后序遍历--迭代算法

```
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:

        # 使用栈辅助实现迭代算法进行后续遍历
        if not root: return []
        stack, res = [], []
        prev = None
        while root or stack:
            while root:
                stack.append(root)
                root = root.left
            root = stack.pop()

            # 后续遍历的特殊之处：顺序是左右中
            # 因此当该节点存在右节点时，将该节点再次押入栈中，搜索右节点
            # root.right == prev 该条件说明root右节点已经加入res中
            if not root.right or root.right == prev:
                res.append(root.val)
                prev, root = root, None
            else:
                stack.append(root)
                root = root.right
        return res
```

遍历

```
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root: return []
        #根结点入queue
        queue = [root]
        res = []
        while queue:
            res.append([node.val for node in queue])
            #存储当前层的孩子节点列表
            ll = []
            #对当前层的每个节点遍历
            for node in queue:
                #如果左子节点存在，入队列
                if node.left:
                    ll.append(node.left)
                #如果右子节点存在，入队列
                if node.right:
                    ll.append(node.right)
            #后把queue更新成下一层的结点，继续遍历下一层
            queue = ll
        return res
```