LCT

算法简介

LCT 是一种动态维护森林连通性、路径信息的数据结构，时间复杂度为 $O\left(n\log n\right)$。

算法思想

LCT 将树上路径分为实链和虚链，每个非叶结点仅向他的一个儿子结点连实边，其余儿子连虚边。

每个实链用一棵 splay 保存，同一棵树的 splay 之间靠虚边连接，每个 splay 维护一个深度递增的结点序列。

每棵 splay 树的根结点的父结点(注意，不是原树的父结点)设为这个 splay 深度最小的结点的在原树中的父节点。

LCT 的核心操作为 $\text{access}(x)$。

$\text{access}(x)$ 的目的是将 x 结点到根结点的路径变为一条实链，便于后续操作，方法很简单，不断向上修改父子关系即可。

我们还想得到两个非根结点的路径信息，我们可以先将其中一个结点变为根结点，再使用 access 操作。

将一个结点变为根结点即为 $\text{makeroot}(x)$ 操作，方法为先 $\text{access}(x)$ 再颠倒这条实链。

颠倒实链可以考虑 $\text{splay}(x)$ 。x 是 splay 中深度最小的点，无右儿子。

因此交换 x 左右儿子。 x 无左儿子，成为深度最大的点，最后打上懒标记即可。

考虑到 LCT 维护的是森林，为了判断连通性，我们还需要 $\text{findroot}(x)$ 即得到结点 x 所在原树的根结点。

方法为先 $\text{access}(x)$ 便可以得到结点 x 到根结点的路径。

考虑到原树的根结点为深度最小的点，我们只需要 $\text{splay}(x)$ 然后从结点 x 出发不断访问右节点即可，但要注意下放懒标记。

有了这些基本操作，便可以实现树上的连边、删边、两点间的路径信息维护等操作了。

代码模板

洛谷p3690

给定 n 个点和权值，接下来 m 个操作。

1. 询问 x 到 y 路径的权值的异或和，保证 x 与 y 已经连通
2. 连接 x 与 y 若 x 与 y 已经连通，则无视这个操作

3. 删除 x 与 y 的连边，若 x 与 y 无连边，则无视这个操作
4. 把结点 x 的权值改为 y

```
const int MAXN=1e5+5;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],w[MAXN],s[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    void build(int *a,int n){
        _rep(i,1,n){
            ch[i][0]=ch[i][1]=fa[i]=0;
            s[i]=w[i]=a[i];
            flip[i]=false;
        }
    }
    void push_up(int k){
        s[k]=s[lch(k)]^s[rch(k)]^w[k];
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
```

```
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            ch[k][1]=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u)fa[u]=v;
    }
    void cut(int u,int v){
        split(u,v);
        if(ch[v][0]==u&&ch[u][1]==0){
            ch[v][0]=fa[u]=0;
            push_up(v);
        }
    }
    void change(int k,int v){
        access(k);splay(k);
        w[k]=v;
        push_up(k);
    }
}LCT;
int a[MAXN];
int main()
{
    int n=read_int(),m=read_int(),opt,u,v;
    _rep(i,1,n)
```

```
    a[i]=read_int();
    LCT.build(a,n);
    _rep(i,1,m){
        opt=read_int(),u=read_int(),v=read_int();
        if(opt==0){
            LCT.split(u,v);
            enter(LCT.s[v]);
        }
        else if(opt==1)
        LCT.link(u,v);
        else if(opt==2)
        LCT.cut(u,v);
        else
        LCT.change(u,v);
    }
    return 0;
}
```

代码练习

路径信息维护

例题一

洛谷p1501

给定一棵 n 个结点的树，每个结点初始权值为 1，接下来 q 个操作：

1. 将 u 到 v 路径上所有点权值加上 c
2. 删除 u_1 与 v_1 的连边，添加 u_2 与 v_2 的连边，保证操作后仍然是一棵树
3. 将 u 到 v 路径上所有点权值乘上 c
4. 查询 u 到 v 路径上权值和

