Previous Next

129. Sum Root to Leaf Numbers

March 14, 2020 | 9.7K views

Average Rating: 4.50 (16 votes)

Given a binary tree containing digits from `0-9` only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123`.

Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

Example:

```
Input: [1,2,3]
    1
   / \
  2   3
Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.
```

Example 2:

```
Input: [4,9,0,5,1]
    4
   / \
  9   0
 / \
5   1
Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

Solution

Overview

Prerequisites

There are three DFS ways to traverse the tree: preorder, postorder and inorder. Please check two minutes picture explanation, if you don't remember them quite well: here is Python version and here is Java version.

Optimal Strategy to Solve the Problem

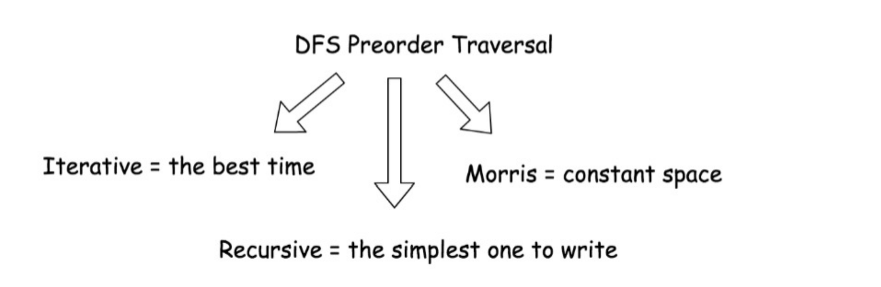
> Root-to-left traversal is so-called *DFS preorder traversal*. To implement it, one has to follow straightforward strategy Root->Left->Right.

Since one has to visit all nodes, the best possible time complexity here is linear. Hence all interest here is to improve the space complexity.

> There are 3 ways to implement preorder traversal: iterative, recursive and Morris.

Iterative and recursive approaches here do the job in one pass, but they both need up to $\mathcal{O}(H)$ space to keep the stack, where H is a tree height.

Morris approach is two-pass approach, but it's a constant-space one.

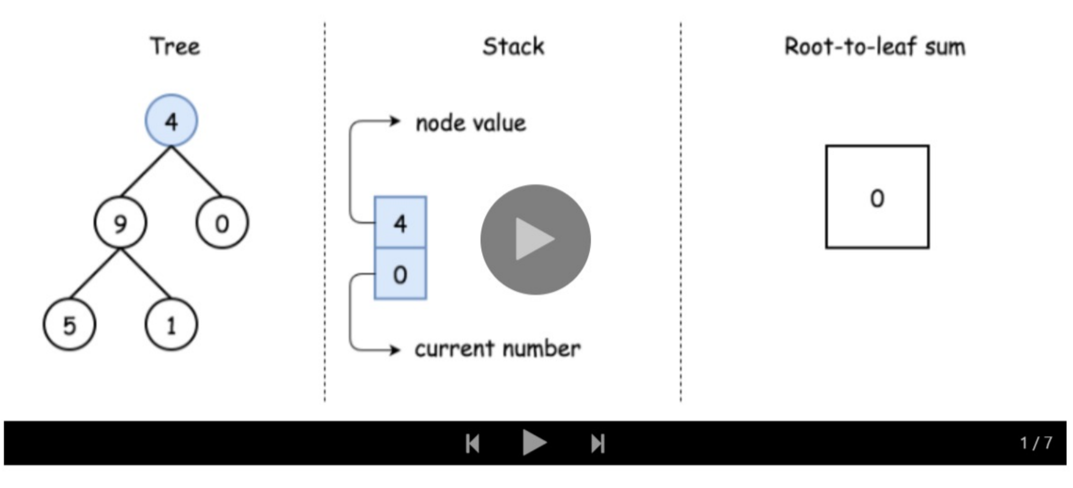


Approach 1: Iterative Preorder Traversal.

Intuition

Here we implement standard iterative preorder traversal with the stack:

- Push root into stack.
- While stack is not empty:
  - Pop out a node from stack and update the current number.
  - If the node is a leaf, update root-to-leaf sum.
  - Push right and left child nodes into stack.
- Return root-to-leaf sum.



Implementation

Note, that Javadocs recommends to use ArrayDeque, and not Stack as a stack implementation.

Java | Python | Copy

