Previous Next

94. Binary Tree Inorder Traversal

Aug. 30, 2017 | 277.5K views

Average Rating: 4.39 (107 votes)

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

Example:

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

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

Follow up: Recursive solution is trivial, could you do it iteratively?

Solution

Approach 1: Recursive Approach

The first method to solve this problem is using recursion. This is the classical method and is straightforward. We can define a helper function to implement recursion.

Java

```
1  class Solution {
2      public List < Integer > inorderTraversal(TreeNode root) {
3          List < Integer > res = new ArrayList < > ();
4          helper(root, res);
5          return res;
6      }
7
8      public void helper(TreeNode root, List < Integer > res) {
9          if (root != null) {
10             if (root.left != null) {
11                 helper(root.left, res);
12             }
13             res.add(root.val);
14             if (root.right != null) {
15                 helper(root.right, res);
16             }
17         }
18     }
19 }
```

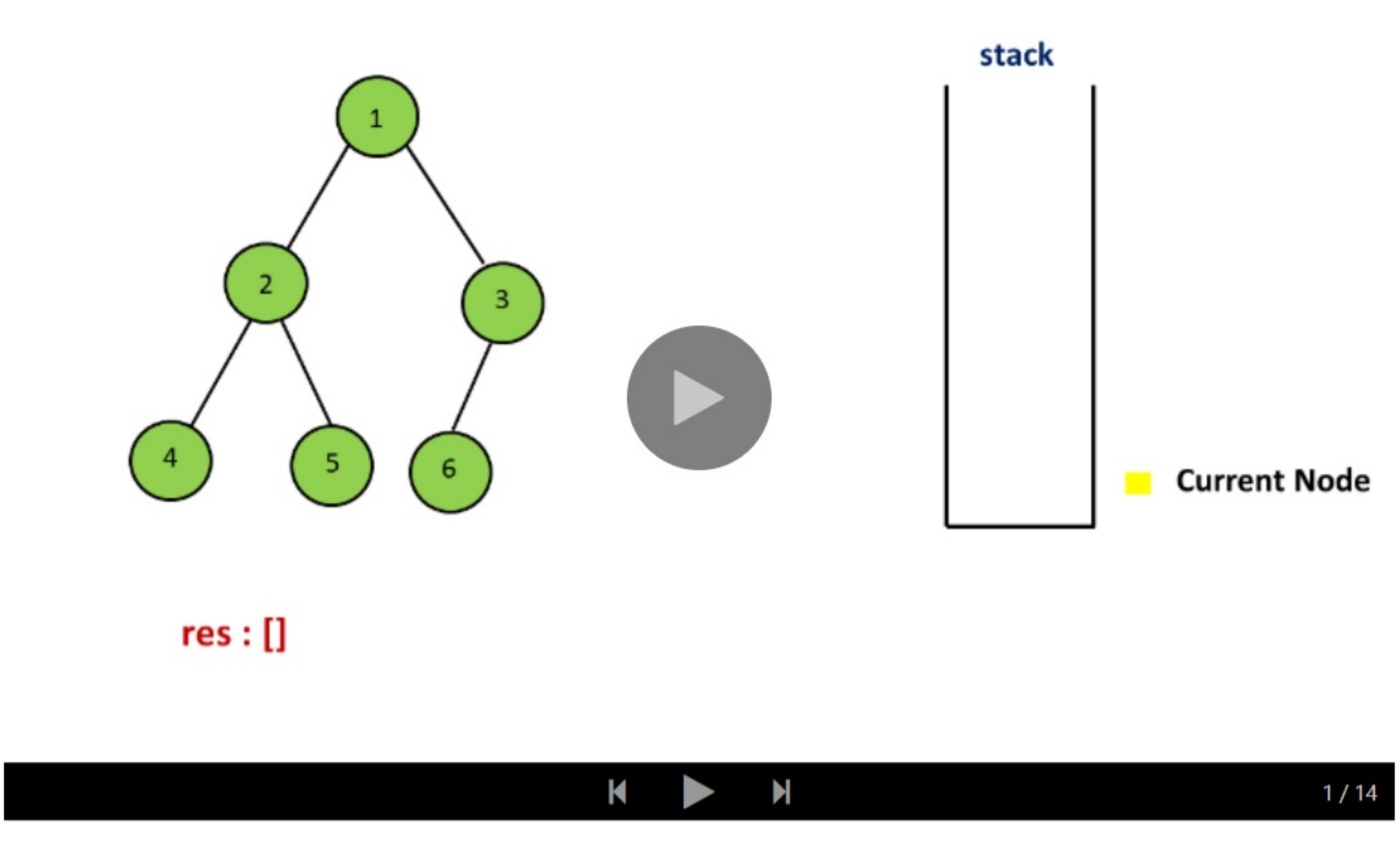
Complexity Analysis

- Time complexity : $O(n)$. The time complexity is $O(n)$ because the recursive function is $T(n) = 2 \cdot T(n/2) + 1$.
- Space complexity : The worst case space required is $O(n)$, and in the average case it's $O(\log n)$ where n is number of nodes.

Approach 2: Iterating method using Stack

The strategy is very similiar to the first method, the different is using stack.

