590. N-ary Tree Postorder Traversal

Oct. 21, 2018 | 15.4K views

Average Rating: 4.50 (6 votes)

Given an n-ary tree, return the *postorder* traversal of its nodes' values.

Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

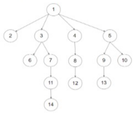
Follow up:

Recursive solution is trivial, could you do it iteratively?

Example 1:

```
Input: root = [1,null,3,2,4,null,5,6]
Output: [5,6,3,2,4,1]
```

Example 2:



```
Input: root = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null
Output: [2,6,14,11,7,3,12,8,4,13,9,10,5,1]
```

Constraints:

- The height of the n-ary tree is less than or equal to `1000`
- The total number of nodes is between `[0, 10^4]`

Solution

How to traverse the tree

First of all, please refer to this article for the solution in case of binary tree. This article offers the same ideas with a bit of generalisation.

There are two general strategies to traverse a tree:

- *Breadth First Search* (`BFS`)

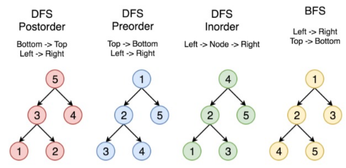
  We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

- *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

```python
# Definition for a Node.
class Node(object):
    def __init__(self, val, children):
        self.val = val
        self.children = children
```

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 stack following the order `Top->Bottom` and `Left->Right`. Since DFS postorder traversal is `Bottom->Top` and `Left->Right` the output list should be reverted after the end of loop.

Java | Python — Copy

```python
class Solution(object):
    def postorder(self, root):
        """
        :type root: Node
        :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)
                for c in root.children:
                    stack.append(c)

        return output[::-1]
```

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 $\mathcal{O}(N)$.

Rate this article:

Previous Next

Comments: 8

Sort By

Type comment here... (Markdown is supported)

Preview Post

deleted_user ★ 200 February 1, 2019 3:27 AM

A recursive version:

```
class Solution {
    public List<Integer> postorder(Node root) {
        return (root == null) ? new LinkedList<>() :
```

Read More

4 Share Reply

donelleyr ★ 0 April 29, 2019 12:07 AM

C# Iterative version. Time complexity: O(N) Space complexity: O(N).

```
public IList<int> Postorder(Node root) {
    IList<int> result = new List<int>();
    if (root == null) return result;
```

Read More

2 Share Reply

woshi471983103 ★ 1 January 22, 2019 4:25 AM

thx bro nice work

1 Share Reply

hackerjatt ★ 2 March 11, 2019 10:42 PM

def postorder(self, root: 'Node') -> List[int]:
if not root:
return self.res
for child in root.children:
self.postorder(child)

Read More

0 Share Reply

danielehabor ★ 0 February 22, 2019 2:08 PM

Are you sure this solution works for
1
/|
2 5 4
/ |

Read More

0 Share Reply

SHOW 1 REPLY

Olament ★ 1 July 6, 2020 2:57 AM

Iterative solution

```
def postorder(self, root: 'Node') -> List[int]:
    if not root:
        return []
```

Read More

0 Share Reply

neilteng ★ 20 June 8, 2020 1:38 AM

Facing the same problem in Binary Tree Postorder Traversal's solution, this solution does not actually traverse the tree in postorder but just tweak the output..

```
// reference: https://www.***.org/iterative-postorder-traversal-of-n-ary-tree/#:
```

Read More

0 Share Reply

gerait_ ★ 28 March 28, 2020 12:04 AM

Can anyone suggest c++ code with the same complexity as this solution?

0 Share Reply

nsahadat ★ 11 February 27, 2020 8:22 AM

class Solution(object):
def postorder(self, root):
"""
:type root: Node

Read More

0 Share Reply