```python
class Solution:
    def sumNumbers(self, root: TreeNode):
        root_to_leaf = 0
        stack = [(root, 0) ]

        while stack:
            root, curr_number = stack.pop()
            if root is not None:
                curr_number = curr_number * 10 + root.val
                # if it's a leaf, update root-to-leaf sum
                if root.left is None and root.right is None:
                    root_to_leaf += curr_number
                else:
                    stack.append((root.right, curr_number))
                    stack.append((root.left, curr_number))

        return root_to_leaf
```

Complexity Analysis

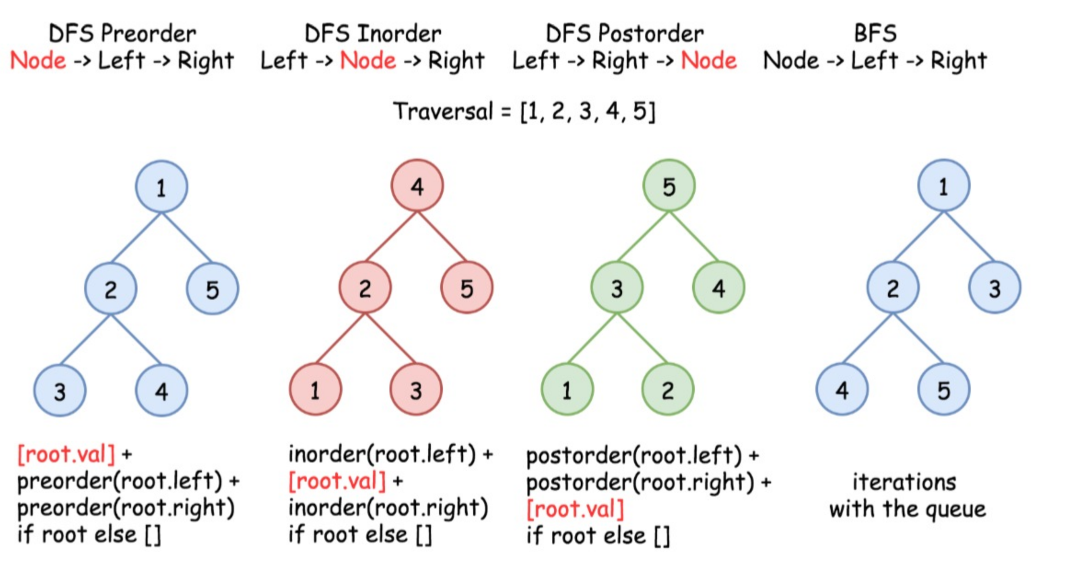
- Time complexity: $\mathcal{O}(N)$ since one has to visit each node.
- Space complexity: up to $\mathcal{O}(H)$ to keep the stack, where H is a tree height.

Approach 2: Recursive Preorder Traversal.

Iterative approach 1 could be converted into recursive one.

Recursive preorder traversal is extremely simple: follow Root->Left->Right direction, i.e. do all the business with the node (= update the current number and root-to-leaf sum), and then do the recursive calls for the left and right child nodes.

P.S. Here is the difference between *preorder* and the other DFS recursive traversals. On the following figure the nodes are numerated in the order you visit them, please follow `1-2-3-4-5` to compare different DFS strategies implemented as recursion.



Implementation

Java | Python | Copy

```python
class Solution:
    def sumNumbers(self, root: TreeNode):
        def preorder(r, curr_number):
            nonlocal root_to_leaf
            if r:
                curr_number = curr_number * 10 + r.val
                # if it's a leaf, update root-to-leaf sum
                if not (r.left or r.right):
                    root_to_leaf += curr_number

                preorder(r.left, curr_number)
                preorder(r.right, curr_number)

        root_to_leaf = 0
        preorder(root, 0)
        return root_to_leaf
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$ since one has to visit each node.
- Space complexity: up to $\mathcal{O}(H)$ to keep the recursion stack, where H is a tree height.

Approach 3: Morris Preorder Traversal.

We discussed already iterative and recursive preorder traversals, which both have great time complexity though use up to $\mathcal{O}(H)$ to keep the stack. We could trade in performance to save space.

The idea of Morris preorder traversal is simple: to use no space but to traverse the tree.

