Previous Next

145. Binary Tree Postorder Traversal

Oct. 12, 2018 | 45.4K views

★★★★★
Average Rating: 2.78 (41 votes)

Given a binary tree, return the *postorder* traversal of its nodes' values.

Example:

```
Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [3,2,1]
```

Follow up: Recursive solution is trivial, could you do it iteratively?

Solution

How to traverse the tree

There are two general strategies to traverse a tree:

- *Breadth First Search* (`BFS`)

  We scan through the tree level by level, following the order of height, from top to bottom. The nodes on a higher level would be visited before the ones on the lower levels.

- *Depth First Search* (`DFS`)

  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach down to a leaf, and then back to root to reach another branch.

  The DFS strategy can further be distinguished as `preorder`, `inorder`, and `postorder` depending on the relative order among the root node, left node, and right node.

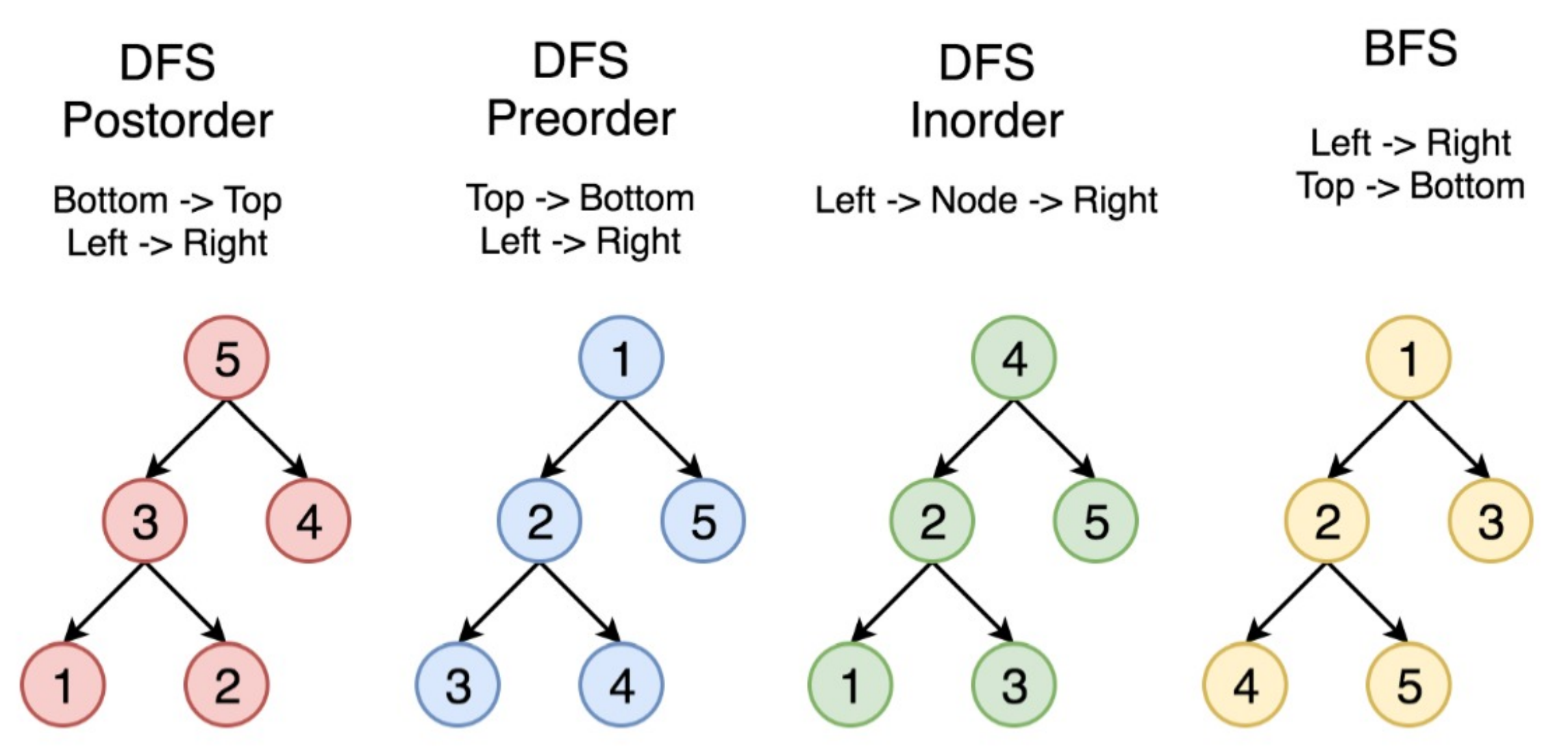


Figure 1. The nodes are numerated in the order you visit them, please follow `1-2-3-4-5` to compare different strategies.

