LCT

Link-Cut-Tree，中文名为动态树，是一种可以动态链信息、连通性、边权、子树信息等神奇的东西的数据结构。与树链剖分需要维护重边和轻边类似，LCT需要维护树的**实边**和**虚边**的。一个结点最多连出一条向儿子的实边，因此实边会聚集成链。splay维护一条实链。

适用范围

- 链上求和
- 链上求最值
- 链上修改
- 子树修改
- 子树求和

（上述五种均可用树链剖分解决）

- 换根
- 断开树上的一条边
- 连接两个点，保证连接后仍然是一棵树

性质

所有结点存在且仅存在于一个splay中

每个splay中的结点的深度按中序遍历顺序递增

概念

偏爱儿子：Preferred Child，最后一次被access过的儿子。

实边：Preferred Edge，连接父节点与偏爱儿子的边。

偏爱路径：Preferred Path，由实边及实边连接的结点构成的链。

辅助树：Auxiliary Tree，由一条偏爱路径上的所有节点所构成的Splay称作这条链的辅助树。

基本操作

pushup(x)：上推操作，维护信息。

```
inline void pushup(int p){
    // maintain other variables
    siz[p] = siz[ls] + siz[rs] + 1;
}
```

pushdown(x)：下推操作，维护信息。

```
inline void pushdown(int p){
    if (tag[p] != std_tag){
        // pushdown the tag
        tag[p] = std_tag;
    }
}
```

isroot(x)：判断x是否为树根。

```
inline bool isroot(int x) { return to[fa[x]][0] != x && to[fa[x]][1] != x; }
```

get(x)：对应splay中获取x是父亲的哪个儿子。

```
inline bool get(int x) { return to[fa[x]][1] == x; }
```

rotate(x)：splay中的旋转操作，将x向上旋转一层。

```
void rotate(int x){
    int f = fa[x], ff = fa[f], k = get(x);
    if (!isroot(f)) to[ff][get(f)] = x;  //这句一定要写在前面，因为 isroot 的判断机制
    to[f][k] = to[x][k ^ 1], fa[to[x][k ^ 1]] = f;
    to[x][k ^ 1] = f, fa[f] = x, fa[x] = ff;
    pushup(f), pushup(x);
}
```

splay(x)：将x旋转到树根。（详细见splay）

```
void splay(int x){
    update(x); //先下传标记，因为 splay 操作是自下向上的，上方的标记需要先 pushdown 下来
    for (int f = fa[x]; !isroot(x); rotate(x), f = fa[x])
        if (!isroot(f)) rotate(get(x) == get(f) ? f : x);
}
```

核心部分

access(x)

将原树上根到x中的所有点放在一条实链中，使根到x成为一条实链，并且在同一棵splay中。(x到根节点的路径变为偏爱路径)

算法流程：伸展x，把x的父亲的右儿子修改为下一个x，如果是第一次那就是0（虚边边实边，原右实儿子变虚儿子），对x的父亲重复上面步骤。

```
inline void access(int x){
    for(int y=0;x;x=fa(y=x)){
        splay(x);
        rs(x)=y;
        update(x);
    }
}
```

update(x)

在access操作后，递归地从上到下pushdown更新信息。

makeroot(x)

使x成为整个LCT的根。

```
void makeroot(int x) { access(x), splay(x), rev[x] ^= 1; }
```

link(x,y)：在x点和y点之间加一条边。

算法流程：先将x变为根，然后直接连到轻边上去。

```
void link(int x, int y) {
    makeroot(x);
    if(findroot(y)==x) return;
    fa(x) = y;
}
```

cut(x,y)：删除x和y点之间的边。

算法流程：Markroot(x)->判断合法性->双向短边->更新信息

①如果保证合法，直接 `split(x,y)`，此时原树上x为根，辅助树上y为根，那么x在辅助树上一定是y的左儿子，直接断开即可。

②如果不保证合法，需要满足三个条件：

x与y连通；x与y路径上没有其他链；x没有右儿子。

三面三个条件保证x与y之间有边，需要用到find函数。

```
inline void cut(int x,int y){
    mkroot(x); //换根
    if(findroot(y)!=x||fa(y)!=x||ls(y)) return;
    fa(y)=rs(x)=0;
    update(x); //这是根据题目更新信息的函数
}
```

