比赛链接

补题情况

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

题解

D. Dumae

题意

要求构造一个 $1\sim n$ 的排列，设元素 i 的位置为 p_i需要满足 $L_i\le p_i\le R_i$同时有 m 个额外约束，表示 $p_u\lt p_v$

题解

首先如果 $p_u\lt p_v$则连一条有向边 $u\to v$然后进行拓扑，如果出现环显然无解。接下来考虑将从小到大考虑每个位置上的数。

如果不存在所有形如 $u\to v$ 的约束，那么一种经典的做法为从所有未被选中且满足 L_i 不小于当前位置的点中选中 R_i 最小的点放入该位置。

对于每个限制 $u\to v$显然要先选 u 再选 v于是可以更新约束 $R_u=\min(R_u,R_v-1)$

然后选位置时从当前入度为 0 且 L_i 不小于当前位置的点中选择一个 R_u 最小的点。时间复杂度 $O(n\log n+m)$

```
const int MAXN=3e5+5,MAXM=1e6+5;
struct Edge{
    int to,next;
}edge[MAXM];
int head[MAXN],edge_cnt,deg[MAXN];
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
    deg[v]++;
```

```
}
int L[MAXN],R[MAXN],ans[MAXN];
vector<int> c[MAXN];
struct cmp{
    bool operator () (const int a,const int b){
        return R[a]>R[b];
    }
};
int temp[MAXN],tp[MAXN];
bool topu(int n){
    int tpn=0;
    queue<int> q;
    _rep(i,1,n){
        temp[i]=deg[i];
        if(deg[i]==0)
        q.push(i);
    }
    while(!q.empty()){
        int u=q.front();q.pop();
        tp[++tpn]=u;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            deg[v]--;
            if(deg[v]==0)
            q.push(v);
        }
    }
    if(tpn!=n)
    return false;
    for(int j=n;j;j--){
        int u=tp[j];
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            R[u]=min(R[u],R[v]-1);
        }
    }
    _rep(i,1,n)
    deg[i]=temp[i];
    return true;
}
int main(){
    int n=read_int(),m=read_int();
    _rep(i,1,n){
        L[i]=read_int();
        R[i]=read_int();
    }
    while(m--){
        int u=read_int(),v=read_int();
        Insert(u,v);
    }
```

```
    if(!topu(n)){
        puts("-1");
        return 0;
    }
    _rep(i,1,n){
        if(deg[i]==0)
        c[L[i]].push_back(i);
    }
    priority_queue<int,vector<int>,cmp> q;
    bool flag=true;
    _rep(t,1,n){
        for(int u:c[t])
        q.push(u);
        if(q.empty()){
            flag=false;
            break;
        }
        int u=q.top();q.pop();
        if(R[u]<t){
            flag=false;
            break;
        }
        ans[t]=u;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            deg[v]--;
            if(deg[v]==0){
                if(L[v]<=t)
                q.push(v);
                else
                c[L[v]].push_back(v);
            }
        }
    }
    if(flag){
        _rep(i,1,n)
        enter(ans[i]);
    }
    else
    puts("-1");
    return 0;
}
```

G. Fascination Street

题意

给定 n 个位置，选择位置 i 的费用为 w_i一个合法方案要求每个位置自己被选中或者自己左右两边相邻的格子至少有一个被选中。

允许 k 次操作，每次可以交换位置 i 和位置 j 的费用。问合法方案的最小费用。

题解

直接处理交换显然不可作，但交换可以等价于对方案中选中的 $x\le k$ 个位置不付费，再对方案中没有选中的 x 个位置付费。

设 $\text{dp}(i,0/1,0/1,k_1,k_2)$ 表示只考虑前 i 个位置，保证前 $i-1$ 个位置已经合法，其中第 $i-1,i$ 个位置是否被选中。

