图的邻接表+DFS写法

例题：1021 Deepest Root (25分)

1021 Deepest Root (25分)

A graph which is connected and acyclic can be considered a tree. The height of the tree depends on the selected root. Now you are supposed to find the root that results in a highest tree. Such a root is called **the deepest root**.

Input Specification:

Each input file contains one test case. For each case, the first line contains a positive integer *N* (≤104) which is the number of nodes, and hence the nodes are numbered from 1 to *N*. Then *N*−1 lines follow, each describes an edge by given the two adjacent nodes' numbers.

Output Specification:

For each test case, print each of the deepest roots in a line. If such a root is not unique, print them in increasing order of their numbers. In case that the given graph is not a tree, print `Error: K components` where `K` is the number of connected components in the graph.

Sample Input 1:

```
5
1 2
1 3
1 4
2 5
```

Sample Output 1:

```
3
4
5
```

Sample Input 2:

```
5
1 3
1 4
2 5
3 4
```

Sample Output 2:

```
Error: 2 components
```

思路

1. 由于我写了好几遍，一直是用邻接矩阵做的图的题，养成了思维定势。这时用邻接矩阵会出现超限的情况。因此我采用了邻接表的写法。不是变化特别多好像，还能节省空间。以后或许可以多采用邻接表写法。
2. 找到最深根的思路也很简单。
    - 首先随便找个点DFS，把最高点的root记录进数组。
    - 把数组的点insert进set集合里。
    - 从刚刚找到的其中一个最高点开始再做一次DFS，获得另一个最高root的点集合
    - 两个集合取并集，就能得到最高点的集合

邻接表写法

1. 邻接表graph

```
int number, maxDep = 0;
vector<vector<int>> e;
vector<bool> visit;
void init()
{
    cin >> number;
    e.resize(number + 1);
    visit.assign(number + 1, false);
    for(int i=1;i<number;i++)
    {
        int x, y;
        cin >> x >> y;
        e[x].push_back(y);
        e[y].push_back(x);
    }
}
```

2. 邻接表DFS

```
void dfs(int index, int dep)
{
    if (dep > maxDep)
    {
        maxDep = dep;
        otherAns.clear();
    }
```

```
    if (dep == maxDep)
        otherAns.push_back(index);
    visit[index] = true;
    for (int i = 0; i < e[index].size(); i++)
    {
        if (visit[e[index][i]] == false)
            dfs(e[index][i], dep + 1);
    }
}
```

代码参考

```
#include <set>
#include <vector>
#include <iostream>
using namespace std;
int number, maxDep = 0;
vector<vector<int>> e;
vector<bool> visit;
set<int> ans;
vector<int> otherAns;
void init()
{
    cin >> number;
    e.resize(number + 1);
    visit.assign(number + 1, false);
    for(int i=1;i<number;i++)
    {
        int x, y;
        cin >> x >> y;
        e[x].push_back(y);
        e[y].push_back(x);
    }
}
void dfs(int index, int dep)
{
    if (dep > maxDep)
    {
        maxDep = dep;
        otherAns.clear();
    }
    if (dep == maxDep)
        otherAns.push_back(index);
    visit[index] = true;
    for (int i = 0; i < e[index].size(); i++)
    {
        if (visit[e[index][i]] == false)
            dfs(e[index][i], dep + 1);
    }
}
int main()
{
    init();
    int count = 0;
    for(int i=1;i<=number;i++)
    {
```

```
        if (visit[i] == false)
        {
            dfs(i, 0);
            count++;
        }
    }
    if(count>1)
    {
        printf("Error: %d components", count);
        return 0;
    }
    visit.assign(number + 1, false);
    ans.insert(otherAns.begin(), otherAns.end());
    dfs(*otherAns.begin(), 0);
    ans.insert(otherAns.begin(), otherAns.end());
    for (auto it = ans.begin(); it != ans.end(); it++)
        cout << *it << endl;
    return 0;
}
```