find:找到当前辅助树对应的原树的根

在 `access(x)` 后，再 `splay(x)`，这样根就是该 Splay 中的最小值，根据"二叉搜索树"性质，一路走左儿子即可。注意过程中下推标记，注意最后把根 `splay` 上去保证复杂度。

```
int find(int x){
    access(x), splay(x);
    while (ls) pushdown(x), x = ls;
    return splay(x), x;
}
```

fix(x,k)：将x的点权修改成k。

```
access(x), splay(x), val[x] = y, pushup(x);
```

split(x,y)：拿出一颗splay维护x点与y点之间的路径，方便区间操作。

```
void split(int x,int y){
    makeroot(x);access(y);splay(y);
}
```

reverse(x)：交换x的左右子树。

```
void reverse(int x,int y){
    swap(ls(x),rs(x)),tg(ls(x))^=1,tg(rs(x))^=1;
}
```

常用宏定义

```
#define ls(x) (spl[x].ch[0])
#define rs(x) (spl[x].ch[1])
#define fa(x) (spl[x].fa)
#define get(x,f) (rs(f)==x)
#define connect(x,f,s) spl[fa(x)=f].ch[s]=x
#define update(x) spl[x].res=spl[ls(x)].res^spl[rs(x)].res^spl[x].val
#define ntroot(x) (ls(fa(x))==x||rs(fa(x))==x)
#define reverse(x) std::swap(ls(x),rs(x)),spl[x].tag^=1
```