一道简单LCT练手题，多加几个懒标记即可。

```
const int MAXN=1e5+5,Mod=51061;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],sz[MAXN],w[MAXN],s[MAXN];
    int mul_tag[MAXN],add_tag[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    void node_init(int k,int v){
        ch[k][0]=ch[k][1]=fa[k]=0;
        sz[k]=1;
```

```
        w[k]=s[k]=v;
        mul_tag[k]=1,add_tag[k]=0,flip[k]=false;
    }
    void push_up(int k){
        s[k]=(s[lch(k)]+s[rch(k)]+w[k])%Mod;
        sz[k]=sz[lch(k)]+sz[rch(k)]+1;
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_mul(int k,int v){
        s[k]=1LL*s[k]*v%Mod;
        w[k]=1LL*w[k]*v%Mod;
        add_tag[k]=1LL*add_tag[k]*v%Mod;
        mul_tag[k]=1LL*mul_tag[k]*v%Mod;
    }
    void push_add(int k,int v){
        s[k]=(s[k]+1LL*sz[k]*v)%Mod;
        w[k]=(w[k]+v)%Mod;
        add_tag[k]=(add_tag[k]+v)%Mod;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
        if(mul_tag[k]!=1){
            push_mul(lch(k),mul_tag[k]);
            push_mul(rch(k),mul_tag[k]);
            mul_tag[k]=1;
        }
        if(add_tag[k]){
            push_add(lch(k),add_tag[k]);
            push_add(rch(k),add_tag[k]);
            add_tag[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
```

```
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            ch[k][1]=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u)fa[u]=v;
    }
    void cut(int u,int v){
        split(u,v);
        if(ch[v][0]==u&&ch[u][1]==0){
            ch[v][0]=fa[u]=0;
            push_up(v);
        }
    }
    void update_add(int u,int v,int val){
        split(u,v);
        push_add(v,val);
```

```
    }
    void update_mul(int u,int v,int val){
        split(u,v);
        push_mul(v,val);
    }
    int query_sum(int u,int v){
        split(u,v);
        return s[v];
    }
}LCT;
int main()
{
    int n=read_int(),m=read_int();
    _rep(i,1,n)
    LCT.node_init(i,1);
    _for(i,1,n){
        int u=read_int(),v=read_int();
        LCT.link(u,v);
    }
    while(m--){
        char opt=get_char();
        int u=read_int(),v=read_int();
        if(opt=='+')
        LCT.update_add(u,v,read_int());
        else if(opt=='-'){
            LCT.cut(u,v);
            u=read_int(),v=read_int();
            LCT.link(u,v);
        }
        else if(opt=='*')
        LCT.update_mul(u,v,read_int());
        else
        enter(LCT.query_sum(u,v));
    }
    return 0;
}
```

例题二

洛谷p3703

给定一棵以 1 为根节点的有根树，初始时每个节点的颜色互不相同，定义路径的权值为路径节点的颜色种数。接下来 m 个操作：

1. 将根节点到 u 的路径染成一个从未出现的颜色
2. 查询 $u\to v$ 路径权值
3. 查询所有以 u 为根的子树节点中到 1 节点的路径的最大权值

考虑维护每个点 u 到根节点的路径权值，记为 $\text{dis}(u)$，不难发现操作 2 等效于查询 $\text{dis}(u)+\text{dis}(v)-2\text{dis}(\text{lca}(u,v))+1$。

对于操作 3，等价于查询以 u 为根的子树的最大权值。

接下来重点考虑操作 1，联想到 LCA 的 access 操作。

假如一开始所有连边均为虚边，不难发现 $\text{dis}(u)$ 等于 u 到根节点的虚边数 $+1$，然后操作 1 正好等价于 $\text{access}(u)$ 操作。

然后考虑更新答案，发现 $\text{access}(u)$ 过程中将 $u\to v$ 的实边变成虚边只需要将 v 子树所有点权值加一即可，相对的虚边变实边对应子树点权减一。

