29、目录删除，考点 or 实现——数据结构/二叉树

题目描述

某文件系统中有 N 个目录，每个目录都有一个独一无二的 ID。

每个目录只有一个父目录，但每个父目录下可以有零个或者多个子目录，目录结构呈树状结构。

假设，根目录的 ID 为 0，且根目录没有父目录，其他所有目录的 ID 用唯一的正整数表示，并统一编号。

现给定目录 ID 和其父目录 ID 的对应父子关系表[子目录 ID，父目录 ID]，以及一个待删除的目录 ID，请计算并返回一个 ID 序列，表示因为删除指定目录后剩下的所有目录，返回的ID序列以递增序输出。

注意

1、被删除的目录或文件编号一定在输入的 ID 序列中；

2、当一个目录删除时，它所有的子目录都会被删除。

输入描述

输入的第一行为父子关系表的长度 m；

接下来的 m 行为 m 个父子关系对；

最后一行为待删除的 ID。

序列中的元素以空格分割，参见样例。

输出描述

输出一个序列，表示因为删除指定目录后，剩余的目录 ID。

用例

输入	5 8 6 10 8 6 0 20 8 2 6 8
输出	2 6
说明	目录结构如下所示： 6 / \ 2 8 / \ 10 20 删除目录8，同时它的子目录10也被删除，剩余2和6两个目录。

题目解析

本题咋看上去是让模拟N叉树结构，然后做节点删除操作，最后遍历N叉树。

但是这样的话思考的话，就太复杂了。

本题其实并不需要删除节点，也不需要遍历N叉树，我们可以在模拟N叉树的过程中，就统计节点，并排除要删除的节点的插入。

我首先，统计了所有父节点下的子节点，比如

8 6
10 8
6 0
20 8
2 6

可以统计为：

```
tree : {
  0: [6]
  6: [8, 2],
  8: [10, 20]
  2: []
}
```

然后从根节点0开始遍历N叉树

然后从根节点0开始遍历N叉树

```
let children = tree[0]
for(let i=0; i<children.length; i++) {
  if(children[i] !== remove) {
    // 记录树节点
    // 递归处理children[i]，即将children[i]当成父节点继续遍历查找其子节点，重复上面逻辑
  }
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = sc.nextInt();

    int[][] relations = new int[m][2];
    for (int i = 0; i < m; i++) {
      relations[i][0] = sc.nextInt();
      relations[i][1] = sc.nextInt();
    }

    int del = sc.nextInt();

    System.out.println(getResult(m, relations, del));
  }

  public static String getResult(int m, int[][] relations, int del) {
    HashMap<Integer, ArrayList<Integer>> tree = new HashMap<>();

    for (int[] relation : relations) {
      int child = relation[0];
      int father = relation[1];
```

```
      int father = relation[1];
      tree.putIfAbsent(father, new ArrayList<>());
      tree.get(father).add(child);
    }

    if (del == 0) {
      return "";
    }

    ArrayList<Integer> res = new ArrayList<>();
    dfs(tree, 0, del, res);

    res.sort((a, b) -> a - b);
    StringJoiner sj = new StringJoiner(" ");
    for (Integer v : res) {
      sj.add(v + "");
    }
    return sj.toString();
  }

  public static void dfs(
      HashMap<Integer, ArrayList<Integer>> tree, int node, int del, ArrayList<Integer> res) {
    if (tree.containsKey(node)) {
      ArrayList<Integer> children = tree.get(node);
      for (Integer child : children) {
        if (child != del) {
          res.add(child);
          dfs(tree, child, del, res);
        }
      }
    }
  }
}
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    m = parseInt(lines[0]);
  }

  if (m && lines.length === m + 2) {
    lines.shift();
    const del = parseInt(lines.pop());
    const arr = lines.map((line) => line.split(" ").map(Number));

    console.log(getRemainTreeEle(arr, del));
    lines.length = 0;
  }
});

function getRemainTreeEle(arr, del) {
```

```
  let tree = {};

  for (let i = 0; i < arr.length; i++) {
    let [child, father] = arr[i];
    tree[father] ? tree[father].push(child) : (tree[father] = [child]);
  }

  if (del === 0) return "";

  const res = [];
  dfs(tree, 0, del, res);

  return res.sort((a, b) => a - b).join(" ");
}

function dfs(tree, node, del, res) {
  const children = tree[node];
  if (children)
    for (let i = 0; i < children.length; i++) {
      if (children[i] !== del) {
        res.push(children[i]);
        dfs(tree, children[i], del, res);
      }
    }
}
```

Python算法源码

```python
# 输入获取
m = int(input())
relations = [list(map(int, input().split())) for _ in range(m)]
remove = int(input())


def dfs(tree, node, remove, res):
    if tree.get(node) is not None:
        children = tree[node]
        for child in children:
            if child != remove:
                res.append(child)
                dfs(tree, child, remove, res)


# 算法入口
def getResult():
    tree = {}

    for child, father in relations:
        if tree.get(father) is None:
            tree[father] = []
        tree[father].append(child)

    if remove == 0:
        return ""

    res = []
    dfs(tree, 0, remove, res)

    res.sort()
    return " ".join(map(str, res))


# 调用算法
print(getResult())
```