CCPC Wannafly Camp Day1

比赛链接

E. 树与路径

题意

给定一个有根树，定义任意两点 u,v 的路径为 $\text{dis}(u,p)\text{dis}(v,p)(p=\text{LCP}(u,v))$

给定 m 条路径，询问以每个结点为跟时所有路径的权值和。

题解

先考虑每条路径对以路径上每个结点为根节点时答案的贡献，记 $\text{dis}(u,v)=L$

对 $u\to p$ 上路径的每个结点，不难得出该路径对某个结点 i 的贡献为 $(d_u-d_i)(L-d_u+d_i)$

而对路径上的其他结点，该路径对该点的贡献等于该路径对路径上距离该点最近的结点的贡献。

于是可以树上差分维护该路径对所有点的贡献，其中路径 $u\to p(\text{不含} p)$ 的点的差分值为 $2(d_u-d_i)+1-L$同理 $v\to p(\text{不含} p)$ 也类似。

而其他点差分值为 0。发现差分值为等差数列，于是考虑利用子树和维护路径 $u\to p(\text{不含} p)$ 的差分值。

注意子树和维护最后结点 p 的差分值为 $2(d_u-d_p)+1-L+2(d_v-d_p)+1-L=2$而 p 的实际差分值应该为 0，需要修正。

总时间复杂度 $O(n+m\log n)$

```
const int MAXN=3e5+5;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt]=Edge{v,head[u]};
    head[u]=edge_cnt;
}
namespace LCA{
    int d[MAXN],sz[MAXN],f[MAXN];
    int h_son[MAXN],mson[MAXN],p[MAXN];
    void dfs_1(int u,int fa,int depth){
        sz[u]=1;f[u]=fa;d[u]=depth;mson[u]=0;
        for(int i=head[u];i;i=edge[i].next){
```

```
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs_1(v,u,depth+1);
            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==f[u]||v==h_son[u])
            continue;
            dfs_2(v,v);
        }
    }
    void Init(int root){dfs_1(root,0,0);dfs_2(root,root);}
    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 dp1[MAXN],dp2[MAXN];
void dfs1(int u,int fa){
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dfs1(v,u);
        dp1[u]+=dp1[v];
        dp2[u]+=dp2[v]+dp1[v];
    }
}
void dfs2(int u,int fa){
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(v==fa)continue;
        dp2[v]+=dp2[u];
        dfs2(v,u);
    }
}
int main()
{
    int n=read_int(),m=read_int();
    _for(i,1,n){
```

```
        int u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    LCA::Init(1);
    LL s=0;
    while(m--){
        int
u=read_int(),v=read_int(),p=LCA::query_lca(u,v),len=LCA::d[u]+LCA::d[v]-2*L
CA::d[p];
        dp1[u]+=2,dp1[v]+=2,dp1[p]-=4;
        dp2[u]+=1-len,dp2[v]+=1-len,dp2[p]-=2;
        s+=1LL*(LCA::d[u]-LCA::d[p])*(LCA::d[v]-LCA::d[p]);
    }
    dfs1(1,0);
    dp2[1]+=s;
    dfs2(1,0);
    _rep(i,1,n)enter(dp2[i]);
    return 0;
}
```

