Binary In-order Traversal

28 February 2025 11:18

Question:

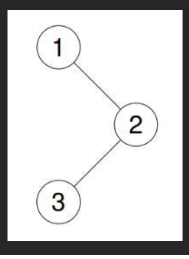
Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:

Input: `root = [1,null,2,3]`

Output: `[1,3,2]`

Explanation:



The question is asking you to perform an **inorder traversal** of a binary tree **iteratively** (without recursion).

<u>Input and Output</u>

Input:
- A **binary tree root node**.

Output:
- A **list** containing node values in **inorder traversal order**.

What You Need to Do
- Start from the root node.
- Use a **stack** to keep track of nodes while traversing.
- Follow **Left → Root → Right** order.

Input Tree:

markdown Copy Edit

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

Expected Output:

`[4, 2, 5, 1, 3]`

(Because **left subtree (4, 2, 5) → root (1) → right subtree (3)**)

Why Use a Stack?

Since recursion automatically handles function calls, we need a **stack** to manually keep track of nodes while we traverse.

The general approach is:

1. **Go left as far as possible**, pushing nodes onto the stack.
2. **Pop the stack**, process the node (store its value).
3. **Go to the right** subtree and repeat.

Code:

python

```python
def inorderTraversal(self, root):
    res, stack = [], []
    curr = root
    while curr or stack:
        while curr:
            stack.append(curr)
            curr = curr.left  # Go left as much as possible
        curr = stack.pop()
        res.append(curr.val)
        curr = curr.right  # Then go right
    return res
```

Explanation:

Initial State:
- stack = []
- res = []
- curr = 1 (root)

```python
while curr or stack:
```

- **Loop runs as long as there's a node (curr) or nodes in the stack**.
- Ensures we process all nodes **even when curr is None**.

```python
while curr:
    stack.append(curr)
    curr = curr.left
```

- **Push nodes onto the stack** while moving **left**.
- This mimics recursion by storing nodes that still need to be processed.
- Stops when there are no more left children.

```python
curr = stack.pop()  #
res.append(curr.val)
```

- Pop the **last inserted node** (this is the leftmost unprocessed node).
- Add its value to res because in **inorder traversal, we visit left first**.
- Now, we need to check its **right subtree**.

```python
curr = curr.right
```

- After processing the node, move to its **right child**.
- If the node **has no right child**, the next iteration will pop from the stack.
- If it **has a right child**, we will push its leftmost children onto the stack next.

```python
return res  # Return the final inorder traversal list
```

Example Walkthrough

Let's say we have this tree:

markdown Copy Edit

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

Step-by-Step Execution

Step	`curr`	Stack Before	Stack After	`res` (Result)
1	`1`	`[]`	`[1]`	`[]`
2	`2`	`[1]`	`[1, 2]`	`[]`
3	`4`	`[1, 2]`	`[1, 2, 4]`	`[]`
4	Pop `4`	`[1, 2, 4]`	`[1, 2]`	`[4]`
5	Pop `2`	`[1, 2]`	`[1]`	`[4, 2]`
6	`5`	`[1]`	`[1, 5]`	`[4, 2]`
7	Pop `5`	`[1, 5]`	`[1]`	`[4, 2, 5]`
8	Pop `1`	`[1]`	`[]`	`[4, 2, 5, 1]`

6	`5`	`[1]`	`[1, 5]`	`[4, 2]`
7	Pop `5`	`[1, 5]`	`[1]`	`[4, 2, 5]`
8	Pop `1`	`[1]`	`[]`	`[4, 2, 5, 1]`
9	`3`	`[]`	`[3]`	`[4, 2, 5, 1]`
10	Pop `3`	`[3]`	`[]`	`[4, 2, 5, 1, 3]`

```
return res  # Return the final inorder traversal list
```