注意：递归三步以及由局部到整体的思想

二叉树的三种遍历：

- 先序遍历（根左右）

```python
class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def help(root):
            if not root:
                return
            res.append(root.val)
            help(root.left)
            help(root.right)
        help(root)
        return res
```

- 中序遍历（左根右）

```python
class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def help(root):
            if not root:
                return
            help(root.left)
            res.append(root.val)
            help(root.right)
        help(root)
        return res
```

- 后序遍历（左右根）

```python
class Solution:
    def preorderTraversal(self, root: TreeNode) -> List[int]:
        res = []
        def help(root):
            if not root:
                return
            help(root.left)
            help(root.right)
            res.append(root.val)
        help(root)
        return res
```

1.平衡二叉树

- 解法1：自顶向下方法，思路是构造一个获取当前节点最大深度的方法 depth(root) ，通过比较此子树的左右子树的最大高度差 abs(height(root.left) - height(root.right))，来判断此子树是否是二叉平衡树。若树的所有子树都平衡时，此树才平衡。

```python
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        if not root:
```

```
            # 一直递归，直到根节点为空，返回true
            return True
        # 从上往下，判断当前左右节点差是否小于2，并且左右节点向下递归
        return abs(self.height(root.left) - self.height(root.right)) < 2 and self.isBal

    def height(self, root):
        if not root:
            return False
        # 从下往上递归，获取当前节点的深度
        return max(self.height(root.left), self.height(root.right)) + 1
```

- 解法2：从底至顶，思路是对二叉树做先序遍历，从底至顶返回子树最大高度，若判定某子树不是平衡树则“剪枝”，直接向上返回。

```
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        return self.help(root) != -1

    def help(self, root):
        # 利用前序遍历方法，找到每个根节点的最大左右节点做差
        if not root:
            return 0
        left = self.help(root.left)
        if left == -1:
            return -1
        right = self.help(root.right)
        if right == -1:
            return -1
        return max(left, right) + 1 if abs(left - right) < 2 else -1
```

2.翻转二叉树

- 递归思想，由最小的单元开始交换

```
class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        if not root:
            return
        tmp = self.invertTree(root.left)
        root.left = self.invertTree(root.right)
        root.right = tmp
        return root
    # 简化
    def invertTree(self, root: TreeNode) -> TreeNode:
        if not root:
            return
        root.left, root.right = self.invertTree(root.right), self.invertTree(root.left
        return root
```

3.N 叉树的前序遍历

- 解法1：递归,与二叉树相同，循环子节点即可。

```python
class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        res = []
        def help(root):
            if not root:
                return
            res.append(root.val)
            if not root.children:
                return
             for child in root.children:
                 help(child)
         help(root)
         return res
```

- 解法2：迭代,利用队列遍历节点。

```python
class Solution:
    def preorder(self, root: 'Node') -> List[int]:
        if not root:
            return
        result = []
        queue = collections.deque()
        queue.append(root)
        while queue:
            node = queue.pop()
             result.append(node.val)
             for child in node.children[::-1]:
                 queue.append(child)
         return result
```

4.从上到下打印二叉树 II

- BFS方法，把每层每层放入队列

```python
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        result = []
        # 双端队列
        queue = collections.deque()
        queue.append(root)
        while queue:
             # 创建临时列表存储每行的数据
             tmp = []
             for _ in range(len(queue)):
                 node = queue.popleft()
                 tmp.append(node.val)
                 # 接下来判断是否有子节点
```

```
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
            result.append(tmp)
        return result
```

5.二叉树的锯齿形层序遍历

- 类似第三题，把结果奇数的列表倒序即可

```
class Solution:
    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        res = []
        flag = False
        queue = collections.deque()
        queue.append(root)
        while queue:
            tmp = []
            for _ in range(len(queue)):
                node = queue.popleft()
                tmp.append(node.val)
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
            if flag:
                tmp = tmp[::-1]
            flag = not flag
            res.append(tmp)
        return res
```

