589. N-ary Tree Preorder Traversal

Oct. 28, 2018 | 18.2K views

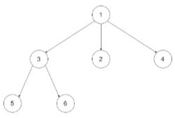
Given an n-ary tree, return the *preorder* traversal of its nodes' values.

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:

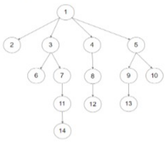
Recursive solution is trivial, could you do it iteratively?

Example 1:




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

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: [1,2,3,6,7,11,14,4,8,12,5,9,13,10]
```

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

Approach 1: Iterations

Algorithm

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.

First of all, here is the definition of the tree `Node` 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 output list following the order `Top->Bottom` and `Left->Right`, that naturally reproduces preorder traversal.

Java | Python | Copy

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

        stack, output = [root, ], []
        while stack:
            root = stack.pop()
            output.append(root.val)
            stack.extend(root.children[::-1])

        return output
```

Complexity Analysis

- Time complexity : we visit each node exactly once, and for each visit, the complexity of the operation (*i.e.* appending the child nodes) is proportional to the number of child nodes `n` (n-ary tree). Therefore the overall 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

jakubsuszynski 4 December 9, 2018 8:27 PM

```
class Solution {
    public List<Integer> preorder(Node root) {
        List<Integer> list = new ArrayList<>();
        if(root != null) {
```

Read More

4 Share Reply

SHOW 1 REPLY

rohan999 8 September 11, 2019 2:29 AM

Just pointing out that this is exactly a **DFS traversal**. And if you know DFS for graphs, try to apply the same logic here for the iterative answer.

3 Share Reply

WilmerKrisp 21 June 13, 2020 8:28 PM

The page (mentioned in the article) does not open https://leetcode.com/articles/binary-tree-preorder-transversal/

2 Share Reply

srihank 174 May 13, 2020 8:28 AM

Recursive and Iterative Solution.

Recursive:

```
    public List<Integer> preorder(Node root) {
```

Read More

1 Share Reply

deleted_user 200 February 1, 2019 4:24 AM

If our input tree is small enough, the recursive form is faster / won't cause a stack overflow:

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

Read More

1 Share Reply

sheva29 10 January 15, 2019 2:59 AM

You can use a Stack and avoid the Collections.reverse() to save some time:

```
class Solution {
    public List<Integer> preorder(Node root) {
        Stack<Node> s = new Stack();
```

Read More

1 Share Reply

SHOW 2 REPLIES

clcaste 0 June 23, 2020 8:58 PM

A very basic question, how do I use the Node class to create the tree?
I want to test the above solution but can't construct the tree to pass as argument

0 Share Reply

Prashanth_123 1 March 26, 2020 9:44 AM

```
class Solution {
    List<Integer> output;
    public List<Integer> preorder(Node root) {
        output = new LinkedList<>();
```

Read More

0 Share Reply