Function Documentation: postorderTraversal

Description

The postorderTraversal function performs a postorder traversal on a binary tree. Postorder traversal visits nodes in the following order: left subtree, right subtree, and then the root node. This function returns a list of node values in the postorder sequence.

Parameters

- **root (Optional[TreeNode]):** The root node of the binary tree. The tree can be empty (None), or it can be a valid TreeNode object representing the root of the tree.

Returns

- **List[int]:** A list of integers representing the values of the nodes in the postorder traversal order.

Example 1

- **Input:** root = [1, null, 2, 3]
- **Output:** [3, 2, 1]

Example 2

- **Input:** root = []
- **Output:** []

Example 3

- **Input:** root = [1]
- **Output:** [1]

Constraints

- The number of nodes in the tree is in the range [0, 100].
- Node values are in the range [-100, 100].

Approach

1. **Initialization:**

- Create two stacks: stack1 to hold nodes for processing, and stack2 to store the postorder traversal sequence in reverse.

2. **Traversal:**

- Push the root node onto stack1.
- *While stack1 is not empty:*
  - Pop a node from stack1.
  - Append the node's value to stack2.
  - Push the left and right children of the node onto stack1.

3. **Reverse the Result:**

- Since the nodes are processed in reverse postorder in stack2, reverse stack2 to obtain the correct postorder traversal sequence.

Follow-Up

The function provided is an iterative solution to postorder traversal. The recursive approach is straightforward but may lead to stack overflow with deep trees. The iterative approach uses explicit stacks to manage traversal order, which is more suitable for environments with limited recursion depth.