考虑再建一棵线段树维护，另外注意 $\text{access}(u)$ 过程中涉及的节点 v 对应的是 splay 子树的根节点，不是真正需要的深度最小的点。

为了找到深度最小的节点可以考虑利用 splay 的 push_up 维护。总时间复杂度 $O(n\log n)$

ps. 树剖也可做，但比较复杂，需要魔改，可以参考这篇 题解

```
const int MAXN=1e5+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 Tree{
    int d[MAXN],sz[MAXN],f[MAXN],dfn[MAXN],dfs_t;
    int hson[MAXN],mson[MAXN],p[MAXN],dfw[MAXN];
    void dfs1(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;
            dfs1(v,u,depth+1);
            sz[u]+=sz[v];
            if(sz[v]>mson[u]){
                hson[u]=v;
                mson[u]=sz[v];
            }
        }
    }
    void dfs2(int u,int top){
        dfn[u]=++dfs_t,p[u]=top;
        dfw[dfs_t]=d[u];
        if(mson[u])
        dfs2(hson[u],top);
        for(int i=head[u];i;i=edge[i].next){
            int v=edge[i].to;
```

```
            if(v==f[u]||v==hson[u])continue;
            dfs2(v,v);
        }
    }
    int lef[MAXN<<2],rig[MAXN<<2],s[MAXN<<2],tag[MAXN<<2];
    void push_up(int k){
        s[k]=max(s[k<<1],s[k<<1|1]);
    }
    void push_add(int k,int v){
        s[k]+=v;
        tag[k]+=v;
    }
    void push_down(int k){
        if(tag[k]){
            push_add(k<<1,tag[k]);
            push_add(k<<1|1,tag[k]);
            tag[k]=0;
        }
    }
    void build(int k,int L,int R){
        lef[k]=L,rig[k]=R;
        int M=L+R>>1;
        if(L==R){
            s[k]=dfw[M];
            return;
        }
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
        push_up(k);
    }
    void init(int n){
        dfs1(1,0,1);
        dfs2(1,1);
        build(1,1,n);
    }
    void update(int k,int L,int R,int v){
        if(L<=lef[k]&&rig[k]<=R){
            push_add(k,v);
            return;
        }
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=L)
        update(k<<1,L,R,v);
        if(mid<R)
        update(k<<1|1,L,R,v);
        push_up(k);
    }
    void update(int u,int w){
        update(1,dfn[u],dfn[u]+sz[u]-1,w);
    }
```

```
    int query(int k,int pos){
        if(lef[k]==rig[k])
        return s[k];
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=pos)
        return query(k<<1,pos);
        else
        return query(k<<1|1,pos);
    }
    int query(int k,int L,int R){
        if(L<=lef[k]&&rig[k]<=R)
        return s[k];
        push_down(k);
        int mid=lef[k]+rig[k]>>1;
        if(mid>=R)
        return query(k<<1,L,R);
        else if(mid<L)
        return query(k<<1|1,L,R);
        else
        return max(query(k<<1,L,R),query(k<<1|1,L,R));
    }
    int 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;
    }
    int query1(int u,int v){
        return query(1,dfn[u])+query(1,dfn[v])-2*query(1,dfn[lca(u,v)])+1;
    }
    int query2(int u){
        return query(1,dfn[u],dfn[u]+sz[u]-1);
    }
}
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],s[MAXN],dis[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    void build(int n){
        _rep(i,1,n){
            ch[i][0]=ch[i][1]=0;
            fa[i]=Tree::f[i];
            s[i]=i;
            dis[i]=Tree::d[i];
        }
    }
    void push_up(int k){
```

```
        if(lch(k))
        s[k]=s[lch(k)];
        else
        s[k]=k;
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            if(ch[k][1])Tree::update(s[ch[k][1]],1);
            if(t)Tree::update(s[t],-1);
            ch[k][1]=t;
            push_up(k);
        }
    }
}LCT;
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);
    }
    Tree::init(n);
    LCT.build(n);
    while(m--){
        int opt=read_int();
        if(opt==1)
        LCT.access(read_int());
        else if(opt==2){
            int u=read_int(),v=read_int();
```

```
            enter(Tree::query1(u,v));
        }
        else
        enter(Tree::query2(read_int()));
    }
    return 0;
}
```

