又是一道"简单题"?

101. 对称二叉树

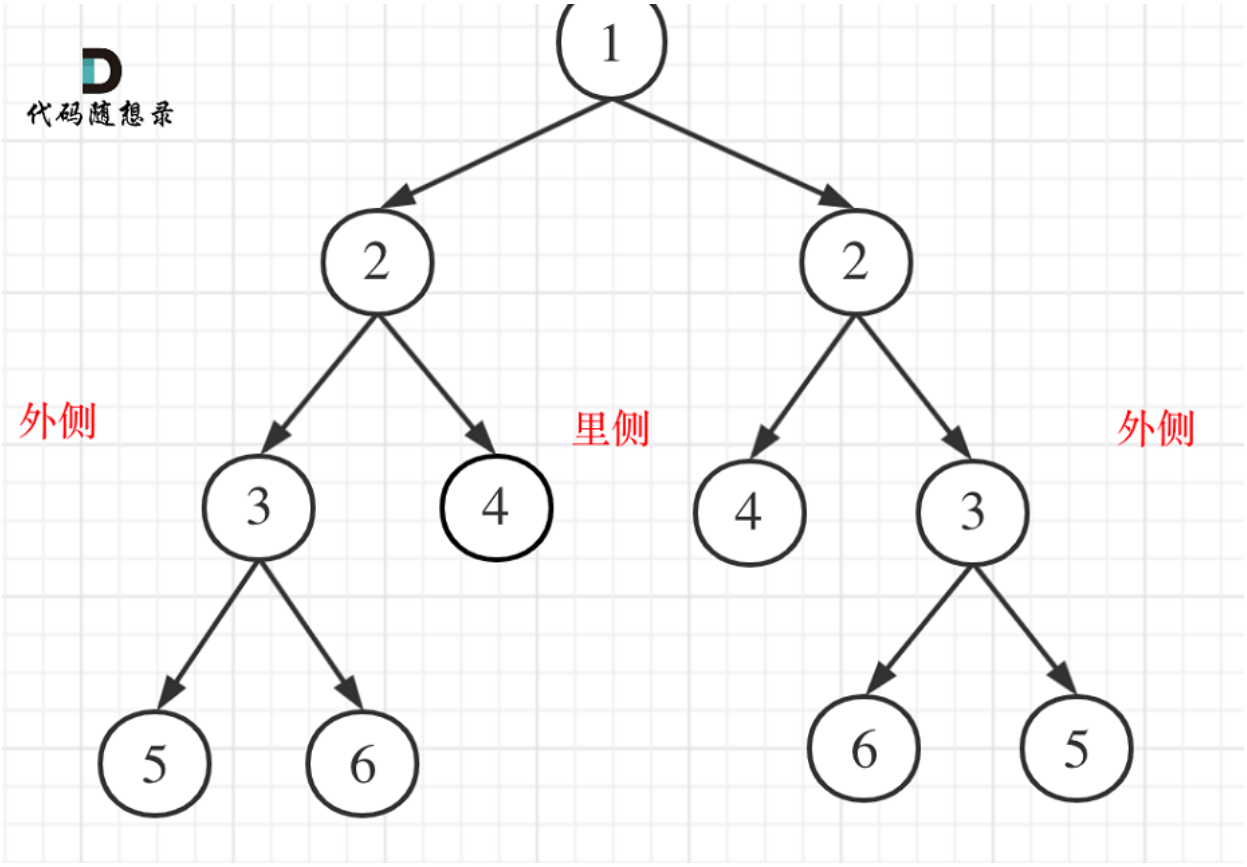
给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树 `[1,2,2,3,4,4,3]` 是对称的。

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

> 方法(1)可以通过层次遍历算法求得每一层的结点，（用一个数组记录每一层的结点）
> 此时就要清楚什么是镜像对称：
> 对记录每一层结点的数组用双指针遍历(类似于字符串反转)，比较如果有一个不满足条件就return false
> **分析:这种方法不行，此时只有考虑每层结点数值的对称性没有考虑到结点的位置对称性**

正确的分析方法



本题的遍历采用**“后序遍历”**算法，
因为我们要通过**递归函数的返回值**来判断**两个子树的内侧结点和外侧结点**是否相等
正是因为要遍历两棵树而且要比较内侧和外侧结点，所以准确来说是**一个树的遍历顺序是左右中，另一个树的遍历顺序是右左中**

递归方法

1、确定递归函数的参数和返回值
因为要比较**根节点的两个子树**是否相互翻转的，
bool compare(TreeNode* left,TreeNode* right);

2、确定终止条件

```
左节点为空，右节点不为空 return false;
左节点不为空，右结点不为空 return false;
左节点和右结点均不为空，且左右节点的数值不相等，return false
```

3、确定单层递归的逻辑
此时确定的是**左右节点均不为空，且数值相等相同的情况**

代码如下：

```
bool outside = compare(left->left, right->right);   // 左子树：左、
bool inside = compare(left->right, right->left);    // 左子树：右、
bool isSame = outside && inside;                    // 左子树：中、
return isSame;
```

