比赛链接

补题情况

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

题解

B. xay loves monotonicity

题意

给定一个序列 A 和序列 B，其中 $0\le b_i\le 1$ 接下来三种操作：

1. $a_i\gets t$
2. 对 $l\le i\le r$，$b_i\gets b_i\oplus 1$
3. 给定 l,r，选取最长下标序列 $l\le i_1\le i_2\le \cdots i_k\le r$，满足 $a_{i_1}\le a_{i_2}\le\cdots \le a_{i_k}$，且对任意 $i_t\lt j\lt i_{t+1}$ 有 $a_j\lt a_{i_t}$。

对每个操作 3，输出 $b_{i_t}\neq b_{i_{t+1}}$ 的个数。

题解

设 $ma(L,R)=\max(a[L\sim R]),mb(L,R)$ 表示 $ma(L,R)$ 对应的 b_i，如果存在多个就取最右边的。

设 $\text{query}(L,R,p,q)$ 表示假如当前序列末尾对应 $a_i=p,b_i=q$ 时遍历区间 (L,R) 得到的答案。

于是，如果 $a_i\gt ma(L,M)$，则 $\text{query}(L,R,p,q)=\text{query}(M+1,R,p,q)$。

否则，有 $\text{query}(L,R,p,q)=\text{query}(L,M,p,q)+\text{query}(M+1,R,ma(L,M),mb(L,M))$。

建立线段树，每个区间维护 $\text{query}(M+1,R,ma(L,M),mb(L,M))$。

这样，对一个询问，如果该询问正好对应一个线段树区间，则查询复杂度 $O(\log n)$。

否则，将该询问拆分成 $O(\log n)$ 个线段树区间，串联查询计算答案，时间复杂度 $O(\log^2 n)$。

对与修改操作，修改完暴力询问更新 $\text{query}(M+1,R,ma(L,M),mb(L,M))$，由于这是线段树区间，

所以复杂度为 $O(\log n)$

所以修改的总复杂度也是 $O\left(\log^2 n\right)$总时间复杂度 $O\left(n\log n+q\log^2 n\right)$

ps. 比赛写了 $O(nq)$ 的假算法，居然过了。

```
const int MAXN=2e5+5;
int a[MAXN],b[MAXN];
int lef[MAXN<<2],rig[MAXN<<2],s[MAXN<<2],tag[MAXN<<2];
struct Node{
    int a,b;
}mv[MAXN<<2];
Node Max(Node L,Node R){
    if(L.a>R.a)
    return L;
    else
    return R;
}
void push_tag(int k){
    mv[k].b^=1;
    tag[k]^=1;
}
void push_down(int k){
    if(tag[k]){
        push_tag(k<<1);
        push_tag(k<<1|1);
        tag[k]=0;
    }
}
int query(int k,int a,int b){
    if(lef[k]==rig[k])
    return mv[k].a>=a&&mv[k].b!=b;
    push_down(k);
    if(a<=mv[k<<1].a)
    return query(k<<1,a,b)+s[k];
    else
    return query(k<<1|1,a,b);
}
void push_up(int k){
    mv[k]=Max(mv[k<<1],mv[k<<1|1]);
    s[k]=query(k<<1|1,mv[k<<1].a,mv[k<<1].b);
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R){
        mv[k]=Node{a[M],b[M]};
        return;
    }
    build(k<<1,L,M);
```

```
    build(k<<1|1,M+1,R);
    push_up(k);
}
Node query_max(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)
    return mv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=R)
    return query_max(k<<1,L,R);
    else if(mid<L)
    return query_max(k<<1|1,L,R);
    else
    return Max(query_max(k<<1,L,R),query_max(k<<1|1,L,R));
}
int query(int k,int L,int R,int a,int b){
    if(L<=lef[k]&&rig[k]<=R)
    return query(k,a,b);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=R)
    return query(k<<1,L,R,a,b);
    else if(mid<L)
    return query(k<<1|1,L,R,a,b);
    else{
        Node t=query_max(k<<1,L,R);
        if(a<=t.a)
        return query(k<<1,L,R,a,b)+query(k<<1|1,L,R,t.a,t.b);
        else
        return query(k<<1|1,L,R,a,b);
    }
}
void update1(int k,int pos,int v){
    if(lef[k]==rig[k]){
        mv[k].a=v;
        return;
    }
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=pos)
    update1(k<<1,pos,v);
    else
    update1(k<<1|1,pos,v);
    push_up(k);
}
void update2(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R){
        push_tag(k);
        return;
    }
    push_down(k);
```

```
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)
    update2(k<<1,L,R);
    if(mid<R)
    update2(k<<1|1,L,R);
    push_up(k);
}
int main(){
    int n=read_int();
    _rep(i,1,n)a[i]=read_int();
    _rep(i,1,n)b[i]=read_int();
    build(1,1,n);
    int q=read_int();
    while(q--){
        int opt=read_int(),t1=read_int(),t2=read_int();
        if(opt==1)
        update1(1,t1,t2);
        else if(opt==2)
        update2(1,t1,t2);
        else{
            if(t1==t2)
            enter(0);
            else{
                Node t=query_max(1,t1,t1);
                enter(query(1,t1+1,t2,t.a,t.b));
            }
        }
    }
    return 0;
}
```