Here the problem is to implement postorder traversal using iterations.

Approach 1: Recursive Postorder Traversal

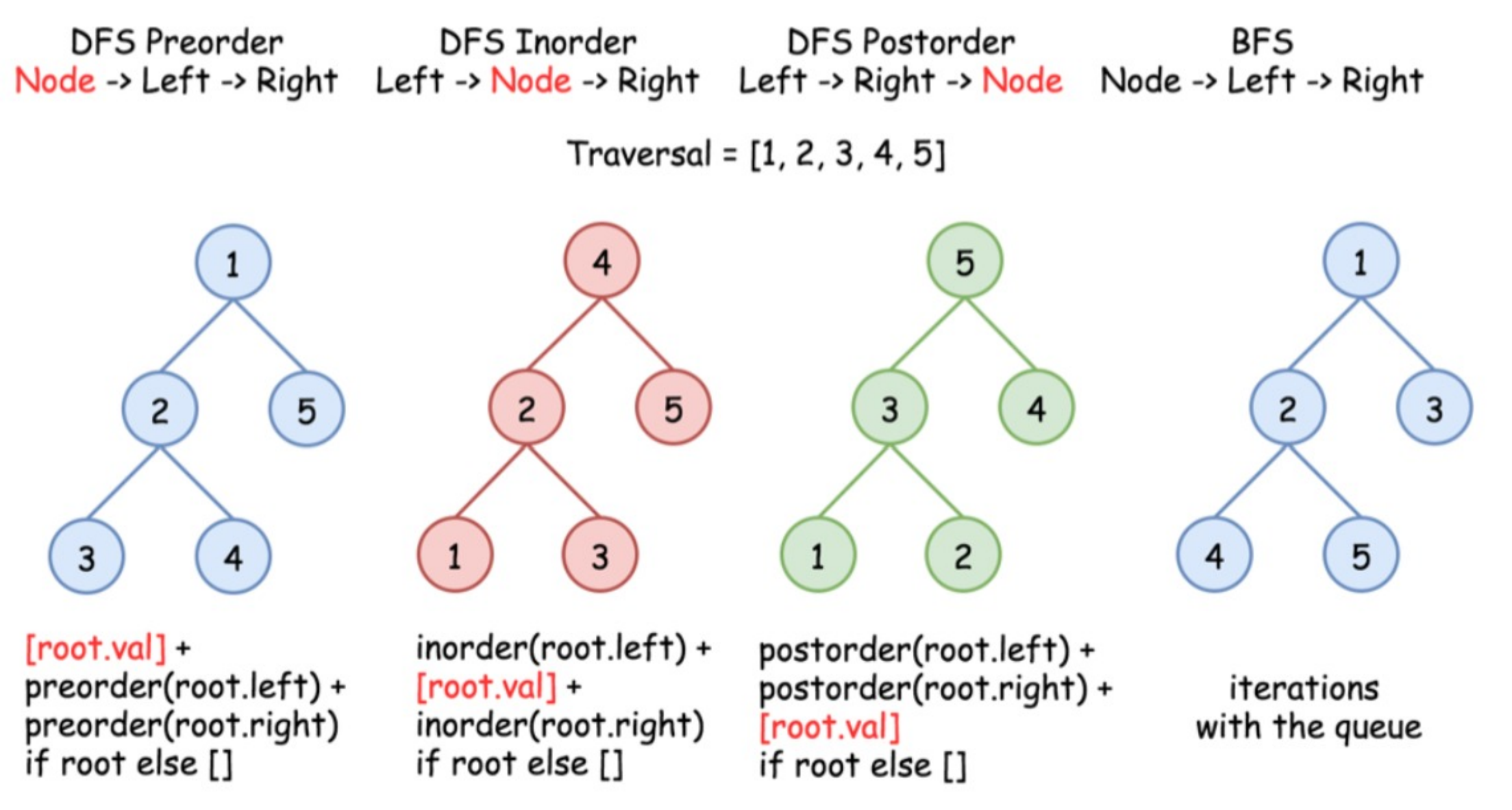


Figure 2. Recursive DFS traversals.