生成树维护

例题一

洛谷p3366

求最小生成树。

考虑 splay 维护一条链上的最长边，如果新加入边 (u,v) 导致成环，且原树中 $u\to v$ 路径上的最长边大于新加入的边。

则删去最长边再加入新加入边。然后发现最长边比较难直接维护，于是考虑将边也是为新节点，点权等于边权，原图中的点的点权为 0。

splay 维护最大点权以及点权最大的点的编号即可。

关于删边操作直接 $\text{split}(u,v)$ 后找到最长边对应的点的编号，然后将该编号 splay 到根节点。

不难发现该节点在 splay 中的左树恰好为 $u\to v$ 链删去该边后的一半，而右树代表另一半链。

同时 split 后 u 为原树中的根节点，于是将该编号在 splay 中的左右儿子的父节点置 0 即可分裂为两棵树，然后再加入新边即可。

注意空间要开 $O(n+m)$

```
const int MAXN=1e5+5;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],node_cnt;
    pair<int,int> w[MAXN],s[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    int node_init(int v){
        int k=++node_cnt;
        w[k]=s[k]=make_pair(v,k);
        return k;
    }
    void push_up(int k){
        s[k]=max(max(s[lch(k)],s[rch(k)]),w[k]);
```

```
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            ch[k][1]=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
```

```
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u)fa[u]=v;
    }
    void add_edge(int u,int v,int val,LL &ans){
        makeroot(u);
        if(findroot(v)!=u){
            int k=node_init(val);
            link(u,k);
            link(v,k);
            ans+=val;
        }
        split(u,v);
        if(s[v].first>val){
            int k0=s[v].second,k1=node_init(val);
            ans-=s[v].first-val;
            splay(s[v].second);
            fa[lch(k0)]=fa[rch(k0)]=0;
            link(u,k1);
            link(v,k1);
        }
    }
}LCT;
int a[MAXN];
int main()
{
    int n=read_int(),m=read_int();
    LL ans=0;
    _rep(i,1,n)
    LCT.node_init(0);
    while(m--){
        int u=read_int(),v=read_int(),w=read_int();
        LCT.add_edge(u,v,w,ans);
    }
    enter(ans);
    return 0;
}
```

例题二

洛谷p4234

定义生成树的权值为生成树权值最大的边减去权值最小的边，问权值最小的生成树。

考虑从大到小加入边，当遇到环时删去权值最大的边。每次加入边后如果图构成树则计算当前树上最大边减去最小边。

显然这是类似单调队列的贪心做法，但是本人暂时想不出正确性的证明。

```
const int MAXM=2e5+5,MAXN=5e4+MAXM,inf=1e9;
int n,m,blk_cnt;
struct Edge{
    int u,v,w;
    bool del;
    bool operator < (const Edge &b)const{
        return w>b.w;
    }
}edge[MAXM];
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],node_cnt;
    pair<int,int> w[MAXN],s[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    int node_init(int v){
        int k=++node_cnt;
        w[k]=s[k]=make_pair(v,k);
        return k;
    }
    void push_up(int k){
        s[k]=max(max(s[lch(k)],s[rch(k)]),w[k]);
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
```

```
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            ch[k][1]=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u)fa[u]=v;
    }
    void add_edge(int u,int v,int val){
        if(u==v){
            int k=++node_cnt;
            edge[k-n-1].del=true;
            return;
```

```
        }
        makeroot(u);
        if(findroot(v)!=u){
            int k=node_init(val);
            link(u,k);
            link(v,k);
            blk_cnt--;
            return;
        }
        split(u,v);
        int k0=s[v].second,k1=node_init(val);
        splay(k0);
        fa[lch(k0)]=fa[rch(k0)]=0;
        edge[k0-n-1].del=true;
        link(u,k1);
        link(v,k1);
    }
}LCT;
int main()
{
    n=read_int(),m=read_int(),blk_cnt=n;
    _for(i,0,n)
    LCT.node_init(0);
    _for(i,0,m){
        edge[i].u=read_int();
        edge[i].v=read_int();
        edge[i].w=read_int();
    }
    sort(edge,edge+m);
    int pos=0,ans=inf;
    _for(i,0,m){
        LCT.add_edge(edge[i].u,edge[i].v,edge[i].w);
        if(blk_cnt==1){
            while(edge[pos].del)pos++;
            ans=min(ans,edge[pos].w-edge[i].w);
        }
    }
    enter(ans);
    return 0;
}
```

