# 0145. 二叉树的后序遍历

ITCharge 大约 3 分钟

- 标签：栈、树、深度优先搜索、二叉树
- 难度：简单

## 题目链接

- 0145. 二叉树的后序遍历 - 力扣

## 题目大意

**描述**：给定一个二叉树的根节点 `root` 。

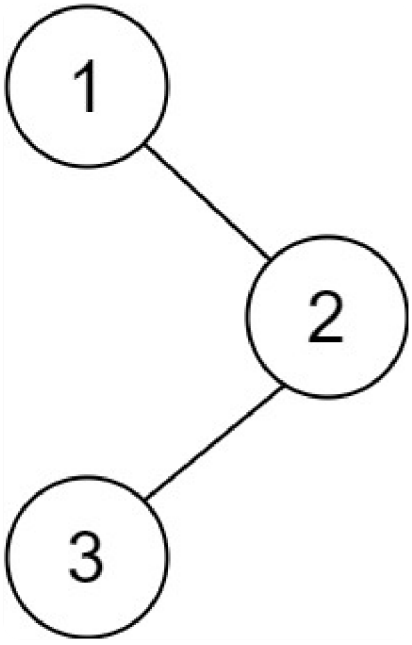
**要求**：返回该二叉树的后序遍历结果。

**说明**：

- 树中节点数目在范围 $[0, 100]$ 内。
- $-100 \leq Node.val \leq 100$。

**示例**：

- 示例 1：



img

```py
输入：root = [1,null,2,3]
输出：[3,2,1]
```

- 示例 2：

```py
输入：root = []
输出：[]
```

# 解题思路

## 思路 1：递归遍历

二叉树的后序遍历递归实现步骤为：

1. 判断二叉树是否为空，为空则直接返回。
2. 先递归遍历左子树。
3. 然后递归遍历右子树。
4. 最后访问根节点。

## 思路 1：代码

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

        postorder(root)
        return res
```

## 思路 1：复杂度分析

- **时间复杂度**：$O(n)$。其中 $n$ 是二叉树的节点数目。
- **空间复杂度**：$O(n)$。

## 思路 2：模拟栈迭代遍历

我们可以使用一个显式栈 `stack` 来模拟二叉树的后序遍历递归的过程。

与前序、中序遍历不同，在后序遍历中，根节点的访问要放在左右子树访问之后。因此，我们要保证：**在左右孩子节点访问结束之前，当前节点不能提前出栈。**

我们应该从根节点开始，先将根节点放入栈中，然后依次遍历左子树，不断将当前子树的根节点放入栈中，直到遍历到左子树最左侧的那个节点，从栈中弹出该元素，并判断该元素的右子树是否已经访问完毕，如果访问完毕，则访问该元素。如果未访问完毕，则访问该元素的右子树。

二叉树的后序遍历显式栈实现步骤如下：

1. 判断二叉树是否为空，为空则直接 |。
2. 初始化维护一个空栈，使用 `prev` 保存前一个访问的节点，用于确定当前节点的右子树是否访问完毕。
3. 当根节点或者栈不为空时，从当前节点开始：
   1. 如果当前节点有左子树，则不断遍历左子树，并将当前根节点压入栈中。
   2. 如果当前节点无左子树，则弹出栈顶元素 `node` 。
   3. 如果栈顶元素 `node` 无右子树（即 `not node.right` ）或者右子树已经访问完毕（即 `node.right == prev` ），则访问该元素，然后记录前一节点，并将当前节点标记为空节点。
   4. 如果栈顶元素有右子树，则将栈顶元素重新压入栈中，继续访问栈顶元素的右子树。

## 思路 2：代码

```py
class Solution:
    def postorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        res = []
        stack = []
        prev = None                    # 保存前一个访问的节点，用于确定当前节点的右子
```

```
树是否访问完毕

        while root or stack:        # 根节点或栈不为空
            while root:
                stack.append(root)  # 将当前树的根节点入栈
                root = root.left    # 继续访问左子树，找到最左侧节点

            node = stack.pop()      # 遍历到最左侧，当前节点无左子树时，将最左侧节点弹出

            # 如果当前节点无右子树或者右子树访问完毕
            if not node.right or node.right == prev:
                res.append(node.val)# 访问该节点
                prev = node         # 记录前一节点
                root = None         # 将当前根节点标记为空
            else:
                stack.append(node)  # 右子树尚未访问完毕，将当前节点重新压回栈中
                root = node.right   # 继续访问右子树

        return res
```

## 思路 2：复杂度分析

- **时间复杂度**：$O(n)$。其中 $n$ 是二叉树的节点数目。
- **空间复杂度**：$O(n)$。