重链剖分

算法简介

一种动态维护树上路径、子树信息的算法，单次操作时间复杂度$O\left(\log^2 n\right)$

算法思想

重链剖分的关键是把树上修改问题转化为区间修改问题

为方便理解，先考虑点权树问题

使用dfs序可以把子树查询和修改问题转化区间修改问题，单次操作时间复杂度$O\left(\log n\right)$

因此重链剖分重点在如何解决树上路径修改问题，考虑将树划分成若干条链，分别对每条链进行修改、查询操作，便可实现树上路径修改

为简化问题，先考虑根结点到某结点的路径，我们不希望看见该路径涉及成过多条链的情况，而将树划分成重路径和轻边可以很好地解决这一问题

首先给出重儿子和轻儿子的定义：对于非叶子节点，将所有儿子中以其为根的子树含结点数最多的儿子定义为重儿子，其余为轻儿子

父节点向重儿子连一条重边，每个轻儿子连一条轻边。若干重边组成一条重链

这样，根结点到某结点的路径便被划分成若干条轻边和重链相间的路径，易知每经过一条轻边，子树所含结点数减半，因此一条路径最多经过$\log n$条轻边，因此单次修改或查询根结点到某结点的路径的时间复杂度变为$O\left(\log^2 n\right)$

而对于端点为两个非根结点的路径，可以考虑像倍增法求LCA那样不断把两个结点往上提，直到两个结点位于同一条链

代码实现

先一遍dfs处理出每个结点的深度、父结点、重儿子

第二遍dfs确定dfs序和每条链的起点，注意dfs要先重儿子，再轻儿子，确保重链上dfs序连续

路径上查询和修改操作需要每次选择路径两端点中所在链的顶端结点的深度较大的结点，线段树查询/修改后跳到所在链的顶端结点的父结点

代码模板

洛谷p3384

模板题

题目大意是说给定一棵n个结点的点权树，每次操作为查询路径或子树上的权值和，或者给路径或子树上的每个点权值加v□所有结果取模

代码

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cctype>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
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 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;
int mod;
struct Tree{
    LL a[MAXN<<2],sum[MAXN<<2],lazy[MAXN<<2];
    int lef[MAXN<<2],rig[MAXN<<2];
    void init(int n,int *w){
        _rep(i,1,n)
        a[i]=w[i];
        build(1,1,n);
    }
    void push_up(int k){
        sum[k]=(sum[k<<1]+sum[k<<1|1])%mod;
    }
    void build(int k,int L,int R){
        lef[k]=L,rig[k]=R;
        int M=L+R>>1;
```

```
        if(L==R)
        return sum[k]=a[M],void();
        build(k<<1,L,M);
        build(k<<1|1,M+1,R);
        push_up(k);
    }
    void push_down(int k){
        if(lazy[k]){
            int lson=k<<1,rson=k<<1|1;
            lazy[lson]=(lazy[lson]+lazy[k])%mod;
            sum[lson]=(sum[lson]+lazy[k]*(rig[lson]-lef[lson]+1))%mod;
            lazy[rson]=(lazy[rson]+lazy[k])%mod;
            sum[rson]=(sum[rson]+lazy[k]*(rig[rson]-lef[rson]+1))%mod;
            lazy[k]=0;
        }
    }
    LL query(int k,int L,int R){
        if(L<=lef[k]&&rig[k]<=R)
        return sum[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 (query(k<<1,L,R)+query(k<<1|1,L,R))%mod;
    }
    void update(int k,int L,int R,LL v){
        if(L<=lef[k]&&rig[k]<=R){
            sum[k]=(sum[k]+(rig[k]-lef[k]+1)*v)%mod;
            lazy[k]=(lazy[k]+v)%mod;
            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);
    }
}tree;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],m;
void Insert(int u,int v){
    edge[++m].to=v;
    edge[m].next=head[u];
    head[u]=m;
```

```
}
int d[MAXN],w[MAXN],sz[MAXN],f[MAXN],dfs_id[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN],p[MAXN],dfs_w[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;p[u]=top;dfs_w[dfs_t]=w[u];
    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);
    }
}
LL query_son(int u){return tree.query(1,dfs_id[u],dfs_id[u]+sz[u]-1);}
void update_son(int u,int w){tree.update(1,dfs_id[u],dfs_id[u]+sz[u]-1,w);}
LL query_path(int u,int v){
    LL ans=0;
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        ans=(ans+tree.query(1,dfs_id[p[u]],dfs_id[u]))%mod;
        u=f[p[u]];
    }
    if(d[u]>d[v])
    swap(u,v);
    ans=(ans+tree.query(1,dfs_id[u],dfs_id[v]))%mod;
    return ans;
}
void update_path(int u,int v,int w){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        tree.update(1,dfs_id[p[u]],dfs_id[u],w);
        u=f[p[u]];
    }
```

