树链剖分（重链剖分）

原题链接

P3384 【模板】轻重链剖分/树链剖分

前置数组

```
int siz[maxn], father[maxn], son[maxn], depth[maxn];
int tim = 0, dfsOrder[maxn], top[maxn], w[maxn];
int val[maxn];
vector<int> g[maxn];

struct Node
{
    int val, l, r;
    int lazy;
}tree[4 * maxn];
```

siz：包含当前节点及其子树的所有节点数量

father：父节点

son：重儿子节点

depth：节点深度

tim：计数器，用来求dfs序

dfsOrder：dfs序

top：当前节点所在重链的起始节点，重链的起始节点和叶子节点的top为其本身

w：存放原数组根据dfs排序后的新数组

val：存放原数组

tree：线段树

第一次dfs

```
int siz[maxn], father[maxn], son[maxn], depth[maxn];
void dfs1(int u, int v)
{
    father[u] = v;
    depth[u] = depth[v] + 1;
    siz[u] = 1;
    int maxsize = -1;
    for (auto& to : g[u])
    {
        if (to == v)
            continue;
        dfs1(to, u);
        siz[u] += siz[to];
```

```
        if (siz[to] > maxsize)
        {
            son[u] = to;
            maxsize = siz[to];
        }
    }
}
```

第二次dfs

```
int tim = 0, dfsOrder[maxn], top[maxn], w[maxn];
int val[maxn];
void dfs2(int u, int v)
{
    dfsOrder[u] = ++tim;
    top[u] = v;
    w[tim] = val[u];
    if (!son[u])
        return;
    dfs2(son[u], v);
    for (auto& to : g[u])
    {
        if (to == son[u] || to == father[u])//这里要注意是father[u]而不是v!
            continue;
        dfs2(to, to);
    }
}
```

链上修改

给定两节点，在这两节点在树上最短路径上的所有节点都修改

```
void chainModify(int x, int y, int z)
{
    z %= mod;
    while (top[x] != top[y])
    {
        if (depth[top[x]] < depth[top[y]])
            swap(x, y);
        modify(dfsOrder[top[x]], dfsOrder[x], z);
        x = father[top[x]];
    }
    if (depth[x] > depth[y])
        swap(x, y);
    modify(dfsOrder[x], dfsOrder[y], z);
}
```

链上查询

给定两节点，求这两节点在树上最短路径上所有节点的和

```
int chainQuery(int x, int y)
{
```

```
    int ret = 0;
    while (top[x] != top[y])
    {
        if (depth[top[x]] < depth[top[y]])
            swap(x, y);
        ret += query(dfsOrder[top[x]], dfsOrder[x]);
        ret %= mod;
        x = father[top[x]];
    }
    if (depth[x] > depth[y])
        swap(x, y);
    ret += query(dfsOrder[x], dfsOrder[y]);
    return ret % mod;
}
```

子树修改

给定一节点 x， 将以 x 为根节点的子树内所有节点值都加上 z

```
void sonModify(int from, int x)
{
    modify(dfsOrder[from], dfsOrder[from] + siz[from] - 1, x);
}
```

子树查询

给定一节点 x， 求以 x 为根节点的子树内所有节点值之和

```
int sonQuery(int from)
{
    return query(dfsOrder[from], dfsOrder[from] + siz[from] - 1);
}
```