例题三

洛谷p2387

给定 n 个点 m 条边，边 e_i 的边权为 (a_i,b_i)□定义路径 L 的长度为 $\max_{e_i\in L}a_i+\max_{e_i\in L}b_i$□求点 1 到点 n 的最短路。

考虑将边按 a_i 排序，然后依次加入边，同时维护生成树 T 使得生成树的最大 b_i 最小，然后答案为 $a_i+\max_{e_i\in T} b_i$□

```
const int MAXM=1e5+5,MAXN=5e4+MAXM,inf=1e9;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],node_cnt;
    pair<int,int> w[MAXN],s[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    int node_init(int v){
        int k=++node_cnt;
        w[k]=s[k]=make_pair(v,k);
        return k;
    }
    void push_up(int k){
        s[k]=max(max(s[lch(k)],s[rch(k)]),w[k]);
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
```

```
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            ch[k][1]=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u)fa[u]=v;
    }
    void add_edge(int u,int v,int val){
        if(u==v)return;
        makeroot(u);
        if(findroot(v)!=u){
            int k=node_init(val);
            link(u,k);
            link(v,k);
            return;
        }
        split(u,v);
        if(s[v].first>val){
            int k0=s[v].second,k1=node_init(val);
            splay(k0);
            fa[lch(k0)]=fa[rch(k0)]=0;
            link(u,k1);
            link(v,k1);
        }
    }
    int query(int u,int v){
        makeroot(u);
        if(findroot(v)==u){
```

```
            access(v);
            splay(v);
            return s[v].first;
        }
        else
        return inf;
    }
}LCT;
struct Edge{
    int u,v,w1,w2;
    bool del;
    bool operator < (const Edge &b)const{
        return w1<b.w1;
    }
}edge[MAXM];
int main()
{
    int n=read_int(),m=read_int();
    _for(i,0,n)
    LCT.node_init(0);
    _for(i,0,m){
        edge[i].u=read_int();
        edge[i].v=read_int();
        edge[i].w1=read_int();
        edge[i].w2=read_int();
    }
    sort(edge,edge+m);
    int pos=0,ans=inf;
    _for(i,0,m){
        LCT.add_edge(edge[i].u,edge[i].v,edge[i].w2);
        ans=min(ans,LCT.query(1,n)+edge[i].w1);
    }
    if(ans==inf)
    puts("-1");
    else
    enter(ans);
    return 0;
}
```

变根 LCA 维护

洛谷p3379

LCT 本身支持换根，所以只需要怎么考虑求 $\text{LCA}(u,v)$

考虑先 $\text{access}(u)$ 这样根节点到 u 的路径变实边，于是根节点到 v 的路径的第一条虚边一定从 $\text{LCA}(u,v)$ 出发。

不难发现 access 的过程就是不断从一条实链沿着虚边跳到另一条实链的过程，于是access 的过程中最后一次 splay 的节点就是 LCA

```
const int MAXN=5e5+5;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],node_cnt;
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            ch[k][1]=t;
        }
    }
    void makeroot(int k){
        access(k);
```

```
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u)fa[u]=v;
    }
    int LCA(int u,int v){
        access(u);
        int t;
        for(t=0;v;t=v,v=fa[v]){
            splay(v);
            ch[v][1]=t;
        }
        return t;
    }
}LCT;
int main()
{
    int n=read_int(),m=read_int(),s=read_int();
    _for(i,1,n){
        int u=read_int(),v=read_int();
        LCT.link(u,v);
    }
    LCT.makeroot(s);
    while(m--){
        int u=read_int(),v=read_int();
        enter(LCT.LCA(u,v));
    }
    return 0;
}
```

