# 重构树

## 算法简介

一种用于解决从任意询问点出发在边权或点权等限制下能到达的点集的信息维护问题的算法。

## 算法例题

### 例题一

洛谷p4197

#### 题意

给定 $n$ 个点和 $m$ 条边，每个点给定一个点权，每条边给定一个边权。接下来 $q$ 个询问。

每次询问从点 $v$ 开始不经过边权超过 $x$ 的边可以走到的点中第 $k$ 大的权值。

#### 题解

考虑 $\text{Kruskal}$ 算法重构树，即求取最小生成树过程中将加边操作改为生成一个新结点同时将新结点作为两个连通块的根节点的祖先。

将重构树上的新结点的点权设为该结点代表边的边权，原图中结点点权设为 $0$。于是重构树上的点权从叶子结点到根节点单增。

于是每次询问从原图某结点开始可以到达的边权不超过 $x$ 的结点等价于询问该结点的权值不超过 $x$ 的最远的祖先结点的子树。

于是倍增求出满足条件的祖先结点，最后利用 $\text{dfs}$ 序和主席树处理第 $k$ 大询问即可。时间复杂度 $O((n+q)\log n+m\log m)$

另外这题可以以离线询问，将询问和边然后按权值排序，然后通过线段树合并处理加边操作和询问操作。

```
#define rch(k) node[node[k].ch[1]]
const int MAXN=1e5+5,MAXM=5e5+5,MAXL=80;
struct Node{
    int s,ch[2];
}node[MAXN*MAXL];
int root[MAXN<<1],n,seg_n,node_cnt;
void update(int &k,int p,int pos,int lef=1,int rig=seg_n){
    node[k=++node_cnt]=node[p];
    node[k].s++;
    if(lef==rig)return;
    int mid=lef+rig>>1;
```

```
    if(pos<=mid)
    update(node[k].ch[0],node[p].ch[0],pos,lef,mid);
    else
    update(node[k].ch[1],node[p].ch[1],pos,mid+1,rig);
}
int query(int k1,int k2,int rk,int lef=1,int rig=seg_n){
    int mid=lef+rig>>1;
    if(lef==rig)return mid;
    if(rk<=rch(k1).s-rch(k2).s)
    return query(node[k1].ch[1],node[k2].ch[1],rk,mid+1,rig);
    else
    return query(node[k1].ch[0],node[k2].ch[0],rk-
rch(k1).s+rch(k2).s,lef,mid);
}
struct Edge{
    int u,v,w;
    bool operator < (const Edge &b)const{
        return w<b.w;
    }
}edge[MAXM];
int a[MAXN],b[MAXN],p[MAXN<<1],ch[MAXN<<1][2],dfn[MAXN<<1][2],dfs_t;
namespace LCA{
    const int MAXN=2e5+5;
    int d[MAXN],anc[MAXN][MAXL],w[MAXN],log2[MAXN];
    void get_log(int n){
        int log=0;
        while((1<<log)<n){
            for(int i=1<<log,j=min(1<<(log+1),n);i<j;i++)
            log2[i]=log;
            log++;
        }
    }
    void init(int n){
        get_log(n);
        for(int j=1;(1<<j)<n;j++){
            _rep(i,1,n){
                if(!anc[i][j-1])
                continue;
                anc[i][j]=anc[anc[i][j-1]][j-1];
            }
        }
    }
    int query(int p,int k){
        for(int i=log2[d[p]];i>=0;i--){
            if(anc[p][i]&&w[anc[p][i]]<=k)
            p=anc[p][i];
        }
        return p;
    }
};
```

```
int Find(int x){return x==p[x]?x:p[x]=Find(p[x]);}
void dfs(int u,int fa,int dep){
    if(!u)return;
    dfn[u][0]=++dfs_t;LCA::d[u]=dep;
    if(u<=n)
    update(root[dfs_t],root[dfs_t-1],a[u]);
    else
    root[dfs_t]=root[dfs_t-1];
    dfs(ch[u][0],u,dep+1);
    dfs(ch[u][1],u,dep+1);
    dfn[u][1]=dfs_t;
}
int main()
{
    n=read_int();
    int m=read_int(),q=read_int();
    _rep(i,1,n)a[i]=b[i]=read_int();
    sort(b+1,b+n+1);
    seg_n=unique(b+1,b+n+1)-b;
    _rep(i,1,n)a[i]=lower_bound(b+1,b+seg_n,a[i])-b;
    _for(i,0,m){
        edge[i].u=read_int();
        edge[i].v=read_int();
        edge[i].w=read_int();
    }
    sort(edge,edge+m);
    int cur=n+1;
    _for(i,1,2*n)p[i]=i;
    _for(i,0,m){
        int x=Find(edge[i].u),y=Find(edge[i].v);
        if(x!=y){
            p[x]=p[y]=cur;
            ch[cur][0]=x,ch[cur][1]=y;
            LCA::anc[x][0]=LCA::anc[y][0]=cur,LCA::w[cur]=edge[i].w;
            cur++;
        }
    }
    _for(i,1,cur){
        if(!root[Find(i)])
        dfs(Find(i),0,0);
    }
    LCA::init(cur-1);
    while(q--){
        int v=read_int(),x=read_int(),k=read_int();
        int rt=LCA::query(v,x);
        if(node[root[dfn[rt][1]]].s-node[root[dfn[rt][0]-1]].s<k)
        enter(-1);
        else
        enter(b[query(root[dfn[rt][1]],root[dfn[rt][0]-1],k)]);
    }
    return 0;
```

```
}
```