线段树合并

算法简介

一种合并多个线段树(一般为权值线段树)的算法，主要用于解决染色问题，时空间复杂度 $O(m\log n)$

算法思想

更新线段树时动态开点，合并时如果遇到叶子节点或空结点就直接 return否则跑子树。

关于空间复杂度，每次动态开点 $O(\log n)$所以总空间复杂度 $O(m\log n)$注意线段树是四倍空间。

关于时间复杂度，每次合并时间复杂度为两棵线段树的重叠部分的结点数，所以不会超过较小的那棵线段树的结点数。

所以合并操作的总时间复杂度等于动态开点总数，即 $O(m\log n)$

代码模板

```
const int MAXS=MAXN*60;
int root[MAXN],tot;
struct Node{
    int max_cnt,ans;//自己需要维护的信息
    int ch[2];
}node[MAXS];
void push_up(int k){//自定义
    if(node[node[k].ch[0]].max_cnt>=node[node[k].ch[1]].max_cnt)
node[k].max_cnt=node[node[k].ch[0]].max_cnt,node[k].ans=node[node[k].ch[0]]
.ans;
    else
node[k].max_cnt=node[node[k].ch[1]].max_cnt,node[k].ans=node[node[k].ch[1]]
.ans;
}
void update(int &k,int lef,int rig,int pos,int v){
    if(!k) k=++tot;
    if(lef==rig) return node[k].max_cnt+=v,node[k].ans=pos,void();
    int mid=lef+rig>>1;
    if(pos>mid)
    update(node[k].ch[1],mid+1,rig,pos,v);
    else
    update(node[k].ch[0],lef,mid,pos,v);
    push_up(k);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    if(lef==rig) return node[k1].max_cnt+=node[k2].max_cnt,void();
```

```
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    push_up(k1);
}
```

算法练习

习题一

洛谷p4556

题意

给定一棵 n 个节点的数，m 个操作。

每个操作三个参数 x,y,z，表示给结点 x 到 y 的树链上的每个点打上一个 z 号标记。

经过所有操作后输出每个结点被打上的最多的标记的编号(满足条件的标记存在多个时输出编号最小的)，如果该结点未被标记过，输出 0。

题解 1

离散化处理标记编号，防止 MLE。

每个结点用一棵权值线段树维护该结点的所有标记状态，树上差分打标记，最后从叶子结点开始向上合并即可。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
```

```cpp
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5,MAXM=20;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
struct LCA{
    int d[MAXN],anc[MAXN][MAXM],log2[MAXN];
    void dfs(int u,int fa,int dep){
        anc[u][0]=fa,d[u]=dep;
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
            if(v==fa)
            continue;
            dfs(v,u,dep+1);
        }
    }
    void build(int root,int n){
```

```
        log2[1]=0;
        _rep(i,2,n)
        log2[i]=log2[i>>1]+1;
        dfs(root,-1,1);
        _rep(i,1,n){//下标从1开始
            for(int j=1;(1<<j)<n;j++)
            anc[i][j]=-1;
        }
        for(int j=1;(1<<j)<n;j++){
            _rep(i,1,n){
                if(anc[i][j-1]==-1)
                continue;
                anc[i][j]=anc[anc[i][j-1]][j-1];
            }
        }
    }
    int query(int p,int q){
        if(d[p]<d[q])
        swap(p,q);
        for(int i=log2[d[p]];i>=0;i--){
            if(d[p]-(1<<i)>=d[q])
            p=anc[p][i];
        }
        if(p==q)
        return p;
        for(int i=log2[d[p]];i>=0;i--){
            if(anc[p][i]!=-1&&anc[p][i]!=anc[q][i]){
                p=anc[p][i],q=anc[q][i];
            }
        }
        return anc[p][0];
    }
}lca;
const int MAXS=MAXN*60;
int root[MAXN],tot;
struct Node{
    int max_cnt,ans;
    int ch[2];
}node[MAXS];
void push_up(int k){
    if(node[node[k].ch[0]].max_cnt>=node[node[k].ch[1]].max_cnt)
node[k].max_cnt=node[node[k].ch[0]].max_cnt,node[k].ans=node[node[k].ch[0]]
.ans;
    else
node[k].max_cnt=node[node[k].ch[1]].max_cnt,node[k].ans=node[node[k].ch[1]]
.ans;
}
void update(int &k,int lef,int rig,int pos,int v){
    if(!k) k=++tot;
    if(lef==rig) return node[k].max_cnt+=v,node[k].ans=pos,void();
```

```
    int mid=lef+rig>>1;
    if(pos>mid)
    update(node[k].ch[1],mid+1,rig,pos,v);
    else
    update(node[k].ch[0],lef,mid,pos,v);
    push_up(k);
}
void Merge(int &k1,int k2,int lef,int rig){
    if(!k1||!k2) return k1|=k2,void();
    if(lef==rig) return node[k1].max_cnt+=node[k2].max_cnt,void();
    int mid=lef+rig>>1;
    Merge(node[k1].ch[0],node[k2].ch[0],lef,mid);
    Merge(node[k1].ch[1],node[k2].ch[1],mid+1,rig);
    push_up(k1);
}
int X[MAXN],Y[MAXN],Z[MAXN],b[MAXN],ans[MAXN],Max_z;
void dfs(int u){
    for(int i=head[u];i;i=edge[i].next){
        int v=edge[i].to;
        if(lca.anc[u][0]==v)
        continue;
        dfs(v);
        Merge(root[u],root[v],1,Max_z);
    }
    ans[u]=node[root[u]].max_cnt>0?node[root[u]].ans:0;
}
int main()
{
    int n=read_int(),q=read_int(),u,v,p;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    lca.build(1,n);
    _for(i,0,q)
    X[i]=read_int(),Y[i]=read_int(),Z[i]=read_int();
    memcpy(b,Z,sizeof(Z));
    sort(b,b+q);
    Max_z=unique(b,b+q)-b;
    _for(i,0,q){
        Z[i]=lower_bound(b,b+Max_z,Z[i])-b+1;
update(root[X[i]],1,Max_z,Z[i],1);update(root[Y[i]],1,Max_z,Z[i],1);
        p=lca.query(X[i],Y[i]);
        update(root[p],1,Max_z,Z[i],-1);
        p=lca.anc[p][0];
        if(p!=-1)
        update(root[p],1,Max_z,Z[i],-1);
    }
    dfs(1);
    _rep(i,1,n)
```