Here is an illustration:



Java

```
public class Solution {
    public List < Integer > inorderTraversal(TreeNode root) {
        List < Integer > res = new ArrayList < > ();
        Stack < TreeNode > stack = new Stack < > ();
        TreeNode curr = root;
        while (curr != null || !stack.isEmpty()) {
            while (curr != null) {
                stack.push(curr);
                curr = curr.left;
            }
            curr = stack.pop();
            res.add(curr.val);
            curr = curr.right;
        }
        return res;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$.
- Space complexity : $O(n)$.

Approach 3: Morris Traversal

In this method, we have to use a new data structure-Threaded Binary Tree, and the strategy is as follows:

> Step 1: Initialize current as root
>
> Step 2: While current is not NULL,
>
> ```
> If current does not have left child
>
>     a. Add current’s value
>
>     b. Go to the right, i.e., current = current.right
>
> Else
>
>     a. In current's left subtree, make current the right child of the rightmost
>
>     b. Go to this left child, i.e., current = current.left
> ```

For example:

```
     1
   /   \
  2     3
 / \   /
4   5 6
```

First, 1 is the root, so initialize 1 as current, 1 has left child which is 2, the current's left subtree is

```
    2
   / \
  4   5
```

So in this subtree, the rightmost node is 5, then make the current(1) as the right child of 5. Set current = cuurent.left (current = 2). The tree now looks like:

```
    2
   / \
  4   5
       \
        1
         \
          3
         /
        6
```

For current 2, which has left child 4, we can continue with thesame process as we did above

```
   4
    \
     2
      \
       5
        \
         1
          \
           3
          /
         6
```

then add 4 because it has no left child, then add 2, 5, 1, 3 one by one, for node 3 which has left child 6, do the same as above. Finally, the inorder taversal is [4,2,5,1,6,3].

For more details, please check Threaded binary tree and Explaination of Morris Method

Java

```
class Solution {
    public List < Integer > inorderTraversal(TreeNode root) {
        List < Integer > res = new ArrayList < > ();
        TreeNode curr = root;
        TreeNode pre;
        while (curr != null) {
            if (curr.left == null) {
                res.add(curr.val);
                curr = curr.right; // move to next right node
            } else { // has a left subtree
                pre = curr.left;
                while (pre.right != null) { // find rightmost
                    pre = pre.right;
                }
                pre.right = curr; // put cur after the pre node
                TreeNode temp = curr; // store cur node
                curr = curr.left; // move cur to the top of the new tree
                temp.left = null; // original cur left be null, avoid infinite loops
            }
        }
        return res;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. To prove that the time complexity is $O(n)$, the biggest problem lies in finding the time complexity of finding the predecessor nodes of all the nodes in the binary tree. Intuitively, the complexity is $O(n \log n)$, because to find the predecessor node for a single node related to the height of the tree. But in fact, finding the predecessor nodes for all nodes only needs $O(n)$ time. Because a binary Tree with n nodes has $n - 1$ edges, the whole processing for each edges up to 2 times, one is to locate a node, and the other is to find the predecessor node. So the complexity is $O(n)$.
- Space complexity : $O(n)$. Arraylist of size n is used.

Rate this article:

Previous Next

Comments: 38

Sort By

Type comment here... (Markdown is supported)

Preview Post

droidgod ★ 137 March 11, 2018 3:11 AM Report

Morris traversal should be O(1) extra space. You cant count the solution set as extra space since that was what was asked in the first place.
The whole point of Morris traversal is to eliminate the need for extra space(either call stack or stack in interative method).
No point of doing all that extra work of modifying all the nodes for no gain in time or space complexity.

Read More

125 Share Reply

SHOW 4 REPLIES

anuragkalra ★ 55 March 1, 2019 4:56 AM

Solution in Approach #1 does unnecessary check in line 10 and line 14. We are already checking whether (root.left != null) when computing helper(root.left). Solution can be optimized to:

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
```

Read More

55 Share Reply

SHOW 4 REPLIES

jianchao-li ★ 14335 July 15, 2018 10:29 AM

The Morris solution above modifies the tree. The following one recovers it :-)

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> nodes = new ArrayList<>();
```

Read More

41 Share Reply

SHOW 1 REPLY

yousuf2 ★ 31 November 21, 2018 1:56 PM

Why not use a BST to illustrate? The output will be sorted and it'll help foster learning.

18 Share Reply

fantasyfish667 ★ 19 December 30, 2018 3:39 PM Report

Could anyone explain why the time complexity of the Morris method is O(n)? I can't understand this line, 'Because a binary Tree with nn nodes has n-1n−1 edges, the whole processing for each edges up to 2 times, one is to locate a node, and the other is to find the predecessor node.'

5 Share Reply

SHOW 3 REPLIES

rkarunia ★ 7 November 5, 2017 9:20 AM

C++ solution
Using a variable to determine whether to go left or right

```
class Solution {
public:
```

Read More

4 Share Reply

ManuelP ★ 818 September 3, 2017 4:20 AM Report

That's not Morris traversal, since that doesn't destroy the tree (which yours does).

4 Share Reply

SHOW 4 REPLIES

JustKeepCodinggg ★ 58 February 12, 2020 6:27 AM

Why is this a medium question?

6 Share Reply

edzvh ★ 125 July 13, 2019 9:05 PM

How is Approach 1 space ***average*** case `O(log n)` ?? Surely that's ***best*** case??

2 Share Reply

SHOW 2 REPLIES

sohammehta ★ 1137 July 13, 2018 2:17 AM Report

Morris Traversal O(n) Time O(1) Space

See Detailed solution and Pseudo Code

Read More

3 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 ›