双连通分量维护

洛谷p2542

题意

给定一个连通图，接下来两种操作：

1. 询问 u,v 间的关键路径，定义关键路径为删除改路径讲导致 u,v 不连通的路径
2. 删除边 u,v保证删边后图仍然连通

题解

不难发现，如果根据边双连通分量进行缩点，则 u,v 间的关键路径等于 u,v 缩点后两点间的路径长度(同时此时图上所有边都是桥)。

考虑离线操作后逆序处理，加边的同时遇到环就缩点。

利用并查集维护，缩点时暴力将 u,v 路径所代表的 splay 树的所有点的代表节点设置成根节点即可，可以证明均摊复杂度 $O(1)$

然后 LCT 维护树上两点距离，注意由于对 splay 树进行了缩点，所以需要在跳虚边时更新 fa

发现只需要修改 access 操作即可。总时间复杂度 $O(n\log n)$

ps.也可以令初始时所有边权为 1，将缩点操作视为将 u,v 路径上的边权置 0，其他操作不变LCT 或树剖维护均可。

```
const int MAXN=3e4+5,MAXQ=4e4+5;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],s[MAXN],p[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    void build(int n){
        _rep(i,1,n){
            p[i]=i;
            ch[i][0]=ch[i][1]=fa[i]=0;
            s[i]=1;
            flip[i]=false;
        }
    }
    int Find(int k){
        return k==p[k]?k:p[k]=Find(p[k]);
    }
    void push_up(int k){
        s[k]=s[lch(k)]+s[rch(k)]+1;
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
```

```
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]=Find(fa[k])){
            splay(k);
            ch[k][1]=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
```

```
        access(v);
        splay(v);
    }
    void dfs(int k,int rt){
        if(k){
            p[k]=rt;
            dfs(lch(k),rt);
            dfs(rch(k),rt);
        }
    }
    void add_edge(int u,int v){
        int x=Find(u),y=Find(v);
        if(x==y)return;
        makeroot(x);
        if(findroot(y)!=x){
            fa[x]=y;
            return;
        }
        dfs(x,x);
        rch(x)=0;
        push_up(x);
    }
    int query(int u,int v){
        int x=Find(u),y=Find(v);
        split(x,y);
        return s[y]-1;
    }
}LCT;
struct{
    int opt,u,v;
}query[MAXQ];
int main()
{
    int n=read_int(),m=read_int();
    LCT.build(n);
    set<pair<int,int> >s;
    _for(i,0,m){
        int u=read_int(),v=read_int();
        if(u>v)swap(u,v);
        s.insert(make_pair(u,v));
    }
    int q=0;
    while(true){
        int opt=read_int();
        if(opt==-1)break;
        int u=read_int(),v=read_int();
        if(u>v)swap(u,v);
        if(opt==0)s.erase(make_pair(u,v));
        query[q++]={opt,u,v};
    }
    for(set<pair<int,int> >::iterator it=s.begin();it!=s.end();it++)
```

```
    LCT.add_edge(it->first,it->second);
    stack<int> ans;
    for(int i=q-1;i>=0;i--){
        if(query[i].opt==0)
        LCT.add_edge(query[i].u,query[i].v);
        else
        ans.push(LCT.query(query[i].u,query[i].v));
    }
    while(!ans.empty()){
        enter(ans.top());
        ans.pop();
    }
    return 0;
}
```

子树信息维护

洛谷p4219

题意

给定 n 个点，给定两种操作：

1. 加边操作，保证加边后不会出现环
2. 询问经过 $u\to v$ 边的路径数

题解

子树信息需要维护虚儿子和实儿子信息和，实儿子利用 $\text{ch}(k,0/1)$ 维护，虚儿子只需要额外记录即可。

以子树节点个数为例，得到 push_up 操作

```
void push_up(int k){
    s[k]=s[lch(k)]+s[rch(k)]+si[k]+1;//注意只有根节点的信息是正确的
}
```

