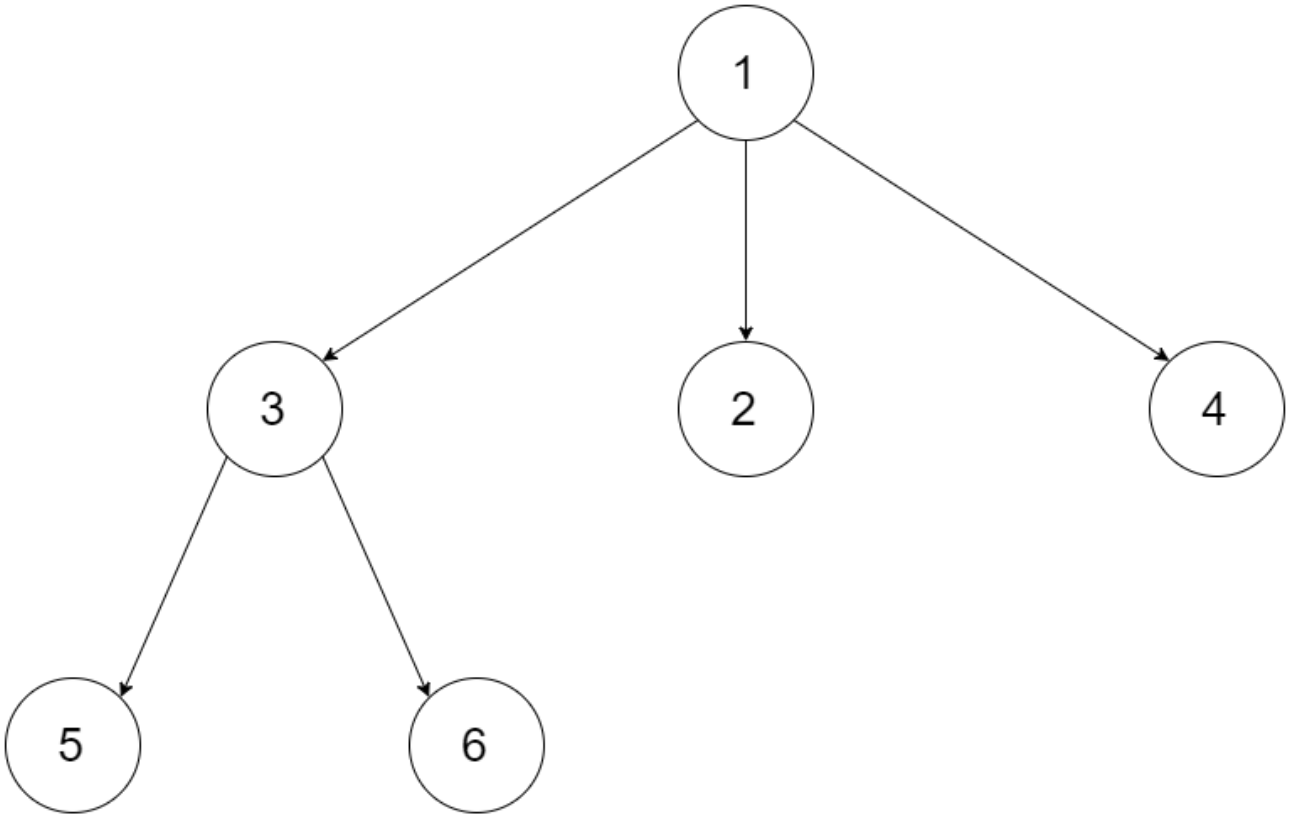
589. N-ary Tree Preorder Traversal (/problems/n-ary-tree-preorder-traversal/)

Oct. 28, 2018 | 9.6K views

Average Rating: 5 (9 votes)

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

For example, given a `3-ary` tree:



Return its preorder traversal as: `[1,3,5,6,2,4]`.

Note:

Recursive solution is trivial, could you do it iteratively?

Solution

Approach 1: Iterations

Algorithm

First of all, please refer to this article (https://leetcode.com/articles/binary-tree-preorder-transversal/) 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

```
# 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 Articles > 589. N-ary Tree Preorder Traversal 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 $\mathcal{O}(N)$.

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

Rate this article:

Previous (/articles/minimum-depth-of-binary-tree/) Next (/articles/unique-binary-search-trees-ii/)

Comments: 4

Sort By

Type comment here... (Markdown is supported)

Preview Post

jakubsuszynski (jakubsuszynski) ★3 December 9, 2018 6:57 AM

(/jakubsuszynski)


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

Read More

3 Share Reply

SHOW 1 REPLY

deleted_user ★31 January 31, 2019 2:54 PM

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

0 Share Reply

sheva29 (sheva29) ★9 January 14, 2019 1:29 PM

(/sheva29)

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

0 Share Reply

SHOW 1 REPLY

rohan999 (rohan999) ★3 September 10, 2019 1:59 PM

(/rohan999)

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.

0 Share Reply