链滴

PAT 甲级刷题实录——1004

作者：aopstudio
原文链接：https://ld246.com/article/1578731974464
来源网站：链滴

原题链接

https://pintia.cn/problem-sets/994805342720868352/problems/994805521431773184

思路

很明显这题需要用到树这个数据结构，问题是怎么来存。一开始我是这样想的：因为它只问了每一层子结点数，所以最简单的情况下我只需要两个数据就行，一个是结点的所在的层级，另一个是结点是含有子结点。所有的结点都存储在vector中，另外创建一个存储每一层叶子结点个数的数组用于最后出结果。遍历vector，记录每一层不含有子结点的个数，即为该层叶子结点个数，并把它存储到该层结果数组中对应的位置上去。代码如下：

```
#include <iostream>
#include <vector>
using namespace std;
struct node
{
    int level;    //所在层级，从0开始
    bool hasChild = false;
};
int main()
{
    int nodeSum, nonLeafSum;    //结点总数，非叶子结点总数
    cin >> nodeSum >> nonLeafSum;
    vector<node> tree(nodeSum+1);    //树，从1开始计数
    tree[1].level = 0;    //初始化根节点所在的层级
    int maxLevel = 0;    //存储最大层数
    for (int i = 0; i < nonLeafSum; i++)
    {
        int parent;
        cin >> parent;
        tree[parent].hasChild = true;
        int childSum;
        cin >> childSum;
        for (int j = 0; j < childSum; j++)
        {
            int childNum;
            cin >> childNum;
            tree[childNum].level = tree[parent].level + 1;    //记录结点所在层级
            if (tree[childNum].level > maxLevel)
            {
                maxLevel = tree[childNum].level;    //更新最大层级
            }
        }
    }
    vector<int> result(maxLevel+1,0);    //存储输出结果，即每层的叶子节点数
    for (int i = 1; i <= nodeSum; i++)
    {
        if (tree[i].hasChild == false)
        {
            result[tree[i].level]++;
        }
```

```
    }
    for (int i = 0; i < maxLevel+1; i++)
    {
        if (i == 0)
            cout << result[i];
        else
            cout << ' ' << result[i];
    }
}
```

这段代码非常简洁，不到50行，乍一看看不出什么问题，于是我就输进评测系统测试了，但只是部分确。但实在是看不出问题在哪，于是我只好上网查查别人的方法。

网上找的方法基本思路和我一样，但有一个地方做了一个更保险的做法，就是在确定每个结点的时，是在第一遍读入数据的时候就开始确定的，而网上的方法则是读完数据以后重新遍历了一遍vector以定每个结点的层级，为此需要在node结构体中加上每个结点的父结点信息。我觉得问题就是出在这因为在一开始的时候我们只知道根节点所在层级。我的方法也是默认根节点是作为第一条数据读入的但如果第一条读入的数据并不是根节点，那么后续的结点层级都会发生错误，因为子结点的层级是父点的层级加1，而第一条读入的如果不是根节点的话我们就不知道父结点的层级。而网上的方法在读其他数据以后根据父结点信息重新从第一个结点也就是根结点开始确定层级，这样就保证了后续结点层级都能正确确定。我这样说可能不太清楚，我把借鉴了网上的思想之后重新修改的代码放在下面

代码

```
#include <iostream>
#include <vector>
using namespace std;
struct node
{
    int parent;
    int level;
    bool hasChild = false;
};
int main()
{
    int nodeSum, nonLeafSum;    //结点总数，非叶子结点总数
    cin >> nodeSum >> nonLeafSum;
    vector<node> tree(nodeSum+1);    //树，用双亲表示法
    tree[1].level = 0;
    tree[1].parent = -1;
    for (int i = 0; i < nonLeafSum; i++)
    {
        int parent;
        cin >> parent;
        tree[parent].hasChild = true;
        int childSum;
        cin >> childSum;
        for (int j = 0; j < childSum; j++)
        {
            int childNum;
            cin >> childNum;
            tree[childNum].parent = parent;
        }
    }
```

```
    int maxLevel = 0;    //存储最大层数
    for (int i = 1; i <= nodeSum; i++)
    {
        for (int j = 1; j <= nodeSum; j++)
        {
            if (tree[j].parent== i)
            {
                tree[j].level = tree[i].level + 1;
                if (tree[j].level > maxLevel)
                    maxLevel = tree[j].level;
            }
        }
    }
    vector<int> result(maxLevel+1,0);    //存储输出结果，即每层的叶子节点数
    for (int i = 1; i <= nodeSum; i++)
    {
        if (tree[i].hasChild == false)
        {
            result[tree[i].level]++;
        }
    }
    for (int i = 0; i < maxLevel+1; i++)
    {
        if (i == 0)
            cout << result[i];
        else
            cout << ' ' << result[i];
    }
}
```

补充知识——树的存储方法

通常有三种存储树的方法，分别是：1. 双亲表示法；2. 孩子表示法；3. 孩子兄弟表示法。关于各种表方法的介绍在这篇博客中介绍得很详细：https://www.cnblogs.com/ssyfj/p/9459887.html

本题最适合采用的就是双亲表示法，因为所有的结点都存储在一个数组中，非常容易实现遍历以记录一层的叶子结点数。但双亲表示法存在的问题就是很难遍历到一个结点的子结点。但本题中不需要遍子结点。