（1.7）LC111.二叉树的最小深度

111. 二叉树的最小深度 - 力扣（LeetCode）

这里强调一下深度，是从下往上算的，当只有一个结点时，深度为1。

而高度，是从根节点往下算的，同样当只有root时高为1。

这道题在先做过二叉树的最大深度以后，很容易惯性思维做错。

二叉树的最大深度代码：

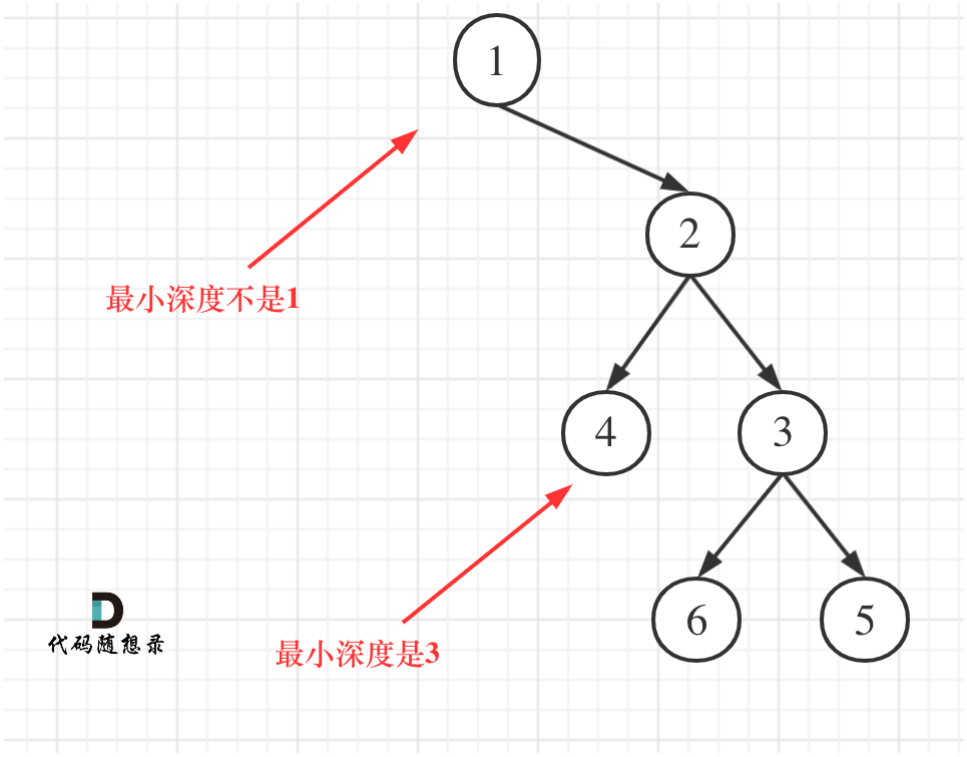
```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root == NULL) return 0;
        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);
        return leftDepth > rightDepth ? 1 + leftDepth : 1 + rightDepth;
    }
}
```

由此会写出二叉树的最小深度代码：

```
return leftDepth < rightDepth ? 1 + leftDepth : 1 + rightDepth;
//如果这段代码用在下图答案就为1了
//所以说上面这个代码使用的条件是，左右子树均不为空
//那么碰到左子树或者右子树为空的情况呢？
//以下图为例，左子树为空时，根节点的mindepth是mindepth(root->right) + 1;
```

然而是大错特错。

一个经典的错误例子：



法一：递归

```
class Solution {
public:
    int minDepth(TreeNode* root) {
        if(root == NULL) return 0;  //空结点返回0
        if(root->left == NULL && root->right != NULL){
            return 1 + minDepth(root->right);
        }
        if(root->left != NULL && root->right == NULL){
            return 1 + minDepth(root->left);
        }

        int res = 1 + min(minDepth(root->left),minDepth(root->right));
        return res;

    }
};
```

法二：层序遍历

这道题层序遍历也能做，前提是由上到下由左到右。

方法是，遍历到的第一个叶子结点即为二叉树的最小深度

```
class Solution {
public:
    int minDepth(TreeNode* root) {
```

```
        queue<TreeNode*> que;
        if(root == NULL) return 0;
        else que.push(root);
        int depth = 0;

        while(!que.empty()){
            int size = que.size();
            depth++;
            for(int i = 0;i < size;i++){
                TreeNode* tmp = que.front();
                que.pop();

                if(tmp->left) que.push(tmp->left);
                if(tmp->right) que.push(tmp->right);
                if(!tmp->left && !tmp->right) return depth;
            }
        }
        return depth;

    }
};
```