# 题目

给定一个二叉树，返回它的 后序 遍历。

**示例:**

输入: [1,null,2,3]

1

\

2

/

3

输出: [3,2,1]

进阶: 递归算法很简单，你可以通过迭代算法完成吗？

# 分析

## 方法一：递归法

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

private:

vector<int> val;

public:

vector<int> postorderTraversal(TreeNode\* root) {

if (NULL == root)

return val;

postorderTraversal(root->left);

postorderTraversal(root->right);

val.push\_back(root->val);

return val;

}

};

另一种写法：

class Solution {

public:

void postorder(TreeNode \*root, vector<int> &res) {

if (root == nullptr) {

return;

}

postorder(root->left, res);

postorder(root->right, res);

res.push\_back(root->val);

}

vector<int> postorderTraversal(TreeNode \*root) {

vector<int> res;

postorder(root, res);

return res;

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是二叉搜索树的节点数。每一个节点恰好被遍历一次。

空间复杂度：O(n)，为递归过程中栈的开销，平均情况下为O(log n)，最坏情况下树呈现链状，为O(n)。

## 方法二：迭代法

class Solution {

public:

vector<int> postorderTraversal(TreeNode \*root) {

vector<int> res;

if (root == nullptr) {

return res;

}

stack<TreeNode \*> stk;

TreeNode \*prev = nullptr;

while (root != nullptr || !stk.empty()) {

while (root != nullptr) {

stk.emplace(root);

root = root->left;

}

root = stk.top();

stk.pop();

if (root->right == nullptr || root->right == prev) {

res.emplace\_back(root->val);

prev = root;

root = nullptr;

} else {

stk.emplace(root);

root = root->right;

}

}

return res;

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是二叉搜索树的节点数。每一个节点恰好被遍历一次。

空间复杂度：O(n)，为迭代过程中显式栈的开销，平均情况下为O(logn)，最坏情况下树呈现链状，为O(n)。