链滴

PAT 甲级刷题实录——1021

作者：aopstudio
原文链接：https://ld246.com/article/1580981351387
来源网站：链滴

原题链接

https://pintia.cn/problem-sets/994805342720868352/problems/994805482919673856

思路

这题实际上包含了两个问题，一个是判断给出的图是不是树，另一个是找出最深的根节点。

对于第一个问题，网上有一些做法是从任意结点开始深度遍历，看能否一次遍历完所有结点，相当于个连通图，如果不能，则总共需要遍历的次数即联通子图的个数。我的做法与此不同，在每次读入时断两个端点是否都已经被访问过，如果不是，则记录两个端点为已访问，如果是，说明图不是树，并将联通子图的个数加一。大家可以在纸上画一画看看是不是这样。

对于第二个问题，我的方法是对每个结点深度遍历，每下一层则记录的 depth 加 1，如果 depth 比 axDepth 大，则更新 maxDepth 为 depth。对每个结点计算出的 maxDepth 做比较，将最大的几结点插入结果集。这里有一个问题，一开始不知道最大的是几，有可能第一个就是最大了，所以插入点分为大于和等于两种情况。具体详见代码

遇到的问题是在第 3 个测试点时提示内存超限。参考了一下网上的方法，在不破坏自己其他算法的情下，将记录图的数据结构从邻接矩阵换成了邻接表。发现 C++ 实现不包含取值信息的邻接表其实非简单，和动态二维数组基本没什么区别，或者可以说是一种简化的动态二维数组。之前我们使用的是ector<vector <int> >,本质上可以看作是一个大数组，同时每个大数组元素代表一个小数组，这里每个小数组的大小是任意的，当我们想用它实现矩阵时，是人为地预先设置好小数组的大小，并且每小数组的大小都是相等的，而如果是邻接表，我们只要不设定小数组的大小，让它的初始大小为 0，后对每个结点对应的数组分别动态插入该结点联通的结点信息即可。

代码

```
#include <iostream>
#include <vector>

using namespace std;
vector<vector<int> > graph;

int maxDepth = 0;
vector<int> deepestRoot;
vector<int> reached;
int findDeepest(int root, int N, int depth);
int main()
{
    int N;
    cin >> N;
    graph.resize(N + 1);
    reached.assign(N + 1, 0);
    bool isTree = true;
    int connected = 1;
    for (int i = 0; i < N - 1; i++)
    {
        int a, b;
        cin >> a >> b;
        if (reached[a] == 1 && reached[b] == 1)    //两个点之前都已经访问过
        {
```

```
            isTree = false;
            connected++;
        }
        else
        {
            reached[a] = reached[b] = 1;
            graph[a].push_back(b);
            graph[b].push_back(a);
        }
    }
    if (isTree == false)    //不是树
        cout << "Error: " << connected << " components";
    else
    {
        int deepst = 0;
        for (int i = 1; i <= N; i++)
        {
            maxDepth = 0;
            reached.assign(N + 1, 0);
            int temp = findDeepest(i, N, 0);
            if (temp > deepst)
            {
                deepst = temp;
                deepestRoot.clear();
                deepestRoot.push_back(i);
            }
            else if (temp == deepst)
            {
                deepestRoot.push_back(i);
            }
        }
        for (int i = 0; i < deepestRoot.size(); i++)
        {
            cout << deepestRoot[i] << endl;
        }
    }
    return 0;
}
int findDeepest(int root, int N, int depth)
{
    reached[root] = 1;
    depth++;
    if (depth > maxDepth)
        maxDepth = depth;
    for (int i = 0; i < graph[root].size(); i++)
    {
        int t = graph[root][i];
        if (reached[t] == 0)    //有通路且没有被访问过
        {
            findDeepest(t, N, depth);
        }
    }
    return maxDepth;
}
```