The most straightforward way is to implement recursion using Left -> Right -> Node traversal strategy.

Java Python3 Copy

```python
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if not root:
            return []
        return self.postorderTraversal(root.left) + self.postorderTraversal(root.right) + [root.val]
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$, where N is the number of nodes. We visit each node exactly once, thus the time complexity is $\mathcal{O}(N)$.
- Space complexity: up to $\mathcal{O}(H)$ to keep the recursion stack, where H is a tree height.

Approach 2: Iterative Preorder Traversal: Tweak the Order of the Output

Let's start from the root, and at each iteration, pop the current node out of the stack and push its child nodes. In the implemented strategy, we push nodes into stack following the order Top->Bottom and Left->Right. Since DFS postorder transversal is Bottom->Top and Left->Right the output list should be reverted after the end of the loop.

Java Python3 Copy

```python
class Solution(object):
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return []

        stack, output = [root, ], []
        while stack:
            root = stack.pop()
            output.append(root.val)
            if root.left is not None:
                stack.append(root.left)
            if root.right is not None:
                stack.append(root.right)

        return output[::-1]
```

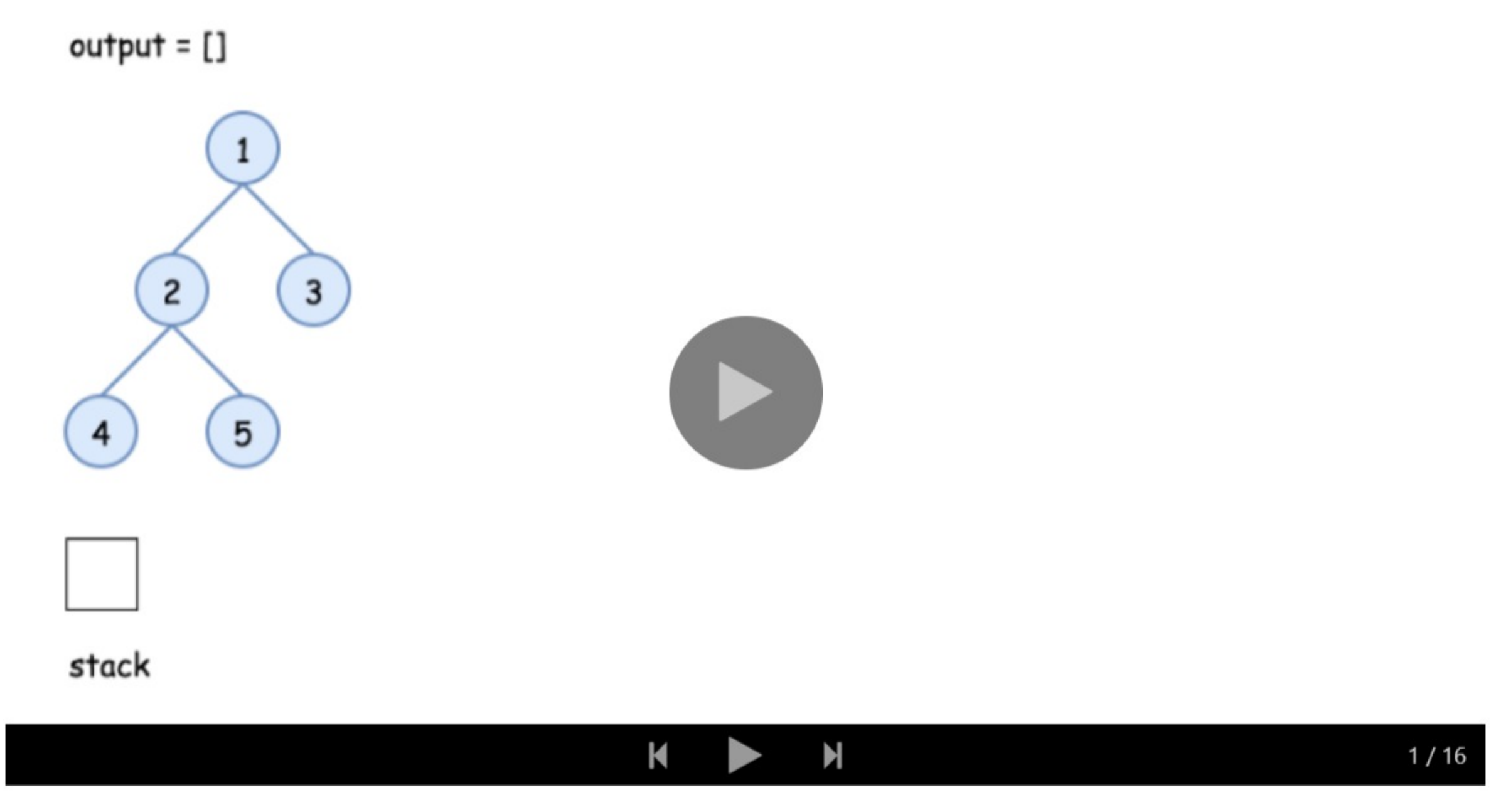
Complexity Analysis

- Time complexity: $\mathcal{O}(N)$, where N is the number of nodes. We visit each node exactly once, thus the time complexity is $\mathcal{O}(N)$.
- Space complexity: up to $\mathcal{O}(H)$ to keep the stack, where H is a tree height.

Approach 3: Iterative Postorder Traversal

Algorithm

The idea is to fulfill the stack following right->node->left strategy. One could pop the last node out of the stack and check if it's the leftmost leaf. If yes, it's time to update the output. Otherwise, one should swap the last two nodes in the stack and repeat all these steps.



Implementation

Java Python3 Copy