然后有 k_1 个位置选中但没付费，有 k_2 个位置没选中但付费的方案的最小费用。不难想到 dp 转移。

边界为 $\text{dp}(0,1,0,0,0)=0$，最终答案为 $\min_{j_1+j_2\neq 0,0\le t\le k}\text{dp}(n,j1,j2,t,t)$，总时间复杂度 $O\left(nk^2\right)$。

```
const int MAXN=2.5e5+5,MAXK=11;
const LL inf=1e18;
int a[MAXN];
LL dp[2][2][2][MAXK][MAXK];
void set_min(LL &x,LL y){
    x=min(x,y);
}
int main(){
    int n=read_int(),m=read_int(),pos=0;
    _rep(i,1,n)
    a[i]=read_int();
    mem(dp[pos],127);
    dp[pos][1][0][0][0]=0;
    _rep(i,1,n){
        pos=!pos;
        mem(dp[pos],127);
        _for(j1,0,2)_for(j2,0,2)
        _rep(k1,0,m)_rep(k2,0,m){
            LL t=dp[!pos][j1][j2][k1][k2];
            set_min(dp[pos][j2][1][k1][k2],t+a[i]);
            set_min(dp[pos][j2][1][k1+1][k2],t);
            if(j1||j2){
                set_min(dp[pos][j2][0][k1][k2],t);
                set_min(dp[pos][j2][0][k1][k2+1],t+a[i]);
            }
        }
    }
    LL ans=inf;
    _for(j1,0,2)_for(j2,0,2){
        if(j1||j2){
            _rep(k,0,m)
            set_min(ans,dp[pos][j1][j2][k][k]);
        }
```

```
    }
    enter(ans);
    return 0;
}
```

J. Histogram Sequence

题意

给定一个序列 h，其中二元组 $(a,b)(a\le b)$ 的权重为 $(b-a+1)\min (h[a\sim b])$。

将所有 $\frac {n(n-1)}2$ 个二元组按权重从小到大排序，求第 $L\sim R$ 个二元组。

题解

考虑构造 n 个集合，第 i 个集合维护所有 $\min (h[a\sim b])=h_i$ 的 (a,b) 二元组。

另外为了保证每个二元组恰属于一个集合，强制规定偏序：当 $h_i=h_j$ 时如果 $i\lt j$ 则 $\min(h_i,h_j)=h_i$。

单调栈求出每个集合对应的左边界 L_i 和右边界 R_i，对每个集合和给定 S，显然可以 $O(1)$ 计算出所有满足 $w(a,b)\lt S$ 的二元组个数。

于是二分答案求出第 L 个二元组的取值 S，然后将每个集合转化为类似迭代器的形式丢入优先队列，求出第 $L\sim R$ 个二元组。

时间复杂度 $O(n\log V+(R-L)\log n)$。

```
const int MAXN=3e5+5;
int h[MAXN],st[MAXN],vl[MAXN],vr[MAXN];
LL cal(LL n,LL len){
    len=min(n,len);
    return len*(n+n+1-len)/2;
}
LL check(int n,LL s){
    LL ans=0;
    _rep(i,1,n){
        LL len=(s-1)/h[i];
        ans+=cal(vr[i]-vl[i]+1,len)-cal(vr[i]-i,len)-cal(i-vl[i],len);
    }
    return ans;
}
int cal2(int n,int len){
    return max(n-len+1,0);
}
struct Node{
    int i,curlen;
```

```
    LL getS()const{
        return 1LL*curlen*h[i];
    }
    int count(){
        return cal2(vr[i]-vl[i]+1,curlen)-cal2(vr[i]-i,curlen)-cal2(i-
vl[i],curlen);
    }
    bool has_next(){
        return curlen<vr[i]-vl[i]+1;
    }
    bool operator < (const Node &b)const {
        return getS()>b.getS();
    }
};
int main(){
    int n=read_int();
    _rep(i,1,n)h[i]=read_int();
    LL ql=read_int(),qr=read_int();
    int tp=0;
    _rep(i,1,n){
        while(tp&&h[st[tp]]>h[i])tp--;
        vl[i]=st[tp]+1;
        st[++tp]=i;
    }
    tp=0;
    for(int i=n;i;i--){
        while(tp&&h[st[tp]]>=h[i])tp--;
        if(!tp)vr[i]=n;
        else
        vr[i]=st[tp]-1;
        st[++tp]=i;
    }
    LL lef=1,rig=MAXN*1e9,ans;
    while(lef<=rig){
        LL mid=lef+rig>>1;
        if(check(n,mid)<ql){
            ans=mid;
            lef=mid+1;
        }
        else
        rig=mid-1;
    }
    priority_queue<Node> q;
    _rep(i,1,n){
        if(1LL*(vr[i]-vl[i]+1)*h[i]>=ans){
            Node t=Node{i,(int)(ans/h[i])};
            if(t.has_next()){
                t.curlen++;
                q.push(t);
            }
```

```
        }
    }
    LL pos=check(n,ans+1);
    for(LL i=ql;i<=qr;i++){
        if(i>pos){
            Node t=q.top();q.pop();
            ans=t.getS();
            pos+=t.count();
            if(t.has_next()){
                t.curlen++;
                q.push(t);
            }
        }
        space(ans);
    }
    return 0;
}
```