I. K 小数查询

题意

给定一个长度为 n 的序列，接下来 q 个操作：

1. 选取区间 $[l,r]$$a_i=\min(a_i,v)(l\le i\le r)$
2. 询问区间 $[l,r]$ 的第 k 小元素

题解

建立线段树套权值线段树。对于查询操作，考虑二分答案，然后查询区间中小于二分值的数个数判定答案合法性，时间复杂度 $O(\log^3 n)$

对于修改操作，考虑在线段树中找到对应区间，然后暴力修改该区间及其所有祖先结点对应的权值线段树。

修改完成后同样打上懒标记，必要时下放标记。

显然每次修改复杂度为 $O(k\log n)$其中 k 为该次修改删除的权值线段树的叶子数。

由于初始时仅有 $O(n\log n)$ 个权值线段树的叶子结点，于是修改操作的总时间复杂度为 $O(n\log^2 n)$

于是总时间复杂度 $O(n\log^3 n)$总空间复杂度 $O(n\log^2 n)$

```
const int MAXN=8e4+5,MAXM=400,Inf=1e9;
int a[MAXN],root[MAXN<<2],lef[MAXN<<2],rig[MAXN<<2],lazy[MAXN<<2];
```

```
int n,sz,s[MAXN*MAXM],ch[MAXN*MAXM][2];
void update_1D(int &k,int pos,int v,int lef=1,int rig=n){
    if(!k)k=++sz;
    s[k]+=v;
    if(lef==rig)return;
    int mid=lef+rig>>1;
    if(pos<=mid)
    update_1D(ch[k][0],pos,v,lef,mid);
    else
    update_1D(ch[k][1],pos,v,mid+1,rig);
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy[k]=Inf;
    _rep(i,L,R)update_1D(root[k],a[i],1);
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
vector<pair<int,int> >del;
void dfs(int k,int pos,int lef=1,int rig=n){
    if(!s[k])return;
    if(lef==rig){
        del.push_back(make_pair(lef,s[k]));
        return;
    }
    int mid=lef+rig>>1;
    if(pos<mid)dfs(ch[k][0],pos,lef,mid);
    dfs(ch[k][1],pos,mid+1,rig);
}
void dfs2(int k,int pos,int lef=1,int rig=n){
    if(!s[k])return;
    if(lef==rig){
        s[k]=0;
        return;
    }
    int mid=lef+rig>>1;
    if(pos<mid)dfs2(ch[k][0],pos,lef,mid);
    dfs2(ch[k][1],pos,mid+1,rig);
    s[k]=s[ch[k][0]]+s[ch[k][1]];
}
void update_v(int k,int v){
    if(s[root[k]]!=rig[k]-lef[k]+1)
    update_1D(root[k],v,rig[k]-lef[k]+1-s[root[k]]);
}
void push_down(int k){
    if(lazy[k]!=Inf){
        dfs2(root[k<<1],lazy[k]);
        update_v(k<<1,lazy[k]);
        lazy[k<<1]=min(lazy[k<<1],lazy[k]);
```

```
        dfs2(root[k<<1|1],lazy[k]);
        update_v(k<<1|1,lazy[k]);
        lazy[k<<1|1]=min(lazy[k<<1|1],lazy[k]);
        lazy[k]=Inf;
    }
}
void update_2D(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R){
        lazy[k]=min(lazy[k],v);
        del.clear();
        dfs(root[k],v);
        while(k){
            _for(i,0,del.size())
            update_1D(root[k],del[i].first,-del[i].second);
            update_v(k,v);
            k>>=1;
        }
        return;
    }
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_2D(k<<1,L,R,v);
    if(mid<R)update_2D(k<<1|1,L,R,v);
}
int query_1D(int k,int L,int R,int lef=1,int rig=n){
    if(!k)return 0;
    if(L<=lef&&rig<=R)return s[k];
    int mid=lef+rig>>1;
    if(mid>=R)return query_1D(ch[k][0],L,R,lef,mid);
    else if(mid<L)return query_1D(ch[k][1],L,R,mid+1,rig);
    return query_1D(ch[k][0],L,R,lef,mid)+query_1D(ch[k][1],L,R,mid+1,rig);
}
int query_2D(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return query_1D(root[k],1,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=R)return query_2D(k<<1,L,R,v);
    else if(mid<L)return query_2D(k<<1|1,L,R,v);
    return query_2D(k<<1,L,R,v)+query_2D(k<<1|1,L,R,v);
}
int main()
{
    n=read_int();
    int m=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    while(m--){
        int opt=read_int(),l=read_int(),r=read_int(),v=read_int();
        if(opt==1){
            if(v>=n)continue;
```

```
            update_2D(1,l,r,v);
        }
        else{
            int lef=1,rig=n,mid,ans=-1;
            while(lef<=rig){
                mid=lef+rig>>1;
                if(query_2D(1,l,r,mid)>=v){
                    ans=mid;
                    rig=mid-1;
                }
                else
                lef=mid+1;
            }
            enter(ans);
        }
    }
    return 0;
}
```