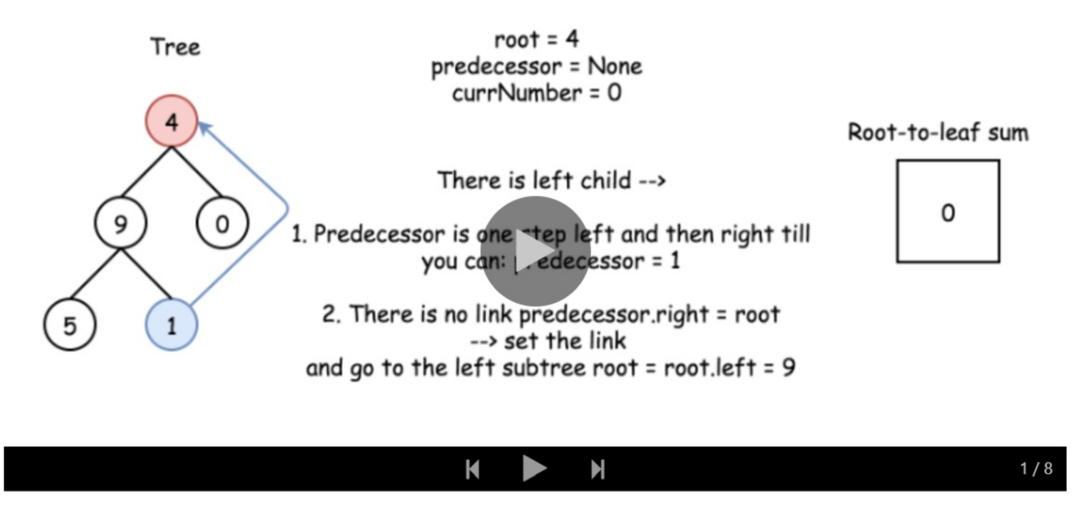
> How that could be even possible? At each node one has to decide where to go: to the left or tj the right, traverse the left subtree or traverse the right subtree. How one could know that the left subtree is already done if no additional memory is allowed?

The idea of Morris algorithm is to set the *temporary link* between the node and its predecessor: `predecessor.right = root`. So one starts from the node, computes its predecessor and verifies if the link is present.

- There is no link? Set it and go to the left subtree.
- There is a link? Break it and go to the right subtree.

There is one small issue to deal with : what if there is no left child, i.e. there is no left subtree? Then go straightforward to the right subtree.

Implementation



Java | Python | Copy

```python
class Solution:
    def sumNumbers(self, root: TreeNode):
        root_to_leaf = curr_number = 0

        while root:
            # If there is a left child,
            # then compute the predecessor.
            # If there is no link predecessor.right = root --> set it.
            # If there is a link predecessor.right = root --> break it.
            if root.left:
                # Predecessor node is one step to the left
                # and then to the right till you can.
                predecessor = root.left
                steps = 1
                while predecessor.right and predecessor.right is not root:
                    predecessor = predecessor.right
                    steps += 1

                # Set link predecessor.right = root
                # and go to explore the left subtree
                if predecessor.right is None:
                    curr_number = curr_number * 10 + root.val
                    predecessor.right = root
                    root = root.left
                # Break the link predecessor.right = root
                # Once the link is broken,
                # it's time to change subtree and go to the right
```

Complexity Analysis

- Time complexity: $\mathcal{O}(N)$.
- Space complexity: $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 8 Sort By

Type comment here... (Markdown is supported)

Preview Post

sainvamshid 6 April 4, 2020 7:12 AM

Morris Algo is too Good

4 Share Reply

SHOW 1 REPLY

trd 2 June 27, 2020 7:22 AM

> Morris approach is two-pass approach, but it's a constant-space one.

If I understand this correctly, this `two-pass` means we go through the `right` branch twice:

Read More

2 Share Reply

dtkmn 2 June 26, 2020 7:33 AM

do we suppose to know 'Morris' in the interview too?

2 Share Reply

SHOW 4 REPLIES

RedComet 1 June 29, 2020 4:22 PM

Morris Algorithm is a great approach. But don't you think it is too hard to code without any bug at the actual interview? Should I code this algorithm? Or is it ok to just mention about it?

1 Share Reply

SHOW 1 REPLY

monester 74 March 17, 2020 10:43 AM

wow Morris algorithm is truly amazing!

1 Share Reply

SHOW 1 REPLY

user0414A 19 June 27, 2020 1:50 PM

Backtracking solution:

```
# Definition for a binary tree node.
# class TreeNode:
```

Read More

0 Share Reply

NaughtyMonkey 0 June 27, 2020 5:12 AM

Does anyone else think the finding predecessor operation in the last solution takes logN time? Then, the overall time complexity should be O(NlogN).

0 Share Reply

SHOW 1 REPLY

ztztzt8888 53 June 26, 2020 1:34 PM

If you want to avoid using the instance variable for recursive dfs:

```
    public int sumNumbers(TreeNode root) {
        int[] sum = new int[1];
```

Read More

0 Share Reply

SHOW 1 REPLY