然后需要维护 si 的值，发现只需要修改直接涉及增删虚边的函数即可，即 access 和 link 操作。

```
void access(int k){
    for(int t=0;k;t=k,k=fa[k]){
        splay(k);
        si[k]-=s[t];
        si[k]+=s[rch(k)];
        rch(k)=t;
        push_up(k);
    }
}
```

```
void link(int u,int v){
    makeroot(u);
    if(findroot(v)!=u){
        splay(v);//要确保v没有父结点
        si[v]+=s[u];
        fa[u]=v;
        push_up(v);
    }
}
```

注意，如果需要维护点权极值等信息时，可以用 set 维护 $si[k]$。

不难发现，对询问操作，可以先断开 u,v，则答案为 $\text{sz}(u)\times \text{sz}(v)$。

```
const int MAXN=1e5+5;
struct Link_Cut_tree{
    int ch[MAXN][2],fa[MAXN],s[MAXN],si[MAXN];
    int Stack[MAXN],top;
    bool flip[MAXN];
    #define lch(k) ch[k][0]
    #define rch(k) ch[k][1]
    void build(int n){
        _rep(i,1,n){
            ch[i][0]=ch[i][1]=fa[i]=0;
            s[i]=1,si[i]=0;
            flip[i]=false;
        }
    }
    void push_up(int k){
        s[k]=s[lch(k)]+s[rch(k)]+si[k]+1;//注意只有根节点的信息是正确的
    }
    void push_flip(int k){
        swap(lch(k),rch(k));
        flip[k]^=1;
    }
    void push_down(int k){
        if(flip[k]){
            push_flip(lch(k));
            push_flip(rch(k));
            flip[k]=0;
        }
    }
    bool isroot(int k){
        return lch(fa[k])!=k&&rch(fa[k])!=k;
    }
    void rotate(int k){
        int pa=fa[k],ga=fa[pa],dir;
        dir=ch[pa][0]==k?0:1;
        if(!isroot(pa))ch[ga][ch[ga][0]==pa?0:1]=k;
        fa[k]=ga,fa[pa]=k;
```

```
        if(ch[k][dir^1])fa[ch[k][dir^1]]=pa;
        ch[pa][dir]=ch[k][dir^1],ch[k][dir^1]=pa;
        push_up(pa);
    }
    void splay(int k){
        Stack[top=1]=k;
        for(int i=k;!isroot(i);i=fa[i])Stack[++top]=fa[i];
        while(top)push_down(Stack[top--]);
        while(!isroot(k)){
            int pa=fa[k],ga=fa[pa];
            if(!isroot(pa))rotate((ch[pa][0]==k)^(ch[ga][0]==pa)?k:pa);
            rotate(k);
        }
        push_up(k);
    }
    void access(int k){
        for(int t=0;k;t=k,k=fa[k]){
            splay(k);
            si[k]-=s[t];
            si[k]+=s[rch(k)];
            rch(k)=t;
            push_up(k);
        }
    }
    void makeroot(int k){
        access(k);
        splay(k);
        push_flip(k);
    }
    int findroot(int k){
        access(k);splay(k);
        push_down(k);
        while(ch[k][0])push_down(k=ch[k][0]);
        splay(k);
        return k;
    }
    void split(int u,int v){
        makeroot(u);
        access(v);
        splay(v);
    }
    void link(int u,int v){
        makeroot(u);
        if(findroot(v)!=u){
            splay(v);//要确保v没有父结点
            si[v]+=s[u];
            fa[u]=v;
            push_up(v);
        }
```

```
    }
    void cut(int u,int v){
        split(u,v);
        if(ch[v][0]==u&&ch[u][1]==0){
            ch[v][0]=fa[u]=0;
            push_up(v);
        }
    }
    LL query(int u,int v){
        cut(u,v);
        makeroot(u);
        makeroot(v);
        LL ans=1LL*s[u]*s[v];
        link(u,v);
        return ans;
    }
}LCT;
int main()
{
    int n=read_int(),q=read_int();
    LCT.build(n);
    while(q--){
        char opt=get_char();
        int u=read_int(),v=read_int();
        if(opt=='A')
        LCT.link(u,v);
        else
        enter(LCT.query(u,v));
    }
    return 0;
}
```