6.二叉树的层序遍历 II

- 思想与第三题相同，再最后存入的时候使用双端队列向左存入即可。

```
class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        result = collections.deque()
        queue = collections.deque()
        queue.append(root)
        while queue:
            tmp = []
            for _ in range(len(queue)):
                node = queue.popleft()
                tmp.append(node.val)
```

```
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)
            result.appendleft(tmp)
        result = list(result)
        return result
```

7.二叉树最大宽度

- 利用层序遍历，给每个节点打上编号，把每层节点对应的编号存到一个临时列表中，最后比较每个临时列表最大与最小的差值，找出最大宽度

```
class Solution:
    def widthOfBinaryTree(self, root: TreeNode) -> int:
        if not root:
            return
        res = []
        result = []
        queue = collections.deque()
        queue.append([root, 1])
        while queue:
            tmp = []
            for _ in range(len(queue)):
                node, index = queue.popleft()
                tmp.append(index)
                if node.left:
                    queue.append([node.left, index * 2])
                if node .right:
                    queue.append([node.right, index * 2 + 1])
            res.append(tmp)
        print(res)
        for r in res:
            result.append(r[-1] - r[0] + 1)
        return max(result)
```

8.完全二叉树的节点个数

- 递归解决

```
class Solution:
    def countNodes(self, root: TreeNode) -> int:
        if not root:
            return 0
        left = self.countNodes(root.left)
        right = self.countNodes(root.right)
        return left + right + 1
```

9.树的子结构

- 递归，找到A与B相同的节点，然后向下遍历；再根据A的左右节点分别去与B对应

```python
class Solution:
    def Matchingtree(self, A, B):
        # 当B节点为空时，说明B已匹配完成
        if not B:
            return True
        if not A or A.val != B.val:
            return False
        return self.Matchingtree(A.left, B.left) and self.Matchingtree(A.right, B.right

    def isSubStructure(self, A: TreeNode, B: TreeNode) -> bool:
        if not A or not B:
            return False
        # 递归A与B，再将A的左节点与右节点与B的根节点比较，以此类推。
        return self.Matchingtree(A, B) or self.isSubStructure(A.left, B) or self.isSub
```

10.路径总和

- 利用层序遍历，向队列存入节点与其对应的值，每次遍历将值累加，知道叶子节点，判断有没有与目标参数对应的数值。

```python
class Solution:
    def hasPathSum(self, root: TreeNode, targetSum: int) -> bool:
        if not root:
            return False
        result = []
        queue = collections.deque()
        queue.append([root, root.val])
        while queue:
            node, path = queue.popleft()
            if not node.left and not node.right and path == targetSum:
                return True
            if node.left:
                queue.append([node.left, path + node.left.val])
            if node.right:
                queue.append([node.right, path + node.right.val])
        return False
```

11.二叉搜索树的第k大节点

- 中序遍历的倒序，二叉搜索树的中序遍历为正序

> 二叉搜索树：若它的左子树不空，则左子树上所有结点的值均小于它的根节点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为二叉搜索树

```python
class Solution:
    def kthLargest(self, root: TreeNode, k: int) -> int:
        result = []
        if not root:
            return
        def get_help(root):
            if not root:
                return
```

```
            get_help(root.right)
             result.append(root.val)
             get_help(root.left)
        get_help(root)
        return result[k-1]
```

- 利用计数的方式，递归到等于k的时候，返回当前的值

```
class Solution:
    def kthLargest(self, root: TreeNode, k: int) -> int:
        def get_help(root):
            if not root:
                return
            get_help(root.right)
            self.count += 1
            if self.count == k:
                self.data = root.val
                 return
             get_help(root.left)
             return
        self.count = 0
        self.data = 0
        get_help(root)
        return self.data
```

12.从前序与中序遍历序列构造二叉树

- 先根据前序遍历找到根节点的值放入新建的数中，然后根据根节点的值获取中序遍历根节点的索引，此时索引左侧为二叉树的左子树，右侧为二叉树的右子树，最后递归左右子树。

```
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        if not preorder and not inorder:
            return
        root = TreeNode(preorder[0])
        mid_index = inorder.index(preorder[0])
        root.left = self.buildTree(preorder[1:mid_index+1], inorder[:mid_index])
        root.right = self.buildTree(preorder[mid_index+1:], inorder[mid_index+1:])
        return root
```