{p_n}$。

当且仅当 $a_{p_{i-1}}$ 是在插入排序时直接插在 a_{p_i} 后面时 $a_{p_{i-1}}\le a_{p_i}$ 无法取等号。

假定有 k 个位置无法取等，根据简单组合数学知识，知最终答案为 ${n+n-1-k\choose n}$。

现在需要维护无法取等的 $a_{p_{i-1}}\le a_{p_i}$ 的个数，发现正向维护是十分困难的，考虑逆序维护。

最后序列共有已经排序好的 n 个元素，设最后一个二元组为 (x,y)，则最后一个二元组取得第 y 个元素一定就是当前序列得第 y 个元素。

不难发现当前序列的第 $y+1$ 个元素一定无法取等，将他标记，然后删除当前序列第 y 个元素，继续处理倒数第二个二元组。

于是问题转化为构造一个支持查询第 k 大和删除固定元素的数据结构，显然可以线段树维护。

另外题目只保证 $\sum m$ 的范围不保证 $\sum n$ 的范围，因此需要记录被修改的位置进行复原。总时间复杂度 $O(m\log n)$。

```
const int MAXN=2e5+5,mod=998244353;
int frac[MAXN<<1],invf[MAXN<<1];
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 C(int n,int m){
    return 1LL*frac[n]*invf[m]%mod*invf[n-m]%mod;
}
int lef[MAXN<<2],rig[MAXN<<2],s[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,s[k]=R-L+1;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void update(int k,int pos,int v){
    s[k]+=v;
    if(lef[k]==rig[k])
    return;
    int mid=lef[k]+rig[k]>>1;
    if(pos<=mid)
    update(k<<1,pos,v);
    else
    update(k<<1|1,pos,v);
}
int query(int k,int rk){
    if(lef[k]==rig[k])
    return lef[k];
    if(rk<=s[k<<1])
    return query(k<<1,rk);
    else
    return query(k<<1|1,rk-s[k<<1]);
}
pair<int,int> a[MAXN];
void solve(){
    int n=read_int(),m=read_int();
    set<int> vis;
    vector<int> upd;
    _for(i,0,m)
    a[i].first=read_int(),a[i].second=read_int();
    for(int i=m-1;i>=0;i--){
        int p1=query(1,a[i].second),p2=query(1,a[i].second+1);
        vis.insert(p2);
        upd.push_back(p1);
        update(1,p1,-1);
    }
    enter(C(2*n-1-vis.size(),n));
    for(int p:upd)
    update(1,p,1);
}
void Init(){
```

```
    int N=2e5;
    build(1,1,N);
    N<<=1;
    frac[0]=1;
    _rep(i,1,N)
    frac[i]=1LL*frac[i-1]*i%mod;
    invf[N]=quick_pow(frac[N],mod-2);
    for(int i=N;i;i--)
    invf[i-1]=1LL*invf[i]*i%mod;
}
int main()
{
    Init();
    int T=read_int();
    while(T--)
    solve();
    return 0;
}
```

E. Down Below

题意

给定 n 个点 m 条边无向图。除 1 号点外每个点 i 有一个怪物，玩家能力值大于 a_i 才能击败怪物，击败怪物玩家能力值上升 b_i。

玩家从 1 号点出发，每条边可以走无限次，但如果玩家上一步是 $u\to v$则这一步不能是 $v\to u$。

玩家只有杀死第 i 个点的怪物才能进入第 i 个点，问玩家的最小初始能力值，使得玩家能否杀死所有怪物。数据保证每个点度数至少为 2。

题解

不难想到二分初始能力值，难点在于如何验证答案。

考虑维护集合 S对集合 S 的每个点 u玩家都可以从 1 号点出发，杀死 S 集合每个点的怪物后回到 u。

考虑拓展集合 S可以从集合 S 的任意点 u 出发，进行 dfs并且不经过所有 $v\in S$ 的点。

从 u 出发的可行能力值为初始能力值 $+\sum_{v\in s}b_v$按照题目规则移动，如果路径出现环，显然这条路径的所有点可以加入 S。

若存在 $u,v\in S$$u\to x,v\to x$ 是两条不相交的路径，假定 $u\to x$ 获得的能量不小于 $v\to x$ 获得的能量，则可以将路径 $u\to x\to v$ 加入 S。

因此每次拓展集合 S 可以暴力对 S 的每个点进行 dfs其他每个点至多被访问一次，时间复杂度 $O(n+m)$。

S 最多拓展 $O(n)$ 次，因此验证答案复杂度 $O(nm)$，总时间复杂度 $O(nm\log V)$。

```
const int MAXN=1e3+5,MAXM=2e3+5,inf=1e9+5;
struct Edge{
    int to,next;
}edge[MAXM<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
int a[MAXN],b[MAXN],f[MAXN],vis[MAXN];
vector<int> vec;
bool dfs(int u,int fa,LL s){
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(vis[u]==u&&vis[v]==v)continue;
        if(v==fa||a[v]>=s)continue;
        if(vis[v]){
            vec.push_back(u);
            if(vis[v]!=vis[u]){
                int pos=v;
                while(pos&&vis[pos]!=pos){
                    vec.push_back(pos);
                    pos=f[pos];
                }
            }
            return true;
        }
        vis[v]=vis[u];
        f[v]=u;
        if(dfs(v,u,s+b[v])){
            if(vis[u]!=u)
            vec.push_back(u);
            return true;
        }
    }
    return false;
}
bool check(int n,int k){
    vec.clear();
    vec.push_back(1);
    bool flag=true;
    while(flag){
        flag=false;
        LL s=k;
        _rep(u,1,n)
        vis[u]=0;
        for(int u:vec){
```

```
            vis[u]=u;
            s+=b[u];
        }
        for(int u:vec){
            if(dfs(u,0,s)){
                flag=true;
                break;
            }
        }
    }
    return vec.size()==n;
}
void solve(){
    int n=read_int(),m=read_int();
    _rep(i,2,n)
    a[i]=read_int();
    _rep(i,2,n)
    b[i]=read_int();
    _rep(i,1,n)head[i]=0;
    edge_cnt=0;
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    int lef=1,rig=inf,ans=inf;
    while(lef<=rig){
        int mid=lef+rig>>1;
        if(check(n,mid)){
            ans=mid;
            rig=mid-1;
        }
        else
        lef=mid+1;
    }
    enter(ans);
}
int main()
{
    int T=read_int();
    while(T--)
    solve();
    return 0;
}
```