完整代码

```
#include <bits/stdc++.h>
#define endl '\n'

using namespace std;

using ll = long long;
using VI = vector<int>;
using PII = pair<int, int>;

const int maxn = 1e5 + 5;
int mod = 1e9 + 7;

int siz[maxn], father[maxn], son[maxn], depth[maxn];
int tim = 0, dfsOrder[maxn], top[maxn], w[maxn];
int val[maxn];
vector<int> g[maxn];

struct Node
```

```
{
    int val, l, r;
    int lazy;
}tree[4 * maxn];

int ls(int n) { return n << 1; }
int rs(int n) { return n << 1 | 1; }

void pushUp(int now)
{
    tree[now].val = (tree[ls(now)].val + tree[rs(now)].val) % mod;
}

void pushDown(int now)
{
    if (tree[now].l == tree[now].r)
        return;
    tree[ls(now)].lazy += tree[now].lazy;
    tree[rs(now)].lazy += tree[now].lazy;
    tree[ls(now)].val += (tree[ls(now)].r - tree[ls(now)].l + 1) * 1ll *
tree[now].lazy % mod;
    tree[rs(now)].val += (tree[rs(now)].r - tree[rs(now)].l + 1) * 1ll *
tree[now].lazy % mod;
    tree[now].lazy = 0;
}

void buildTree( int l, int r,int now=1)
{
    tree[now].l = l, tree[now].r = r;
    if (l == r)
    {
        tree[now].val = w[l] % mod;
        return;
    }
    int mid = (l + r) >> 1;
    buildTree( l, mid,ls(now));
    buildTree( mid + 1, r,rs(now));
    pushUp(now);
}

void modify(int l, int r, int x, int now = 1)
{
    int nowl = tree[now].l, nowr = tree[now].r;
    if (l <= nowl && r >= nowr)
    {
        tree[now].lazy += x;
        tree[now].val += (nowr - nowl + 1) * x;
        tree[now].val %= mod;
        return;
    }
    if (tree[now].lazy)
        pushDown(now);
    int mid = (nowl + nowr) >> 1;
    if (l <= mid)
        modify(l, r, x, ls(now));
```

```
    if (r > mid)
        modify(l, r, x, rs(now));
    pushUp(now);
}

int query(int l, int r, int now = 1)
{
    int nowl = tree[now].l, nowr = tree[now].r;
    if (l <= nowl && r >= nowr)
        return tree[now].val;
    if (tree[now].lazy)
        pushDown(now);
    int ret = 0, mid = (nowl + nowr) >> 1;
    if (l <= mid)
        ret += query(l, r, ls(now));
    if (r > mid)
        ret += query(l, r, rs(now));
    return ret % mod;
}

void dfs1(int u, int v)
{
    father[u] = v;
    depth[u] = depth[v] + 1;
    siz[u] = 1;
    int maxsize = -1;
    for (auto& to : g[u])
    {
        if (to == v)
            continue;
        dfs1(to, u);
        siz[u] += siz[to];
        if (siz[to] > maxsize)
        {
            son[u] = to;
            maxsize = siz[to];
        }
    }
}


void dfs2(int u, int v)
{
    dfsOrder[u] = ++tim;
    top[u] = v;
    w[tim] = val[u];
    if (!son[u])
        return;
    dfs2(son[u], v);
    for (auto& to : g[u])
    {
        if (to == son[u] || to == father[u])//这里要注意是father[u]而不是v!
            continue;
        dfs2(to, to);
    }
```

```
}

void sonModify(int from, int x)
{
    modify(dfsOrder[from], dfsOrder[from] + siz[from] - 1, x);
}

int sonQuery(int from)
{
    return query(dfsOrder[from], dfsOrder[from] + siz[from] - 1);
}

void chainModify(int x, int y, int z)
{
    z %= mod;
    while (top[x] != top[y])
    {
        if (depth[top[x]] < depth[top[y]])
            swap(x, y);
        modify(dfsOrder[top[x]], dfsOrder[x], z);
        x = father[top[x]];
    }
    if (depth[x] > depth[y])
        swap(x, y);
    modify(dfsOrder[x], dfsOrder[y], z);
}

int chainQuery(int x, int y)
{
    int ret = 0;
    while (top[x] != top[y])
    {
        if (depth[top[x]] < depth[top[y]])
            swap(x, y);
        ret += query(dfsOrder[top[x]], dfsOrder[x]);
        ret %= mod;
        x = father[top[x]];
    }
    if (depth[x] > depth[y])
        swap(x, y);
    ret += query(dfsOrder[x], dfsOrder[y]);
    return ret % mod;
}

int main()
{
    int n, m, root;
    cin >> n >> m >> root >> mod;
    for (int i = 1; i <= n; ++i)
        cin >> val[i];
    for (int i = 1; i <= n - 1; ++i)
    {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
```

```
        g[v].push_back(u);
    }
    dfs1(root, root);
    dfs2(root, root);
    buildTree(1, n);
    while (m--)
    {
        int tag;
        cin >> tag;
        if (tag == 1)
        {
            int x, y, z;
            cin >> x >> y >> z;
            chainModify(x, y, z);
        }
        else if (tag == 2)
        {
            int x, y;
            cin >> x >> y;
            cout << chainQuery(x, y) << endl;
        }
        else if (tag == 3)
        {
            int x, y;
            cin >> x >> y;
            sonModify(x, y);
        }
        else if (tag == 4)
        {
            int x;
            cin >> x;
            cout << sonQuery(x) << endl;
        }
    }
    return 0;
}
```