```
    if(ans[i])
    enter(b[ans[i]-1]);
    else
    enter(0);
    return 0;
}
```

题解 2

考虑树剖，将树上问题转换为区间问题，更新路径时只打差分打标记。

最后询问时只建一棵权值线段树，依次释放区间上每个位置的标记，维护标记前缀和、答案。

时间复杂度 $O(m\log^2 n)$空间复杂度 $O(m\log n)$但常数远小于线段树合并。

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <string>
#include <sstream>
#include <cstring>
#include <cctype>
#include <cmath>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <ctime>
#include <cassert>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
#define mem(a,b) memset(a,b,sizeof(a))
using namespace std;
typedef long long LL;
inline int read_int(){
    int t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline LL read_LL(){
    LL t=0;bool sign=false;char c=getchar();
    while(!isdigit(c)){sign|=c=='-';c=getchar();}
    while(isdigit(c)){t=(t<<1)+(t<<3)+(c&15);c=getchar();}
    return sign?-t:t;
}
inline char get_char(){
```

```
    char c=getchar();
    while(c==' '||c=='\n'||c=='\r')c=getchar();
    return c;
}
inline void write(LL x){
    register char c[21],len=0;
    if(!x)return putchar('0'),void();
    if(x<0)x=-x,putchar('-');
    while(x)c[++len]=x%10,x/=10;
    while(len)putchar(c[len--]+48);
}
inline void space(LL x){write(x),putchar(' ');}
inline void enter(LL x){write(x),putchar('\n');}
const int MAXN=1e5+5,MAXM=20;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],edge_cnt;
void Insert(int u,int v){
    edge[++edge_cnt].to=v;
    edge[edge_cnt].next=head[u];
    head[u]=edge_cnt;
}
struct lazy_tag{
    int v,next;
}lazy[MAXN*MAXM];
int head_2[MAXN],lazy_cnt;
void Insert_2(int u,int v){
    lazy[++lazy_cnt].v=v;
    lazy[lazy_cnt].next=head_2[u];
    head_2[u]=lazy_cnt;
}
int Max_cnt[MAXN<<2],Ans[MAXN<<2],lef[MAXN<<2],rig[MAXN<<2];
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    int M=L+R>>1;
    if(L==R)return Ans[k]=M,void();
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
}
void push_up(int k){
    if(Max_cnt[k<<1]>=Max_cnt[k<<1|1]){
        Max_cnt[k]=Max_cnt[k<<1];
        Ans[k]=Ans[k<<1];
    }
    else{
        Max_cnt[k]=Max_cnt[k<<1|1];
        Ans[k]=Ans[k<<1|1];
    }
}
void update(int k,int pos,int v){
```

```
    if(lef[k]==rig[k])
    return Max_cnt[k]+=v,void();
    int mid=lef[k]+rig[k]>>1;
    if(pos<=mid)
    update(k<<1,pos,v);
    else
    update(k<<1|1,pos,v);
    push_up(k);
}
int d[MAXN],sz[MAXN],f[MAXN],dfs_id[MAXN],inv_id[MAXN],dfs_t;
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){
    dfs_id[u]=++dfs_t;inv_id[dfs_t]=u;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 update_path(int u,int v,int w){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        Insert_2(dfs_id[p[u]],w);
        Insert_2(dfs_id[u]+1,-w);
        u=f[p[u]];
    }
    if(d[u]>d[v])
    swap(u,v);
    Insert_2(dfs_id[u],w);
    Insert_2(dfs_id[v]+1,-w);
}
void update_node(int u){
```

```
    for(int i=head_2[u];i;i=lazy[i].next){
        if(lazy[i].v>0)
        update(1,lazy[i].v,1);
        else
        update(1,-lazy[i].v,-1);
    }
}
int X[MAXN],Y[MAXN],Z[MAXN],b[MAXN],ans[MAXN];
int main()
{
    int n=read_int(),q=read_int(),u,v,w;
    _for(i,1,n){
        u=read_int(),v=read_int();
        Insert(u,v);
        Insert(v,u);
    }
    _for(i,0,q)
    X[i]=read_int(),Y[i]=read_int(),Z[i]=read_int();
    memcpy(b,Z,sizeof(Z));
    sort(b,b+q);
    int Max_z=unique(b,b+q)-b;
    dfs_1(1,-1,0);
    dfs_2(1,1);
    build(1,1,Max_z);
    _for(i,0,q)
    update_path(X[i],Y[i],lower_bound(b,b+Max_z,Z[i])-b+1);
    _rep(i,1,n){
        update_node(i);
        ans[inv_id[i]]=Max_cnt[1]>0?b[Ans[1]-1]:0;
    }
    _rep(i,1,n)
    enter(ans[i]);
    return 0;
}
```