剑指55 二叉树的深度

关键点：

树的后序遍历 / 深度优先搜索往往利用 递归 或 栈 实现。

树的深度 = 左子树的深度与右子树的深度中的最大值 + 1

```
// 3分钟完成。思路是遍历过程中，获知每个node的深度，同时不断更新maxdepth。

class Solution {
public:
    int m_depth;
    void DFS(TreeNode* root, int k) {
        if (!root) return;
        k++;
        DFS(root->left, k);
        DFS(root->right, k);
        m_depth = k > m_depth ? k : m_depth;
    }
    int maxDepth(TreeNode* root) {
        m_depth = 0;
        DFS(root, 0);
        return m_depth;
    }
};

// 优化：树的深度 = 左子树的深度与右子树的深度中的最大值 + 1
// ！！！！！  思路简洁 ！！！！！！

class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) return 0;
        return max(maxDepth(root->left), maxDepth(root->right)) + 1;
    }
};


@Krahets K神，方法一是动态规划的思想吗？感觉像是动态规划，这道题有点像类似于爬楼梯、青蛙跳台阶。
@lelelong 哈喽，很好的类比~ 如果把子树的深度看作是子问题的话，是类似动态规划的思想~


```

```
其他：BFS方法，略慢。

class Solution {
public:
    int maxDepth(TreeNode* root) {
        if (!root) return 0;
```

```
        queue <TreeNode*> nodes;
        nodes.push(root);
        TreeNode* q = root;
        int depth = 0;
        while (!nodes.empty()) {
            int size = nodes.size();
            for (int i = 0; i < size; i++) {
                q = nodes.front();
                nodes.pop();
                if (q->left) nodes.push(q->left);
                if (q->right) nodes.push(q->right);
            }
            depth++;
        }
        return depth;
    }
};
```