复杂度分析

splay每次操作均摊复杂度为$O(log_2n)$

access复杂度也为$O(log_2n)$

所有lct的复杂度基本与access一致。

注意

①记得pushup和pushdown

②LCT中的rotate需要先判断isroot,所以连接父亲的父亲的关系的语句在最前面

③LCT中的splay就是旋转到根，不需要旋转到儿子

④建树时可以默认所有边为虚边，直接只指定父亲关系。

模板

P3690 【模板】动态树（Link Cut Tree）

给定 n 个点以及每个点的权值，要你处理接下来的 m 个操作。
操作有四种，操作从 0 到 3 编号。点从 1 到 n 编号。

- `0 x y` 代表询问从 x到 y 的路径上的点的权值的 xor 和。保证 x到 y 是联通的。
- `1 x y` 代表连接 x 到 y，若 x*x* 到 y已经联通则无需连接。
- `2 x y` 代表删除边 (x,y)，不保证边 (x,y) 存在。
- `3 x y` 代表将点 x 上的权值变成 y。

```
#include<bits/stdc++.h>
using namespace std;
const int N=3e5+7;

int read(){ int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return
x*f;}
void write(int x){if(x>9) write(x/10);putchar(x%10+'0');}
void swap(int &x,int &y){x^=y^=x^=y;}

class LCT{
    private:
        #define LCT_SIZE N
```

```
        #define Pushup(x)
(T[x].sum=T[x].val^T[T[x].ch[0]].sum^T[T[x].ch[1]].sum)
        #define Rever(x) (swap(T[x].ch[0],T[x].ch[1]),T[x].rev^=1)
        #define Pushdown(x) (T[x].rev&&
(Rever(T[x].ch[0]),Rever(T[x].ch[1])),T[x].rev=0)
        #define Which(x) (T[T[x].fa].ch[1]==x)
        #define Connect(x,y,d) (T[T[x].fa=y].ch[d]=x)
        #define Isroot(x) (T[T[x].fa].ch[0]^x&&T[T[x].fa].ch[1]^x)
        #define Makeroot(x) (Access(x),Splay(x),Rever(x))
        #define Split(x,y) (Makeroot(x),Access(y),Splay(y))
        int Stack[LCT_SIZE];
        struct Tree{
            int val,sum,fa,rev,ch[2];
        }T[LCT_SIZE];
        inline void Rotate(int x){
            int fa=T[x].fa,pa=T[fa].fa,d=Which(x);
            !Isroot(fa)&&
(T[pa].ch[Which(fa)]=x),T[x].fa=pa,Connect(T[x].ch[d^1],fa,d),Connect(fa,x,d^1),
Pushup(fa),Pushup(x);
        }
        inline void Splay(int x){
            int fa=x,Top=0;
            while(Stack[++Top]=fa,!Isroot(fa)) fa=T[fa].fa;
            while(Top) Pushdown(Stack[Top]),--Top;
            while(!Isroot(x)) fa=T[x].fa,!Isroot(fa)&&
(Rotate(Which(x)^Which(fa)?x:fa),0),Rotate(x);
        }
        inline void Access(int x){for(int son=0;x;x=T[son=x].fa)
Splay(x),T[x].ch[1]=son,Pushup(x);}
        inline int Findroot(int x){Access(x),Splay(x);while(T[x].ch[0])
Pushdown(x),x=T[x].ch[0];return Splay(x),x;}
    public:
        inline void Init(int len,int *data){for(int i=1;i<=len;++i)
T[i].val=data[i];}
        inline void Link(int x,int y){Makeroot(x),Findroot(y)^x&&(T[x].fa=y);}
        inline void Cut(int x,int y){Makeroot(x),!(Findroot(y)^x)&&!
(T[y].fa^x)&&!T[y].ch[0]&&(T[y].fa=T[x].ch[1]=0,Pushup(x));}
        inline void Update(int x,int v){Splay(x),T[x].val=v;}
        inline int Query(int x,int y){return Split(x,y),T[y].sum;}
}LCT;

int n,m,op,x,y,a[N];

signed main(){
    n=read();m=read();
    for(int i=1;i<=n;++i) a[i]=read();
    LCT.Init(n,a);
    for(int i=1;i<=m;++i){
        op=read();x=read();y=read();
        if(op==0){
            write(LCT.Query(x,y));puts("");
        }else if(op==1){
            LCT.Link(x,y);
        }else if(op==2){
            LCT.Cut(x,y);
        }else if(op==3){
            LCT.Update(x,y);
        }
```

```
    }
    return 0;
}
```

luoguP3203 [HNOI2010]弹飞绵羊

游戏一开始，Lostmonkey在地上沿着一条直线摆上n个装置，每个装置设定初始弹力系数ki，当绵羊达到第i个装置时，它会往后弹ki步，达到第i+ki个装置，若不存在第i+ki个装置，则绵羊被弹飞。绵羊想知道当它从第i个装置起步时，被弹几次后会被弹飞。为了使得游戏更有趣，Lostmonkey可以修改某个弹力装置的弹力系数，任何时候弹力系数均为正整数。

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+7;

int ch[N][2],sz[N],fa[N];

inline bool nroot(int x){
    return ch[fa[x]][0]==x||ch[fa[x]][1]==x;
}

inline void pushup(int x){
    sz[x]=sz[ch[x][0]]+sz[ch[x][1]]+1;
}

inline void rotate(int x){
    int y=fa[x],z=fa[y],k=ch[y][1]==x,w=ch[x][!k];
    if(nroot(y))ch[z][ch[z][1]==y]=x;
    ch[x][!k]=y;ch[y][k]=w;
    if(w) fa[w]=y;fa[y]=x;fa[x]=z;
    pushup(y);
}

inline void splay(int x){
    int y,z;
    while(nroot(x)){
        y=fa[x];z=fa[y];
        if(nroot(y)) rotate((ch[y][0]==x)^(ch[z][0]==y)?x:y);
        rotate(x);
    }
    pushup(x);
}

inline void access(int x){
    for(int y=0;x;x=fa[y=x]) splay(x),ch[x][1]=y,pushup(x);
}

signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);
    int n,m;
    cin>>n;
    for(int i=1,k;i<=n;++i){
        sz[i]=1;
        cin>>k;
```

