# 144. Binary Tree Preorder Traversal (/problems/binary-tree-preorder-traversal/)

Oct. 21, 2018 | 17.7K views

Average Rating: 5 (15 votes)

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

**Example:**

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

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

**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 higher level would be visited before the ones with lower levels.

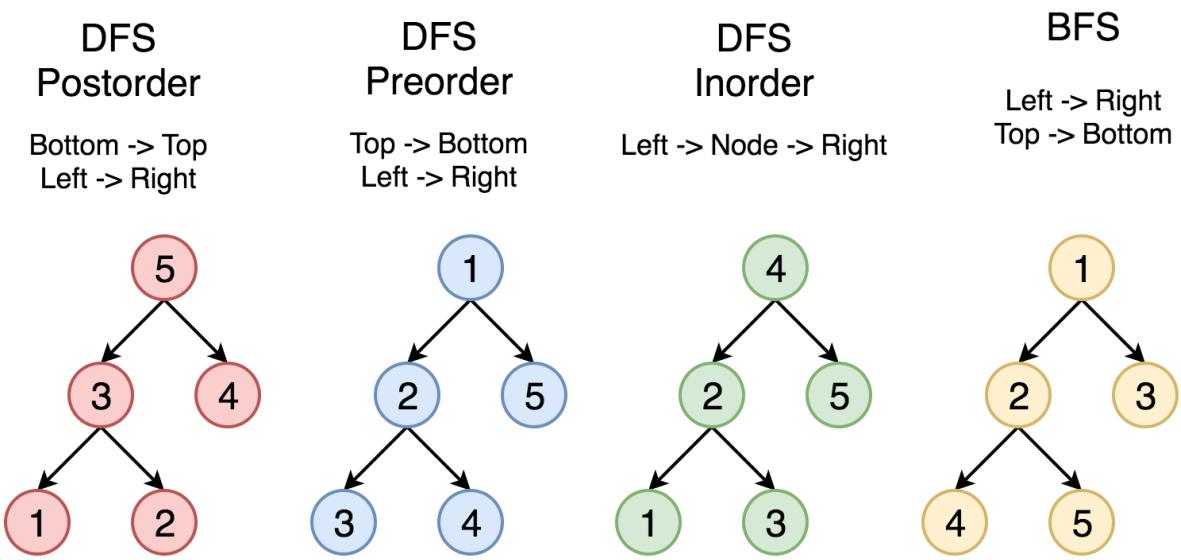
- *Depth First Search* ( `DFS` )

  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and

reach all the way down to certain 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.

On the following figure 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: Iterations

### Algorithm

First of all, here is the definition of the `TreeNode` which we would use in the following implementation.

Java | Python — Copy

```
class TreeNode(object):
    """ Definition of a binary tree node."""
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

Let's start from the root and then at each iteration pop the current node out of the stack and push its child nodes. In the implemented strategy we push nodes into output list following the order

`Top->Bottom` and `Left->Right`, that naturally reproduces preorder traversal.



Java | Python — Copy

```python
class Solution(object):
    def preorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        if root is None:
            return []

        stack, output = [root, ], []

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

        return output
```

**Complexity Analysis**

- Time complexity : we visit each node exactly once, thus the time complexity is $\mathcal{O}(N)$, where $N$ is the number of nodes, *i.e.* the size of tree.
- Space complexity : depending on the tree structure, we could keep up to the entire tree, therefore, the space complexity is $\mathcal{O}(N)$.

## Approach 2: Morris traversal

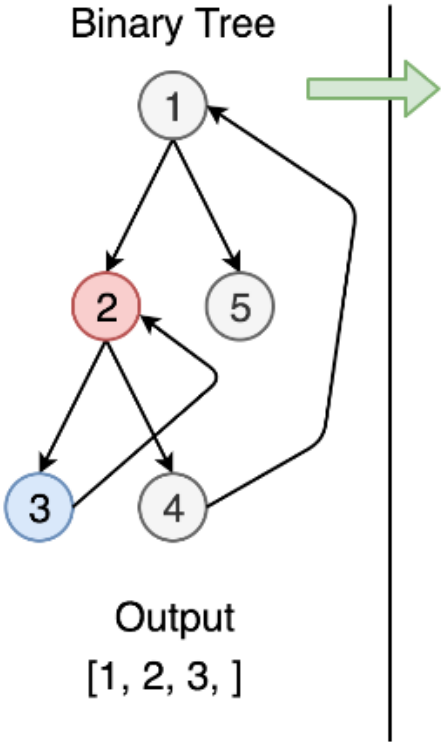
This approach is based on Morris's article (https://www.sciencedirect.com/science/article/pii/0020019079900681) which is intended to optimize the space complexity. The algorithm does not use additional space for the computation, and the memory is only used to keep the output. If one prints the output directly along the computation, the space complexity would be $\mathcal{O}(1)$.

**Algorithm**

Here the idea is to go down from the node to its predecessor, and each predecessor will be visited twice. For this go one step left if possible and then always right till the end. When we visit a leaf (node's predecessor) first time, it has a zero right child, so we update output and establish the pseudo link `predecessor.right = root` to mark the fact the predecessor is visited. When we visit the same predecessor the second time, it already points to the current node, thus we remove pseudo link and move right to the next node.

If the first one step left is impossible, update output and move right to next node.






node = 2

Is left child of the node is None? - No. -Then go left.

node = 2, predecessor = 3

Go right till you meet None or current node (node = 2)

node = 2, predecessor = 3, predecessor.right = None

You meet None
1. Add node to output
2. Link predecessor.right = node (3.right = 2)
3. New node is node.left

node = 3

Is left child of the node is None? - Yes!
1. Add node to output
2. New node is node.right

node = 2

Is left child of the node is None? - No. -Then go left.

node = 2, predecessor = 3

9 / 1

Java | Python

Articles > 144. Binary Tree Preorder Traversal

Copy

