动态树

算法简介

动态树简称 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

```
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];//自定义
}
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);
    }
};
```

代码练习

路径信息维护

习题1

洛谷p3690

给定 n 个点和权值，接下来 m 个操作。

操作 0 ：询问 x 到 y 路径的权值的异或和，保证 x 与 y 已经连通。

操作 1 ：连接 x 与 y 若 x 与 y 已经连通，则无视这个操作。

操作 2 ：删除 x 与 y 的连边，若 x 与 y 无连边，则无视这个操作。

操作 3 ：把结点 x 的权值改为 y

一道LCT裸题，直接上代码。

```
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;
}
```

习题2

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：

操作 1 ：将 u 到 v 路径上所有点权值加上 c

操作 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。

操作 3 ：将 u 到 v 路径上所有点权值乘上 c

操作 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;
}
```

最小生成树维护

洛谷p3366

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。

```
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;
}
```