```
    if(d[u]>d[v])
    swap(u,v);
    tree.update(1,dfs_id[u],dfs_id[v],w);
}
int main()
{
    int n=read_int(),q=read_int(),root=read_int(),opt,x,y,z;
    mod=read_int();
    _rep(i,1,n)
    w[i]=read_LL()%mod;
    _for(i,1,n){
        x=read_int(),y=read_int();
        Insert(x,y);
        Insert(y,x);
    }
    dfs_1(root,-1,0);
    dfs_2(root,root);
    tree.init(n,dfs_w);
    while(q--){
        opt=read_int();
        if(opt==1){
            x=read_int();y=read_int();z=read_int();
            update_path(x,y,z);
        }
        else if(opt==2){
            x=read_int();y=read_int();
            enter(query_path(x,y));
        }
        else if(opt==3){
            x=read_int();y=read_int();
            update_son(x,y);
        }
        else
        enter(query_son(read_int()));
    }
    return 0;
}
```

练习题

洛谷p3038

给出一棵n个结点的边权树，初始边权全为0，有m个操作，操作为将一条路径上的边权加一或询问某条边的权值

考虑一个点有唯一的父结点，因此可以把该点与父结点的连边的边权作为该点的点权。根结点权值任意，不影响结果

而路径修改/查询注意跳过LCA即可

代码

```
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <cctype>
#define _for(i,a,b) for(int i=(a);i<(b);++i)
#define _rep(i,a,b) for(int i=(a);i<=(b);++i)
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 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;
#define lowbit(x) ((x)&(-x))
struct Tree{
    LL c[2][MAXN],n;
    void add(int pos,int v){
        int k=pos;
        while(pos<=n){
            c[0][pos]+=v;
            c[1][pos]+=v*(k-1);
            pos+=lowbit(pos);
        }
    }
    void update(int lef,int rig,int v){
        add(lef,v);
        add(rig+1,-v);
    }
    LL sum(int pos){
        int k=pos;
        LL ans1=0,ans2=0;
        while(pos){
```

```
            ans1+=c[0][pos];
            ans2+=c[1][pos];
            pos-=lowbit(pos);
        }
        return ans1*k-ans2;
    }
    int query(int lef,int rig){
        return sum(rig)-sum(lef-1);
    }
}tree;
struct Edge{
    int to,next;
}edge[MAXN<<1];
int head[MAXN],m;
void Insert(int u,int v){
    edge[++m].to=v;
    edge[m].next=head[u];
    head[u]=m;
}
int d[MAXN],w[MAXN],sz[MAXN],f[MAXN],dfs_id[MAXN],dfs_t;
int h_son[MAXN],mson[MAXN],p[MAXN],dfs_w[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;p[u]=top;dfs_w[dfs_t]=w[u];
    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);
    }
}
//LL query_son(int u){return tree.query(dfs_id[u]+1,dfs_id[u]+sz[u]-1);}
//void update_son(int u,int
w){tree.update(dfs_id[u]+1,dfs_id[u]+sz[u]-1,w);}
LL query_path(int u,int v){
    LL ans=0;
```

```
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        ans+=tree.query(dfs_id[p[u]],dfs_id[u]);
        u=f[p[u]];
    }
    if(d[u]>d[v])
    swap(u,v);
    ans+=tree.query(dfs_id[u]+1,dfs_id[v]);
    return ans;
}
void update_path(int u,int v,int w){
    while(p[u]!=p[v]){
        if(d[p[u]]<d[p[v]])
        swap(u,v);
        tree.update(dfs_id[p[u]],dfs_id[u],w);
        u=f[p[u]];
    }
    if(d[u]>d[v])
    swap(u,v);
    tree.update(dfs_id[u]+1,dfs_id[v],w);
}
int main()
{
    int n=read_int(),q=read_int(),root=1,x,y;
    char opt;
    tree.n=n;
    _for(i,1,n){
        x=read_int(),y=read_int();
        Insert(x,y);
        Insert(y,x);
    }
    dfs_1(root,-1,0);
    dfs_2(root,root);
    while(q--){
        opt=get_char();
        x=read_int(),y=read_int();
        if(opt=='P')
        update_path(x,y,1);
        else
        enter(query_path(x,y));
    }
    return 0;
}
```