105. Construct Binary Tree From Preorder and Inorder Traversal

Nov. 2, 2018 | 77.5K views

Average Rating: 3.74 (80 votes)

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

For example, given

```
preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
```

Return the following binary tree:

```
  3
 / \
9  20
  /  \
 15   7
```

Solution

How to traverse the tree

There are two general strategies to traverse a tree:

- *Breadth First Search* (`BFS`)

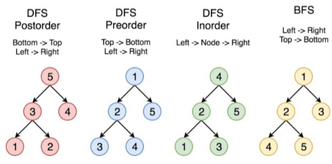
  We scan through the tree level by level, following the order of height, from top to bottom. The nodes on higher level would be visited before the ones with lower levels.

- *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 construct a binary tree from its preorder and inorder traversal.

Approach 1: Recursion

Tree definition

First of all, here is the definition of the `TreeNode` which we would use.

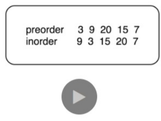
Java | Python — Copy

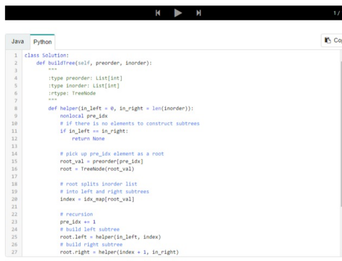
```
# Definition for a binary tree node.
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

Algorithm

As discussed above the preorder traversal follows `Root -> Left -> Right` order, that makes it very convenient to construct the tree from its root.

Let's do it. The first element in the *preorder* list is a root. This root splits *inorder* list into left and right subtrees. Now one have to pop up the root from preorder list since it's already used as a tree node and then repeat the step above for the left and right subtrees.





Java | Python — Copy

```
class Solution:
    def buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """
        def helper(in_left = 0, in_right = len(inorder)):
            nonlocal pre_idx
            # if there is no elements to construct subtrees
            if in_left == in_right:
                return None

            # pick up pre_idx element as a root
            root_val = preorder[pre_idx]
            root = TreeNode(root_val)

            # root splits inorder list
            # into left and right subtrees
            index = idx_map[root_val]

            # recursion
            pre_idx += 1
            # build left subtree
            root.left = helper(in_left, index)
            # build right subtree
            root.right = helper(index + 1, in_right)
```

Complexity analysis

- Time complexity : $O(N)$. Let's compute the solution with the help of master theorem $T(N) = aT\left(\frac{N}{b}\right) + \Theta(N^d)$. The equation represents dividing the problem up into a subproblems of size $\frac{N}{b}$ in $\Theta(N^d)$ time. Here one divides the problem in two subproblemes `a = 2`, the size of each subproblem (to compute left and right subtree) is a half of initial problem `b = 2`, and all this happens in a constant time `d = 0`. That means that $\log_b(a) > d$ and hence we're dealing with case 1 that means $O(N^{\log_b(a)}) = O(N)$ time complexity.

- Space complexity : $O(N)$, since we store the entire tree.

Rate this article:

Previous Next

Comments: 20

Sort By

Type comment here... (Markdown is supported)

Preview Post

kilcars ★ 125 November 23, 2018 11:05 PM

This problem can be solved in O(n) time with two optimizations:

1. There is no need to copy arrays, we can use start & end indexes for inorder & preorder arrays instead.
2. We can use a HashTable/Map to store the indexes of the elements in the inorder array. By this way

Read More

49 Share Reply

SHOW 5 REPLIES

hello_world_cn ★ 279 December 25, 2018 8:56 AM

There are better solutions than this one.

25 Share Reply

SHOW 1 REPLY

ZitaoWang ★ 1022 January 2, 2019 11:38 AM

An iterative solution with `O(n)` time, `O(n)` space.

```
class Solution:
    def buildTree(self, preorder, inorder):
        """
```

Read More

21 Share Reply

zhang-peter ★ 26 March 21, 2019 11:36 AM

python solution Recursion way 7 lines

```
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        def list2preorder(i=0):
```

Read More

13 Share Reply

SHOW 3 REPLIES

leetcodefan ★ 1909 February 7, 2019 10:29 AM

Thanks for updating the solution. Really appreciate it.

6 Share Reply

SHOW 1 REPLY

Cobo ★ 6 November 15, 2018 8:39 AM

Why the complexity is O(N)? I think the divide and conquer formula is T(n) = 2T(1/2*n)+O(n). So by master theory, the complexity should be O(N log N). Am I wrong?

5 Share Reply

SHOW 2 REPLIES

hongyu9310 ★ 34 February 7, 2019 9:08 PM

Python O(n), O(n)

class Solution(object):

```
def buildTree(self, preorder, inorder):
```

Read More

2 Share Reply

SHOW 1 REPLY

softwareshortcut ★ 424 June 6, 2019 2:12 AM

Using a HashTable to store the indices is key here. Otherwise you'd quickly end up with a N^2 solution because for each call you'd need to search for the position of preorder[x] inside of inorder (unsorted) array.

1 Share Reply

husaynhakeem ★ 1 November 4, 2018 3:21 AM

For the java solution, isn't the space complexity O(N*N), since in_order is split in each recursive call and its subarrays are stored in variables?

1 Share Reply

bestnick ★ 311 November 3, 2018 7:56 AM

The problem definition should clarify that the resultant tree should be as balanced as possible or trees like 1->2->3->4->5 (no left child, only right child, hence linked list like or vice versa) are out of scope.

0 Share Reply

SHOW 1 REPLY