

# 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 $O(\log^2 n)$

## 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 $O(\log n)$

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

所有路径分为两种，一种是过根结点的，一种是完全在根结点的某棵子树中的。因此可以考虑分治算法，选取一个点将无根树转换为有根树，然后递归处理每个棵以根节点的儿子为根的子树。如果选取树的重心作为根结点，则每棵子树的结点数不超过 $\frac{n}{2}$ ，因此递归深度不超过 $\log n$

## 例题

### 洛谷p3806

题目大意是说给定一棵 $n$ 个结点的正权树，多次查询，每次查询是否存在长度为 $k$ 的路径

对根结点，先考虑暴力法，用树形dp求出子树上各节点到根结点的距离，然后两两枚举，看看有没有两个结点距离和为 $k$ ，这样每层递归的时间复杂度是 $O(n^2)$ ，显然会T

考虑用中途相遇法，可以将每层递归的时间复杂度优化到 $O(n)$ ，单次查询时间复杂度 $O(n \log n)$

但要注意两点

一、枚举过根结点的路径时路径两端点显然不能在同一棵子树里，所以要先求出一棵子树所有的 $\text{dist}$ ，全部判断完后再进行标记

二、题目给定 $k \leq 10^7$ ，所以不用标记大于 $10^7$ 的 $\text{dist}$

另一个优化是可以离线处理查询，这样只需要分治一次。

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;
}

```

## 变式1


### 洛谷p4178

给定一棵  $n$  个结点的正权树和一个正数  $k$  统计有多少对结点  $(a,b)$  满足  $\text{dist}(a,b) \leq k$

把中途相遇法换成名次树即可，时间复杂度  $O(n \left(\log n\right)^2)$

Last update: 2020-2021:teams:legal\_string:jxm2001: [https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E9%87%8D%E9%93%BE%E5%89%96%E5%88%86&rev=1589420652](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E9%87%8D%E9%93%BE%E5%89%96%E5%88%86&rev=1589420652)  
2020/05/14 09:44 重链剖分

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link: [https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:%E9%87%8D%E9%93%BE%E5%89%96%E5%88%86&rev=1589420652](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E9%87%8D%E9%93%BE%E5%89%96%E5%88%86&rev=1589420652) 

Last update: 2020/05/14 09:44