二叉树 ☆☆

题目描述

小杨有一棵包含 n 个节点的二叉树，且根节点的编号为 1。这棵二叉树任意一个节点要么是白色，要么是黑色。之后小杨会对这棵二叉树进行 q 次操作，每次小杨会选择一个节点，将以这个节点为根的子树内所有节点的颜色反转，即黑色变成白色，白色变成黑色。

小杨想知道 q 次操作全部完成之后每个节点的颜色。

输入输出格式

输入格式：

第一行：正整数 n，表示节点数量

第二行：$(n-1)$ 个正整数，表示节点 2 到 n 的父亲节点

第三行：长度为 n 的 01 串，表示初始颜色

第四行：正整数 q，表示操作次数

接下来 q 行：每行一个正整数 a_i，表示操作节点

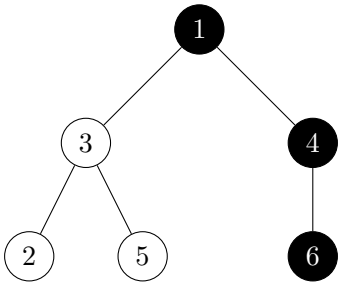
输出格式：

一行：长度为 n 的 01 串，第 i 位为 0 表示节点 i 为白色，第 i 位为 1 表示节点 i 为黑色

输入示例	输出示例
6 3 1 1 3 4 100101 3 1 3 2	010000

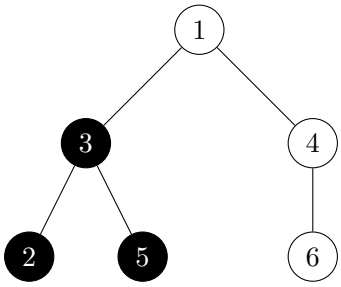
样例解释

1. 初始状态：



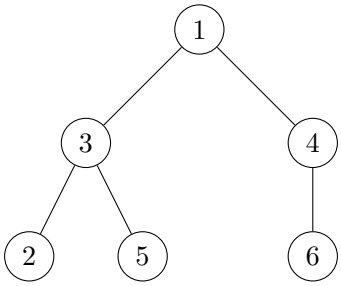
- 节点颜色：1(1), 2(0), 3(0), 4(1), 5(0), 6(1)
- 颜色序列：100101

2. **第一次操作（节点1）：**



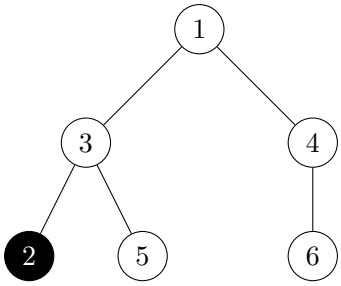
- **反转范围**：整个树（所有节点）
- 反转后颜色：1(0), 2(1), 3(1), 4(0), 5(1), 6(0)
- 颜色序列：011010

3. **第二次操作（节点3）：**



- **反转范围**：节点3及其子树（节点3,2,5）
- 反转后颜色：1(0), 2(0), 3(0), 4(0), 5(0), 6(0)
- 颜色序列：000000

4. **第三次操作（节点2）：**



- **反转范围**：节点2（只包含节点2）
- 反转后颜色：1(0), 2(1), 3(0), 4(0), 5(0), 6(0)
- 颜色序列：010000

算法分析

1. **直接模拟方法**
   对于每次操作，遍历以该节点为根的子树，反转所有节点的颜色
   这种方法的时间复杂度为 $O(n \cdot q)$，在 $n, q = 10^5$ 时会超时。

2. **优化方法：标记传递**
   利用标记数组记录每个节点被反转的次数，最后统一处理。

   （1） **核心思想**

   - 使用一个标记数组 `tag` 记录每个节点被反转的次数
   - 对于每次操作，只在操作节点上增加标记
   - 通过一次遍历，将父节点的标记传递给子节点
   - 如果反转次数是奇数，则反转当前节点, 否则不反转

   （2） **算法步骤**

   - 构建二叉树结构

   ```
   vector<vector<int>> adj(n);
   adj[u].push_back(v);
   adj[v].push_back(u);
   ```

   - 初始化标记数组 `tag` 为 0
   - 对于每个操作，`tag[操作节点]++`
   - 使用 BFS 从根节点开始遍历，将父节点的标记累加到子节点

   ```
   for (int i = 0; i < adj[u].size(); i++) {
       int v = adj[u][i];
       if (v == fa) continue;
       tag[v] += tag[u];
       qu.push({v, u});
   }
   ```

   - 计算每个节点的最终颜色

   ```
   if (tag[u] & 1) col[u] ^= 1;
   ```

拓展思考

- 如果颜色的数目修改为k种，且每次操作是将该节点及其子树颜色全部设置为x,那么算法该如何修改？
- 如果树不是二叉树，而是多叉树，算法是否仍然适用？