[binary tree post order traversal]

my notes:

Binary Trees 101
A binary tree is a hierarchical data structure where each node has:

- At most 2 children: Left 🌿 & Right 🌿.
- Uses: Search Trees, Expression Parsing, ML Models (Decision Trees, Random Forests).
  Let's dive into traversal techniques! 🧵👇

📜 **Tree Traversal**
Traversal = visiting all nodes in a tree. 3 styles:
1️⃣ **Inorder (LNR)**: Left -> Node -> Right
2️⃣ **Preorder (NLR)**: Node -> Left -> Right
3️⃣ **Postorder (LRN)**: Left -> Right -> Node

Postorder is often used for **evaluating expressions** or deleting a tree! 🚀

🔥 **Binary Tree Postorder Traversal Problem**
Goal: Visit all nodes in **postorder**.
Input: Root of the binary tree.
Output: List of nodes in L-R-N order.
Example Tree:

markdown
Copier le code

```
    1
   /  \
  2    3
 /  \
4    5
```

Postorder = [4, 5, 2, 3, 1] 🧩

🛠️ **Steps to Solve Postorder Traversal** (Recursive Way):
1️⃣ If tree is empty, return.

2️⃣ Recur on the left subtree 🌿.
3️⃣ Recur on the right subtree 🌿.
4️⃣ Visit the node 🌟.

Python Code:

python
Copier le code

```
def postorder_traversal(root):
    return (postorder_traversal(root.left) +
            postorder_traversal(root.right) +
            [root.val]) if root else []
```

🔍 **Iterative Solution** for Postorder Traversal:
Use a stack! Simulate the recursion process.
1️⃣ Push root & track nodes.
2️⃣ Pop and add right subtree first.
3️⃣ Reverse the result.

Python Code:

python
Copier le code

```
def postorder_iterative(root):
    stack, res = [root], []
    while stack:
        node = stack.pop()
        if node:
            res.append(node.val)
            stack.extend([node.left, node.right])
    return res[::-1]
```

❓ **Why is this problem important?**

- **FAANG Insight**: Tree traversal questions test recursion, iteration, and your grasp of data structures.
- It's a building block for more advanced algorithms (e.g., parsing syntax trees, graph traversal).

Start with basics, then master variations like zig-zag or level-order! 🚀

```
/**
```

```
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {


    }
};
```

********** solution**********

"""

Method 1: recursive

"""

```
def postorderTraversal(self, root):
    res = []
    def helper(root):
        if root:
            helper(root.left)
```

```
            helper(root.right)
            res.append(root.val)
    helper(root)
    return res

"""
Method 2: iterative, using two stacks
initialize first stack with root node, do the following if first is not None:
    pop from first stack, append the popped element to second stack,
    add left and right node to first stack if they are not None
return the second stack's value in reverse order
"""
def postorderTraversal(self, root):
    if not root: return []
    first, second = [root], []
    while first:
        node = first.pop()
        second.append(node)
        if node.left: first.append(node.left)
        if node.right: first.append(node.right)
    res = [node.val for node in second]
    res.reverse()
    return res
```

whole explanation:
https://techgeekyan.blogspot.com/2017/08/leetcode-145-binary-tree-postorder.html