J. xay loves Floyd

题意

给定一个有向图，初始时 $\text{dis}(u,u)=0,\text{dis}(u,v)=\infty(u\neq v)$

接下来给定若干条边 (u,v,w)使得 $\text{dis}(u,v)=w$询问以下两个程序最终结果中满足 $\text{dis}(u,v)$ 相同的 (u,v) 对数。

```
for k from 1 to n
  for i from 1 to n
    for j from 1 to n
      dis[i][j] <- min(dis[i][j], dis[i][k] + dis[k][j])
```

```
for i from 1 to n
  for j from 1 to n
    for k from 1 to n
```

```
      dis[i][j] <- min(dis[i][j], dis[i][k] + dis[k][j])
```

题解

首先 n 次单点源最短路算法 $O(nm\log m)$ 求出 dis 的真实值。

设 $ok(u,v)$ 表示 $\text{dis}(u,v)$ 是否为正确值，考虑第二个程序得到的 $\text{dis}(i,j)$ 正确的充要条件。

不难发现，只要 $i\to j$ 的最短路上有一点 k 满足 $ok(i,k)\And ok(k,j)$ 即可。

首先考虑找到所有满足条件的 k设 $\text{path}(u,v)$ 表示 $u\to v$ 上最短路的点集，于是有状态转移方程

$$ \text{path}(i,j)=\bigcup_{\text{dis}(i,k)+w(k,j)=\text{dis}(i,j)}\text{path}(i,k) $$

对固定的 i考虑将 j 按 $\text{dis}(i,j)$ 从小到大排序后用 bitset 加速上述转移。

然后按 $1\sim n$ 顺序枚举 j于是有 $\text{ok}(i,j)=\sum_{k=1}^n ok(i,k)\And ok(k,j)\And (k\in \text{path}(i,j))$

用两种 bitset 维护 $\text{ok}(i,\ast),\text{ok}(\ast,i)$上述转移也可以用 bitset 加速。总时间复杂度 $O\left(nm\log m+\frac {n^2m}w\right)$

```
const int MAXN=2e3+5,MAXM=5e3+5,inf=1e9;
struct Edge{
    int to,w,next;
}edge[MAXM];
int head[MAXN],edge_cnt;
void Insert(int u,int v,int w){
    edge[++edge_cnt]=Edge{v,w,head[u]};
    head[u]=edge_cnt;
}
namespace DJ{
    bool vis[MAXN];
    void solve(int n,int s,int *dis){
        _rep(i,1,n){
            dis[i]=inf;
            vis[i]=false;
        }
        dis[s]=0;
        priority_queue<pair<int,int> > q;
        q.push(make_pair(0,s));
        while(!q.empty()){
            int u=q.top().second;q.pop();
            if(vis[u])continue;
            vis[u]=true;
            for(int i=head[u];i;i=edge[i].next){
                int v=edge[i].to;
                if(dis[v]>dis[u]+edge[i].w){
```

```
                    dis[v]=dis[u]+edge[i].w;
                    q.push(make_pair(-dis[v],v));
                }
            }
        }
    }
}
int dis[MAXN][MAXN],d0[MAXN][MAXN];
bitset<MAXN> ok1[MAXN],ok2[MAXN],path[MAXN];
int main(){
    int n=read_int(),m=read_int();
    _rep(i,1,n)_rep(j,1,n)d0[i][j]=inf;
    _rep(i,1,n)d0[i][i]=0;
    while(m--){
        int u=read_int(),v=read_int(),w=read_int();
        d0[u][v]=w;
        Insert(u,v,w);
    }
    _rep(i,1,n)
    DJ::solve(n,i,dis[i]);
    _rep(i,1,n)_rep(j,1,n){
        if(dis[i][j]==d0[i][j])
        ok1[i][j]=ok2[j][i]=true;
    }
    _rep(u,1,n){
        vector<pair<int,int> >vec;
        _rep(v,1,n){
            vec.push_back(make_pair(dis[u][v],v));
            path[v].reset();
        }
        sort(vec.begin(),vec.end());
        for(pair<int,int> p:vec){
            int v=p.second;
            path[v][v]=true;
            for(int i=head[v];i;i=edge[i].next){
                int t=edge[i].to;
                if(dis[u][v]+edge[i].w==dis[u][t])
                path[t]|=path[v];
            }
        }
        _rep(v,1,n){
            if((ok1[u]&ok2[v]&path[v]).any())
            ok1[u][v]=ok2[v][u]=true;
        }
    }
    int ans=0;
    _rep(i,1,n)
    ans+=ok1[i].count();
    enter(ans);
    return 0;
```

```
}
```