Powered by:**NEFU AB-IN**

P2146 [NOI2015] 软件包管理器

- 题意

> **题目背景** 展开
>
> Linux 用户和 OSX 用户一定对软件包管理器不会陌生。通过软件包管理器，你可以通过一行命令安装某一个软件包，然后软件包管理器会帮助你从软件源下载软件包，同时自动解决所有的依赖（即下载安装这个软件包的安装所依赖的其它软件包），完成所有的配置。Debian/Ubuntu 使用的 apt-get，Fedora/CentOS 使用的 yum，以及 OSX 下可用的 homebrew 都是优秀的软件包管理器。
>
> **题目描述**
>
> 你决定设计你自己的软件包管理器。不可避免地，你要解决软件包之间的依赖问题。如果软件包 a 依赖软件包 b，那么安装软件包 a 以前，必须先安装软件包 b。同时，如果想要卸载软件包 b，则必须卸载软件包 a。
>
> 现在你已经获得了所有的软件包之间的依赖关系。而且，由于你之前的工作，除 0 号软件包以外，在你的管理器当中的软件包都会依赖一个且仅一个软件包，而 0 号软件包不依赖任何一个软件包。且依赖关系不存在环（即不会存在 m 个软件包 $a_1, a_2, \ldots, a_m$，对于 $i < m$，a_i 依赖 $a_i + 1$，而 a_m 依赖 a_1 的情况）。
>
> 现在你要为你的软件包管理器写一个依赖解决程序。根据反馈，用户希望在安装和卸载某个软件包时，快速地知道这个操作实际上会改变多少个软件包的安装状态（即安装操作会安装多少个未安装的软件包，或卸载操作会卸载多少个已安装的软件包），你的任务就是实现这个部分。
>
> 注意，安装一个已安装的软件包，或卸载一个未安装的软件包，都不会改变任何软件包的安装状态，即在此情况下，改变安装状态的软件包数为 0。

简而言之，根节点为0，其余所有节点都只有唯一的一个父节点

- $install$，加载一个节点，问需要先加载多少个它所依赖的节点
- $uninstall$，卸载一个节点，问当它被卸载时多少个节点会被牵连

- 思路

当把树建好之后，这就是一个裸的树剖

- $install$，加载节点，查询它到根节点0这一段节点们中，有多少个0，并全赋值为1，可以用线段树维护
- $uninstall$，卸载一个节点，即它的为1的子节点有多少个，这个可以用dfs序加上它的$size$完成

注意两个点

- 虽然板子是线段树加法的板子，但我们要做的不是**累加**，而是**覆盖**，即权值覆盖
- 将判断是否有懒标记改为-1，因为0也有用，代表卸载

- 代码

```
#include <bits/stdc++.h>
using namespace std;
#define LL long long
#define MP make_pair
#define SZ(X) ((int)(X).size())
#define IOS                         \
    ios::sync_with_stdio(false); \
    cin.tie(0);                  \
    cout.tie(0);
#define DEBUG(X) cout << #X << ": " << X << endl;
#define ls i << 1
#define rs i << 1 | 1
typedef pair<int, int> PII;

const int N = 20;
struct Edge
{
    int v, ne;
} e[N << 2];
int h[N];
int cnt;
void add(int u, int v)
{
    e[cnt].v = v;
    e[cnt].ne = h[u];
    h[u] = cnt++;
}
void init()
{
    memset(h, -1, sizeof(h));
    cnt = 0;
}

namespace TreeChain
{
    int w[N];
    int pre[N], sizx[N], son[N], deep[N];
    int dfn[N], top[N], a[N];
    int cnx; // dfs2 pool

    void dfs1(int u, int fa)
    {
        pre[u] = fa;
        deep[u] = deep[fa] + 1;
        sizx[u] = 1;
        int maxson = -1;
        for (int i = h[u]; ~i; i = e[i].ne)
        {
            int v = e[i].v;
            if (v != fa)
```

```
            {
                dfs1(v, u);
                sizx[u] += sizx[v];
                if (maxson < sizx[v])
                {
                    maxson = sizx[v];
                    son[u] = v;
                }
            }
        }
    }

    void dfs2(int u, int t)
    {
        top[u] = t;
        dfn[u] = ++cnx;
        a[cnx] = w[u];
        if (!son[u])
            return;
        dfs2(son[u], t);
        for (int i = h[u]; ~i; i = e[i].ne)
        {
            int v = e[i].v;
            if (v != pre[u] && v != son[u])
            {
                dfs2(v, v);
            }
        }
    }
}
using namespace TreeChain;

namespace xds_add
{
    LL a[N << 2], tr[N << 2], add_tag[N << 2], k;
    int n, x, y;

    inline void pushup(int i)
    {
        tr[i] = tr[ls] + tr[rs];
    }

    void build(int i, int l, int r)
    {
        add_tag[i] = -1;
        if (l == r)
        {
            tr[i] = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(ls, l, mid);
        build(rs, mid + 1, r);
        pushup(i);
```

```
    }

    inline void ADD(int i, int l, int r, LL k)
    {
        // 在ADD函数中将 += 改为 =
        add_tag[i] = k;
        tr[i] = (r - l + 1) * k;
    }

    inline void pushdown(int i, int l, int r, int mid)
    {
        if ((add_tag[i]) == -1)
            return;
        ADD(ls, l, mid, add_tag[i]);
        ADD(rs, mid + 1, r, add_tag[i]);
        add_tag[i] = -1;
    }

    inline void update_ADD(int i, int l, int r, int x, int y, LL k)
    {
        if (l > y || r < x)
            return;
        if (l >= x && r <= y)
            return ADD(i, l, r, k);
        int mid = (l + r) >> 1;
        pushdown(i, l, r, mid);
        update_ADD(ls, l, mid, x, y, k);
        update_ADD(rs, mid + 1, r, x, y, k);
        pushup(i);
    }

    LL query(int i, int l, int r, int x, int y)
    {
        LL res = 0;
        if (l > y || r < x)
            return 0;
        if (l >= x && r <= y)
            return tr[i];
        int mid = (l + r) >> 1;
        pushdown(i, l, r, mid);
        if (x <= mid)
            res = res + query(ls, l, mid, x, y);
        if (y > mid)
            res = res + query(rs, mid + 1, r, x, y);
        return res;
    }
}
using namespace xds_add;

int install(int x)
{
    int ans = deep[x];
    while (top[x] != 1)
    {
```

```
        ans -= query(1, 1, n, dfn[top[x]], dfn[x]);
        update_ADD(1, 1, n, dfn[top[x]], dfn[x], 1);
        x = pre[top[x]];
    }
    ans -= query(1, 1, n, 1, dfn[x]);
    update_ADD(1, 1, n, 1, dfn[x], 1);
    return ans;
}

int uninstall(int x)
{
    int ans = query(1, 1, n, dfn[x], dfn[x] + sizx[x] - 1); // 即自己的dfs序
加上自己的子节点数
    update_ADD(1, 1, n, dfn[x], dfn[x] + sizx[x] - 1, 0);
    return ans;
}

signed main()
{
    init();
    IOS;
    cin >> n;
    for (int i = 2; i <= n; ++i)
    {
        int x;
        cin >> x;
        add(x + 1, i);
    }
    dfs1(1, 0);
    dfs2(1, 1);
    build(1, 1, n);
    int q;
    cin >> q;
    for (int i = 1; i <= q; ++i)
    {
        string s;
        int x;
        cin >> s >> x;
        x++; // 变成下标从1开始
        if (s == "install")
            cout << install(x) << '\n';
        else
            cout << uninstall(x) << '\n';
    }
    return 0;
}
```