```python
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        stack, output = [], []
        while root or stack:
            # push nodes: right -> node -> left
            while root:
                if root.right:
                    stack.append(root.right)
                stack.append(root)
                root = root.left

            root = stack.pop()

            # if the right subtree is not yet processed
            if stack and root.right == stack[-1]:
                stack[-1] = root
                root = root.right
            # if we're on the leftmost leaf
            else:
                output.append(root.val)
                root = None

        return output
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$, where N is the number of nodes. We visit each node exactly once, thus the time complexity is $\mathcal{O}(N)$.
- Space complexity: up to $\mathcal{O}(H)$ to keep the stack, where H is a tree height.

Rate this article: ★★★★★

Previous Next

Comments: 17 Sort By

Type comment here... (Markdown is supported)

Preview Post

jianchao-li ★ 14342 March 9, 2019 3:16 PM

Actually this solution does not visit the nodes in the post order but just tweaks the order of the output.

76 Share Reply

SHOW 7 REPLIES

lenchen1112 ★ 976 January 6, 2020 2:48 PM

Another way to do postorder traversal.

```
class Solution:
    def postorderTraversal(self, root: TreeNode) -> List[int]:
        stack, result = [], []
```

Read More

6 Share Reply

liuyubobobo ★ 324 October 13, 2018 2:04 PM

There are so many ways to complete this classic problem. I offered nine solutions on my Leetcode repo in both C++ and Java. :-)

8 Share Reply

SHOW 2 REPLIES

moyiyiyiyi ★ 7 June 5, 2019 6:08 AM

? ? Why is this listed as hard lol

6 Share Reply

SHOW 1 REPLY

milinthosani ★ 31 January 17, 2020 12:16 PM

Just discovered another method. What do you guys think about this?

```
class Solution(object):
    def postorderTraversal(self, root):
        if not root:
```

Read More

2 Share Reply

SHOW 1 REPLY

ngoc_lam ★ 42 October 13, 2018 8:26 AM

@liaison you should add moris traversal for O(1) space :)

2 Share Reply

SHOW 2 REPLIES

Bamba19 ★ 1 July 1, 2020 1:38 AM

Please use explicit addLast, getLast, addFirst, getFirst of the LinkedList API, or just use Stack push/pop API, otherwise really hard to understand that it is indeed standard PreOrderTraversal with inverted return value

1 Share Reply

frozenleetcode ★ 8 January 19, 2019 2:43 PM

I wonder if `addFirst()` all the time will lead to a time complexity of `O(N^2)`, cause it will push all the elements back to make space for the new added element

1 Share Reply

SHOW 1 REPLY

kk03 ★ 5 July 4, 2020 2:48 AM

I still don't get the idea why [::-1] ,
would do the work.Anyone can explain please?

0 Share Reply

slemur ★ 3 June 23, 2020 1:56 AM

This python solution should/would not be accepted in an interview. The instructions are to do a post-order traversal, which is more tricky to do with a stack than the pre-order traversal because nodes cannot be immediately visited. An interviewer would probably say output[::-1] is a hack.

0 Share Reply

‹ 1 2 ›