树链剖分+dfs序+线段树实现路径或子树信息修改

```
#include <bits/stdc++.h>


#define debug(x) {std::cout<<(x)<<'\n';exit(0);}

const long long N = 2e6;
long long n, m, rt, mod;
long long g[N], e[N], ne[N], ant;
long long a[N];

void add(long long x, long long y)
{
    ant ++;
    e[ant] = y;
    ne[ant] = g[x];
    g[x] = ant;
}

long long cur;
long long id[N]; // dfs 序
long long rid[N]; // id 对应的节点
long long st[N]; // 子树的开端
long long ed[N]; // 子树的末尾
long long hson[N]; // 重儿子
long long hfa[N]; // 重链头
long long fa[N]; // 节点的父亲
long long size[N]; // 子树节点个数
long long deep[N]; // 节点深度

void dfs_1(long long u, long long v)
{
    fa[u] = v;
    deep[u] = 1 + deep[v];
    size[u] = 1;

    for (long long i = g[u]; i; i = ne[i])
    {
        if (e[i] == v) continue;

        dfs_1(e[i], u);

        long long t = size[e[i]];

        size[u] += t;

        if (size[hson[u]] < t)
        {
            hson[u] = e[i];
```

```
        }
    }
}

void dfs_2(long long u, long long v)
{
    if (u == 0 || v == 0) return;
    hfa[u] = v;
    id[u] = st[u] = ++ cur;
    rid[cur] = u;

    dfs_2(hson[u], v);

    for (long long i = g[u]; i; i = ne[i])
    {
        if (hfa[e[i]]) continue;

        dfs_2(e[i], e[i]);
    }

    ed[u] = cur;
}

// --------------------------------

long long tr[N << 2], tg[N << 2], ls[N << 2], rs[N << 2], cnt;
long long tree;

void push_up(long long u, long long l, long long r)
{
    if (l == r) return;
    tr[u] = tr[ls[u]] + tr[rs[u]];
    tr[u] %= mod;
}

void set(long long u, long long l, long long r, long long x)
{
    tg[u] = (tg[u] + x) % mod;
    tr[u] = (tr[u] + (r - l + 1) * x % mod) % mod;
}

void push_down(long long u, long long l, long long r)
{
    if (tg[u] == 0) return;
    if (l == r) return;

    long long mid = (l + r) >> 1;

    set(ls[u], l, mid, tg[u]);
    set(rs[u], mid + 1, r, tg[u]);

    tg[u] = 0;
    push_up(u, l, r);
}
```

```
void build(long long& tree, long long l, long long r)
{
    tree = ++ cnt;
    if (l == r)
    {
        tr[tree] = a[rid[l]];
        return;
    }
    long long mid = (l + r) >> 1;

    build(ls[tree], l, mid);
    build(rs[tree], mid + 1, r);

    push_up(tree, l, r);
}

void add(long long tree, long long l, long long r, long long ll, long long rr,
long long x)
{
    push_down(tree, l, r);

    if (ll <= l && r <= rr)
    {
        set(tree, l, r, x);
        return;
    }

    long long mid = (l + r) >> 1;

    if (ll <= mid) add(ls[tree], l, mid, ll, rr, x);
    if (rr > mid) add(rs[tree], mid + 1, r, ll, rr, x);

    push_up(tree, l, r);
}

long long query(long long tree, long long l, long long r, long long ll, long long
rr)
{
    push_down(tree, l, r);
    if (ll <= l && r <= rr)
        return tr[tree];

    long long mid = (l + r) >> 1;

    long long res = 0;

    if (ll <= mid) res = (res + query(ls[tree], l, mid, ll, rr)) % mod;
    if (rr > mid) res = (res + query(rs[tree], mid + 1, r, ll, rr)) % mod;

    push_up(tree, l, r);

    return res % mod;
}
```

```cpp
void add_stree(long long u, long long x)
{
    add(tree, 1, cur, st[u], ed[u], x);
}

long long query_stree(long long u)
{
    return query(tree, 1, cur, st[u], ed[u]) % mod;
}

void add_path(long long u, long long v, long long x)
{
    if (deep[u] > deep[v]) std::swap(u, v);
    if (hfa[u] == hfa[v])
    {
        add(tree, 1, cur, id[u], id[v], x);
        return;
    }
    if (deep[hfa[u]] > deep[hfa[v]]) std::swap(u, v);

    add(tree, 1, cur, id[hfa[v]], id[v], x);

    add_path(u, fa[hfa[v]], x);
}

long long query_path(long long u, long long v)
{
    if (deep[u] > deep[v]) std::swap(u, v);
    if (hfa[u] == hfa[v])
    {
        return query(tree, 1, cur, id[u], id[v]);
    }

    if (deep[hfa[u]] > deep[hfa[v]]) std::swap(u, v);

    long long ans = query(tree, 1, cur, id[hfa[v]], id[v]);

    return (ans + query_path(u, fa[hfa[v]])) % mod;
}

void solve()
{

    std::cin >> n >> m >> rt >> mod;
    for (long long i = 1; i <= n; i ++)
    {
        std::cin >> a[i];
    }
    for (long long i = 1; i < n; i ++)
    {
        long long x, y; std::cin >> x >> y;
        add(x, y);
        add(y, x);
```

```
    }

    dfs_1(rt, 0);
    dfs_2(rt, rt);
    build(tree, 1, cur);

    while (m --)
    {
        long long mo; std::cin >> mo;

        if (mo == 1)
        {
            long long x, y, z; std::cin >> x >> y >> z;

            add_path(x, y, z);
        }
        else if (mo == 2)
        {
            long long x, y; std::cin >> x >> y;
            std::cout << query_path(x, y) << '\n';
        }
        else if (mo == 3)
        {
            long long x, y; std::cin >> x >> y;
            add_stree(x, y);
        }
        else if (mo == 4)
        {
            long long x; std::cin >> x;
            std::cout << query_stree(x) << '\n';
        }
    }

}

int main()
{
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);


    solve();
    return 0;
}
```