☑ PAT 甲级题目讲解：1004《Counting Leaves》

🧩 题目简介

本题要求从给定的族谱树结构中，逐层统计每一层中**没有孩子的节点（即叶子节点）个数**，并按照层级从上到下输出。

题目中：

- 树的根节点固定为编号 `01`；
- 输入采用非叶子节点及其所有子节点编号的形式；
- 最终输出从根节点出发每一层的叶子节点数，按层次顺序打印。

📏 样例分析

输入：

```
2 1
01 1 02
```

解释：

- 有两个节点，非叶子节点有一个，即 `01`，它有一个孩子 `02`；
- 根节点 `01` 属于第 1 层，但它是非叶节点；
- 节点 `02` 没有孩子，是第 2 层的叶子节点。

输出：

```
0 1
```

表示：

- 第 1 层没有叶子节点；
- 第 2 层有 1 个叶子节点。

🔍 解题思路

整体思路

本题可以建树后使用 **广度优先搜索（BFS）** 从根节点开始一层层向下遍历。

每一层中，如果某个节点没有孩子，则为叶子结点，在当前层计数。

🧬 变量说明

变量名	含义
`n`	总结点数
`m`	非叶节点数量
`cnt[i]`	节点 `i` 的孩子数量
`t[i][j]`	节点 `i` 的第 `j` 个孩子编号
`s[k]`	第 `k` 层的叶子节点数
`k`	层数总数（用于最终输出）

☑ Step 1：建树结构（使用邻接表）

🌳 建树思路

- 输入中每一行表示一个非叶子节点及其所有孩子；
  - 使用二维数组 `t[i][j]` 存储节点 `i` 的第 `j` 个孩子编号；
  - 另设数组 `cnt[i]` 存储每个节点的孩子个数；
  - 则要遍历一棵树所有孩子可以用：

```
for(int i = 1; i <= cnt[p]; i++){ // p 有 cnt[p] 个孩子
    int cd = t[p][i];  // cd: 结点 p 的第 i 个孩子编号
}
```

- 根节点编号固定为 `01`，可直接以整数 `1` 表示。

输入时用二维数组建树：

```
scanf("%d %d", &n, &m);
while(m--){
    int f, k, c;
    scanf("%d %d", &f, &k);
    while(k--){
        scanf("%d", &c);
        t[f][++cnt[f]] = c; // 记录孩子节点编号
    }
}
```

☑ Step 2：层序遍历统计每层叶子数

- 使用队列 `queue<int>` 按层推进；
- 每一层开始时记录当前队列大小 `size`，代表该层节点数；
- 逐层遍历队列中现有结点：
  - 每次从队首弹出节点 `p`，判断其是否为叶子（`cnt[p] == 0`），若是，则 `s[k]++`；
  - 将 `p` 的所有孩子依次入队；
- 每层处理结束后层数 `k++`；
- 最后一次循环后队列为空，但 `k` 多加了一次，因此需 `k--` 纠正。

```
void bfs(int x){
    queue<int> q;
    q.push(x);
    k = 1; // 初始化第一层为 1
    while(!q.empty()){
        int d = q.size(); // 当前层节点个数
        for(int i = 1; i <= d; i++){
            int p = q.front(); q.pop();
            if(!cnt[p]) s[k]++; // 没有孩子就是叶子节点
            for(int i = 1; i <= cnt[p]; i++){
                int cd = t[p][i];
                q.push(cd);
            }
        }
        ++k; // 进入下一层
    }
    k--; // 减去最后多加的一层
}
```

☑ 完整代码

```
#include <bits/stdc++.h>
using namespace std;

int n, m, cnt[105], t[105][105], k, s[105];

void bfs(int x){
    queue<int> q;
    q.push(x);
    k = 1; // 初始化第一层
    while(!q.empty()){
        int d = q.size(); // 当前层的结点数
        for(int i = 1; i <= d; i++){
            int p = q.front(); q.pop();
            if(!cnt[p]) s[k]++; // 当前为叶子节点
            for(int i = 1; i <= cnt[p]; i++){
                int cd = t[p][i]; // 当前 p 的一个孩子
                q.push(cd); // 孩子入队
            }
        }
        ++k; // 层数加 1
    }
    k--; // 减去最后空的那层
}

int main(){
    scanf("%d %d", &n, &m);
    while(m--){
        int f, k, c;
        scanf("%d %d", &f, &k);
        while(k--){
            scanf("%d", &c);
            t[f][++cnt[f]] = c; // 建边
        }
    }
```

```
    bfs(1); // 从根节点 1 开始 BFS
    for(int i = 1; i <= k; i++){
        printf("%d", s[i]);
        if(i < k) printf(" ");
    }
    return 0;
}
```

常见错误提醒

错误类型	表现描述
忘记减去最后空层	BFS 最后一层已空但仍加了 `k`，导致多输出一层
queue 大小变化误用 `q.size()`	在 `for` 内部使用 `q.size()` 会因 `push()` 改变队列大小，造成统计错误
没有判断叶子节点	忘记 `if(!cnt[p])` 会导致漏计

总结归纳

- 本题重点是**树的建模 + 层序遍历**；
- 熟练掌握使用数组模拟邻接表的方法；
- 每轮处理固定数量节点，再推进一层；
- 注意边界处理和输出格式控制。
- **时间复杂度**：
  - 建树：$\mathcal{O}(n)$
  - BFS 遍历：$\mathcal{O}(n)$
- **空间复杂度**：
  - $\mathcal{O}(n)$

思维拓展

- 若题目改为输出每层所有结点编号，可在 BFS 中使用 `depth[]` 记录每个结点层数；
- 类似模型常见于 公司管理结构、操作系统树状调度、层级打印可视化 等问题；
- 进一步练习可尝试：输出树的最大深度、树的直径、从叶子反向建树等变形题。