如上代码中，我们可以看出使用的遍历方式，左子树左右中，右子树右左中，所以我把这个遍历顺序也称之为“后序遍历”（尽管不是严格的后序遍历）。

```
class Solution{
 public:
 //left,right的父节点不是相同的
 bool compare(TreeNode* left,TreeNode* right){
 if(left==NULL&&right!=NULL) return false;
 else if(left!=NULL&&right==NULL) return false;
 else if(left->val!=right->val) return false;
 //接下就是左右节点的不为空,且数值不相等
 //思路是左边子树的遍历顺序是左右中，右边子树的遍历顺序是右左中
  return compare(left->left,right->right)&&
  compare(left->right,right->left);

```

```
  }
  bool isSymmetric(TreeNode* root){
  if(root==NULL) return true;
  return compare(root->left,root->right);
  }
}
```

迭代法

我们这里可以使用队列来比较**(根节点的左右子树)**是否**相互翻转，其实此处用栈也可以实现（代码一样就是把队列定义改为栈定义）**

队列中里面每两个元素加入队列，每两个元素出队列

```
class Solution {
public:
    bool isSymmetric(TreeNode* root) {
  queue<TreeNode*> que;
  if(root==NULL) return true;
  que.push(root->left);
  que.push(root->right);
  while(!que.empty()){
  TreeNode* left=que.front();que.pop();
   TreeNode* right=que.front();que.pop();
   if(!left&&!right) continue;
   if(left==NULL||right==NULL||left->val!=right->val) return false;
   //加入队列依次顺序：左树左节点，右树右节点，左树右节点，右树左节点
   que.push(left->left);
   que.push(right->right);
   que.push(left->right);
   que.push(right->left);
   }
   return true;
    }
};
```

二叉树的最大深度会求了，那么顺手把N叉树也做了

104.二叉树的最大深度

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例：
给定二叉树 [3,9,20,null,null,15,7]，

```
    3
   / \
  9  20
    /  \
   15   7
```

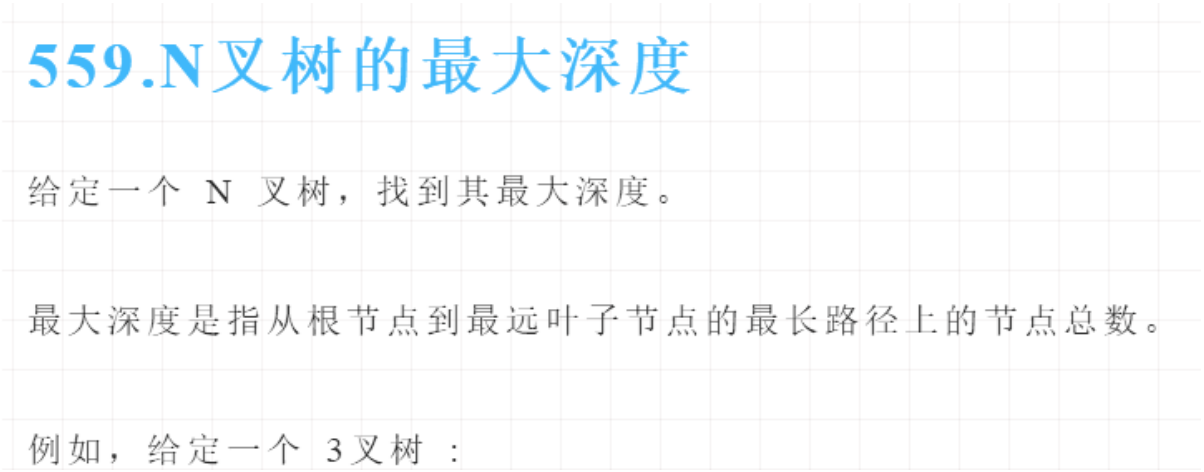
返回它的最大深度 3 。

层次遍历算法

```
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root==NULL) return 0;
        queue<TreeNode*> que;
        int ans=0;
        que.push(root);
        while(!que.empty()){
            int size=que.size();
            for(int i=0;i<size;i++){
                TreeNode* node=que.front();
                que.pop();
                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
            ans++;
        }
```

```
        return ans;
    }
};
前序遍历算法
class Solution {
public:
    int preSearch(TreeNode* root){
        if(root==NULL) return 0;
        return max(preSearch(root->left),preSearch(root->right))+1;
    }
    int maxDepth(TreeNode* root) {
       if(root==NULL) return 0;
       else return preSearch(root);
    }
};
```

思考一下N叉树的最大深度



与二叉树的最大深度一个套路?

111.二叉树的最小深度

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明：叶子节点是指没有子节点的节点。

示例：

给定二叉树 [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

返回它的最小深度 2.

最小深度是从根节点到最近叶子节点的最短路径上的节点数量
思路的分析：
输出的变量为：minx
利用层次遍历算法，在每次进入循环时都对minx自加，在对节点node进行拓展时出现
if(!node->left&&!node->right)return minx;//说明此时的层数即为最小深度

```
class Solution {
public:
    int minDepth(TreeNode* root) {
 queue<TreeNode*> que;
 int depth=0;
 if(root==NULL)return depth;
 que.push(root);
 while(!que.empty()){
 int size=que.size();
  depth++;
  for(int i=0;i<size;i++){
  TreeNode* node=que.top();
  que.pop();
  if(!(node->left)&&!(node->right)) return ans;
```

```
 if(node->left)que.push(node->left);
 if(node->right)que.push(node->right);
 }
 }
 return ans;
    }
};
```