```
        if(i+k<=n) fa[i]=i+k; //弹飞不连边
    }
    cin>>m;
    for(int i=1,op,j,k;i<=m;++i){
        cin>>op;
        if(op==1){
            cin>>j;++j;
            access(j);splay(j); //直接查询
            cout<<sz[j]<<"\n";
        }else{
            cin>>j>>k;++j;
            access(j);splay(j);
            ch[j][0]=fa[ch[j][0]]=0; //直接断边
            if(j+k<=n) fa[j]=j+k;   //直接连边
            pushup(j);
        }
    }
    return 0;
}
```

luoguP2147 [SDOI2008] 洞穴勘测

给定洞穴个数和指令个数，指令分为：连接两个洞穴、摧毁两个洞穴之间的通道、询问两个洞穴是否连通。

只有Link和Cut两种操作。以及优化LCT版本的splay写法。

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e4+7;
struct Node{
    int fa,ch[2];
    bool tg;
}spl[N];

inline int read(){  int x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9')
{if(ch=='-') f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-
'0';ch=getchar();}return x*f;}

#define ls(x) (spl[x].ch[0])
#define rs(x) (spl[x].ch[1])
#define tg(x) (spl[x].tg)
#define fa(x) (spl[x].fa)
#define get(x) (rs(fa(x))==x)
#define reverse(x) swap(ls(x),rs(x)),tg(ls(x))^=1,tg(rs(x))^=1
#define isroot(x) (ls(fa(x))!=x&&rs(fa(x))!=x)

inline void pushdown(int rt){
    if(!rt||!tg(rt)) return;
    reverse(rt);
    tg(rt)^=1;
}

inline void rotate(int rt){
```

```
    int x=fa(rt),y=fa(x),k=get(rt);
    if(!isroot(x)){
        if(ls(y)==x) ls(y)=rt;
        else rs(y)=rt;
    }
    fa(rt)=y;fa(x)=rt;fa(spl[rt].ch[!k])=x;
    spl[x].ch[k]=spl[rt].ch[!k];
    spl[rt].ch[!k]=x;
}

inline void splay(int rt){ //优化伸展，适应LCT
    int top=0,stk[N];
    stk[++top]=rt;
    for(int i=rt;!isroot(i);i=fa(i)) stk[++top]=fa(i);
    while(top) pushdown(stk[top--]);
    while(!isroot(rt)){
        int x=fa(rt),y=fa(x);
        if(!isroot(x)){
            if((ls(x)==rt)^(ls(y)==x)) rotate(rt);
            else rotate(x);
        }
        rotate(rt);
    }
}

inline void access(int x){
    for(int y=0;x;y=x,x=fa(x)){
        splay(x);
        rs(x)=y;
    }
}

inline void mkroot(int x){
    access(x);
    splay(x);
    tg(x)^=1;
}

inline void split(int x,int y){
    mkroot(x);
    access(y);
    splay(y);
}

inline int findroot(int rt){
    access(rt);
    splay(rt);
    while(ls(rt)) rt=ls(rt);
    return rt;
}

inline void link(int x,int y){
    mkroot(x);
    fa(x)=y;
}

inline void cut(int x,int y){
    split(x,y);
```

```
    ls(y)=fa(x)=0;
}

int n,m,u,v;
char op[10];

signed main(){
    n=read();m=read();
    for(int i=1;i<=m;i++){
        scanf("%s",op);
        u=read();v=read();
        if(op[0]=='Q')
            if(findroot(u)==findroot(v)) puts("Yes");
            else puts("No");
        else if(op[0]=='C') link(u,v);
        else cut(u,v);
    }
    return 0;
}
```

luogu P2387 [NOI2014] 魔法森林

给一个无向图，每条边有a和b两种精灵。如果携带的精灵大于等于边所需精灵则可通行，问1到n点最少需要带多少精灵?

