1.在层次优先搜索时，有时会在队列循环外增设一个变量，来记录当前状态，一旦当前队首元素与当前状态不同，就更新当前状态，并做相应处理。这种思路最大的问题在于，往往最后一个状态得不到处理。因此要在队列循环之后单独处理，这很重要，一定要考虑到。

2.无。

1004. Counting Leaves (30)

Input

Each input file contains one test case. Each case starts with a line containing 0 < N < 100, the number of nodes in a tree, and M (< N), the number of non-leaf nodes. Then M lines follow, each in the format:

```
ID K ID[1] ID[2] ... ID[K]
```

Output

For each test case, you are supposed to count those family members who have no child **for every seniority level** starting from the root. The numbers must be printed in a line, separated by a space, and there must be no extra space at the end of each line.

The sample case represents a tree with only 2 nodes, where 01 is the root and 02 is its only child. Hence on the root 01 level, there is 0 leaf node; and on the next level, there is 1 leaf node. Then we should output "0 1" in a line.

Sample Input

```
2 1
01 1 02
```

Sample Output

```
0 1
```

- Edit 0:

```
#include <cstdio>
#include <cstring>

#include <cmath>
```

```
#include <vector>
#include <algorithm>
using std::max;
using std::vector;


const int maxn=110;

vector<int> nodes[maxn];

int leaves[maxn]={0};

int max_depth=0;

void dfs(int index,int depth){
    max_depth=max(max_depth,depth);
    if(nodes[index].size()==0){
        ++leaves[depth];
        return;
    }
    else{
        for(int i=0;i<(int)nodes[index].size();++i){
            dfs(nodes[index][i],depth+1);
        }
    }
}

int main(){
    int nodes_num=0,non_leaf_nodes=0;
    scanf("%d %d",&nodes_num,&non_leaf_nodes);
    int count;
    int index;
    int value;
    for(int i=0;i<non_leaf_nodes;++i){
        scanf("%d %d",&index,&count);
        for(int j=0;j<count;++j){
            scanf("%d",&value);
            nodes[index].push_back(value);
        }
    }
    dfs(1,1);
    for(int i=1;i<=max_depth;++i){
        printf("%d",leaves[i]);
        if(i!=max_depth){
            printf(" ");
        }
    }
    return 0;
}
```

- Edit 1:

```
#include <iostream>
```

```cpp
#include <string>
#include <queue>
#include <vector>

using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::queue;
using std::vector;
using std::ios;

struct Node{
    vector<int> children;
    int deep;
    Node():deep(0){}
};

vector<Node> tree(110);

vector<int> get_leaf(vector<Node> &tree,int root){
    queue<Node*> que;
    tree[root].deep=1;
    int nowDeep=1;
    int counter=0;
    vector<int> ans;
    que.push(&tree[root]);
    Node* front;
    while(!que.empty()){
        front=que.front();
        que.pop();
        if(front->deep!=nowDeep){
            ans.push_back(counter);
            counter=0;
            nowDeep=front->deep;
        }
        if(front->children.size()<=0){
            ++counter;
        }
        for(int i=0;i<(int)front->children.size();++i){
            tree[front->children[i]].deep=front->deep+1;
            que.push(&tree[front->children[i]]);
        }
    }
    ans.push_back(counter);
    return ans;
}

int main(){
    ios::sync_with_stdio(false);
    int node_num,nonleaf_num;
    cin>>node_num>>nonleaf_num;

    int temp;
```

```
    int id;
    int child;
    for(int i=0;i<nonleaf_num;++i){
        cin>>id>>temp;
        for(int j=0;j<temp;++j){
            cin>>child;
            tree[id].children.push_back(child);
        }
    }
    vector<int> ans=get_leaf(tree,1);
    for(int i=0;i<(int)ans.size();++i){
        cout<<ans[i];
        if(i!=(int)ans.size()-1){
            cout<<" ";
        }
        else{
            cout<<endl;
        }
    }
    return 0;
}
```