```python
class Solution(object):
    def preorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        node, output = root, []
        while node:
            if not node.left:
                output.append(node.val)
                node = node.right
            else:
                predecessor = node.left

                while predecessor.right and predecessor.right is not node:
                    predecessor = predecessor.right

                if not predecessor.right:
                    output.append(node.val)
                    predecessor.right = node
                    node = node.left
                else:
                    predecessor.right = None
                    node = node.right

        return output
```

**Complexity Analysis**

- Time complexity : we visit each predecessor exactly twice descending down from the node, thus the time complexity is $\mathcal{O}(N)$, where $N$ is the number of nodes, *i.e.* the size of tree.
- Space complexity : we use no additional memory for the computation itself, but output list contains $N$ elements, and thus space complexity is $\mathcal{O}(N)$.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

## Rate this article:

Previous (/articles/minimize-malware-spread-ii/)

Next (/articles/n-ary-tree-postorder-transversal/)

## Comments: 10

Sort By

Type comment here... (Markdown is supported)

Preview

Post

(/dayupt)

DAYUPT (dayupt) ★ 15 December 24, 2018 7:00 PM

Below the tree graph that you illustrated what is preorder、 inorder、 postorder and bfs, you seemed to write the preorder to postorder for a slip of the pen.

2 Share Reply

deleted_user ★ 31 February 2, 2019 3:32 PM

As with all tree problems like this... if this input is small enough for the call stack, the recursive form is always fastest:

```
class Solution {
    final List<Integer> l = new ArrayList<>();
```

Read More

1 Share Reply

(/minimelissa)

MiniMelissa (minimelissa) ★ 0 June 16, 2019 4:14 PM

Is there Anyone can explain to me why the space complexity is O(N) instead of O(lgN) in the iteration solution? Even though we visite each node, the largest length of the stack we use is logN.

0 Share Reply

SHOW 1 REPLY

(/jamescool)

JamesCool (jamescool) ★ 28 June 8, 2019 11:31 PM

Nice, but I barely get the meaning of "if root is not None:" from Approach 1. Since each node pushed into the stack "is not None", why you unnecessarily re-check if it is None?

0 Share Reply

SHOW 1 REPLY

(/shermm)

SherMM (shermm) ★ 31 March 5, 2019 7:49 AM

This solution worked for me, and isn't quite as complicated as the posted version.

```
def preorderTraversal(self, root: TreeNode) -> List[int]:
        if not root:
            return []
```

Read More

0 Share Reply

(/laiyinlg)

laiyinlg (laiyinlg) ★ 55 January 16, 2019 10:43 AM

Java:

```
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        return helper(root);
```

Read More

0 Share Reply

(/nobler)

nobler (nobler) ★ 7 January 11, 2019 3:28 AM

In approach 1, stack.pollLast() may be O(log n). Total time complexity should be O(n * log n).

0 Share Reply

SHOW 1 REPLY

(/alusov)

alusov (alusov) ★ 9 January 6, 2019 12:25 AM

Hi! Time complexity isn't correct for Morris traversal approach. It's O(N*log(N)) because for each current node you have to create pseudo link. See explanation here https://codeoverflow.wordpress.com/tag/morris-inorder-traversal/ (https://codeoverflow.wordpress.com/tag/morris-inorder-traversal/)

0 Share Reply

(/nujia)

nujia (nujia) ★ 1 December 9, 2018 1:14 AM

Isn't anyone find out that, for a preorder traversal, the predecessor in approach 2 is wrong against the definition? I look up the definition in this link https://www.geeksforgeeks.org/preorder-predecessor-node-binary-tree/ (https://www.geeksforgeeks.org/preorder-predecessor-node-binary-tree/)

0 Share Reply

SHOW 1 REPLY

(/unegor)

unegor (unegor) ★ -2 November 14, 2018 11:02 PM

Please fix copy paste `(predecessor.right != null) && (predecessor.right != node)`

-2 Share Reply

SHOW 1 REPLY