动态加边可用LCT来完成，属于LCT的综合应用题了。

```
#include<bits/stdc++.h>
#define ls(x) t[x].son[0]
#define rs(x) t[x].son[1]
#define get(x) (rs(t[x].fa)==x)
#define isroot(x) (ls(t[x].fa)!=x && rs(t[x].fa)!=x)
using namespace std;

const int maxn=2e5+7,inf=1e9+7;

inline void read(int &data){
    int x=0,f=1;char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-') f=f*-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9'){
        x=x*10+ch-'0';
        ch=getchar();
    }
    data=x*f;
}

struct E{
    int from,to,a,b;
}e[maxn];

struct LCT{
    int son[2],fa,mx,id;
    bool mark;
```

```
}t[maxn];

int n,m,idnum,ans,w[maxn],st[maxn];

bool cmp(E x,E y){
    return (x.a==y.a)?x.b<y.b:x.a<y.a; //先对a排序再对b排序
}

void pushup(int x){
    t[x].mx=w[x],t[x].id=x;
    if(ls(x)&&t[ls(x)].mx>t[x].mx) t[x].mx=t[ls(x)].mx,t[x].id=t[ls(x)].id;
    if(rs(x)&&t[rs(x)].mx>t[x].mx) t[x].mx=t[rs(x)].mx,t[x].id=t[rs(x)].id;
}

void rotate(int x){
    int f=t[x].fa,ff=t[f].fa,flag=get(x);
    t[x].fa=ff;
    if(!isroot(f)) t[ff].son[rs(ff)==f]=x;
    t[f].son[flag]=t[x].son[flag^1],t[t[x].son[flag^1]].fa=f;
    t[x].son[flag^1]=f,t[f].fa=x;
    pushup(f),pushup(x);
}

void pushmark(int x){
    if(t[x].mark){
        t[x].mark=0,t[ls(x)].mark^=1,t[rs(x)].mark^=1;
        swap(ls(x),rs(x));
    }
}

void splay(int x){
    int top=0,now=x;st[++top]=now;
    while(!isroot(now)) st[++top]=(now=t[now].fa);
    while(top) pushmark(st[top--]);
    while(!isroot(x)){
        int f=t[x].fa,ff=t[f].fa;
        if(!isroot(f))((rs(ff)==f)^(rs(f)==x))?rotate(x):rotate(f);
        rotate(x);
    }
}

void access(int x){ //将根到x的所有点放在实链上
    for(int y=0;x;y=x,x=t[y].fa){
        splay(x);t[x].son[1]=y,pushup(x);
    }
}

void makeroot(int x){ //将x置为根
    access(x);splay(x);t[x].mark^=1;pushmark(x);
}

int findroot(int x){ //原树上的根
    access(x);
    splay(x);
    pushmark(x);
    while(ls(x)) pushmark(x=ls(x));
    return x;
}
```

```
void split(int x,int y){ //打通路径
    makeroot(x);access(y);splay(y);
}

void link(int x,int y){  //连接两个节点
    makeroot(x);
    if(findroot(y)!=x) t[x].fa=y;
}

bool check(int x,int y){ //查看是否在同一棵树中
    makeroot(x);
    return findroot(y)!=x;
}

signed main(){
    read(n);read(m);ans=inf;
    for(int i=1;i<=m;i++){
        read(e[i].from);read(e[i].to);read(e[i].a);read(e[i].b);
    }
    sort(e+1,e+1+m,cmp); //按a从小到大排序
    for(int i=1;i<=m;i++){ //用LCT维护边权b的最小生成树
        w[i+n]=e[i].b;
        if(e[i].from==e[i].to)continue; //去重边
        if(check(e[i].from,e[i].to))  link(e[i].from,i+n),link(i+n,e[i].to);
        else{
            split(e[i].from,e[i].to); //splay打通边两端的路径
            int now=t[e[i].to].id,maxb=t[e[i].to].mx;
            if(maxb<=e[i].b) continue; //这条边的b比之前选取的都大就不需要考虑了
            splay(now),t[ls(now)].fa=t[rs(now)].fa=0;
            link(e[i].from,i+n),link(i+n,e[i].to);
        }
        if(!check(1,n)){ //若在同一个连通块了
            split(1,n);
            ans=min(ans,t[n].mx+e[i].a); //1到n的路径上最大b的最小值+当前a
        }
    }
    if(ans==inf) puts("-1");
    else printf("%d\n",ans);
    return 0;
}
```

参考资料

https://blog.csdn.net/qq_36551189/article/details/79152612

https://zhuanlan.zhihu.com/p/346299929

https://www.luogu.com.cn/blog/Soulist/solution-p2387

https://www.cnblogs.com/chenxiaoran666/p/LCT.html

https://www.cnblogs.com/chenxiaoran666/p/LCT_problems.html

https://www.cnblogs.com/flashhu/p/9498517.html