102. Binary Tree Level Order Traversal (/problems/binary-tree-level-order-traversal/)

March 15, 2019 | 23.4K views

Average Rating: 4.95 (22 votes)

Given a binary tree, return the *level order* traversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree `[3,9,20,null,null,15,7]`,

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

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

Solution

How to traverse the tree

There are two general strategies to traverse a tree:

- *Depth First Search* (`DFS`)

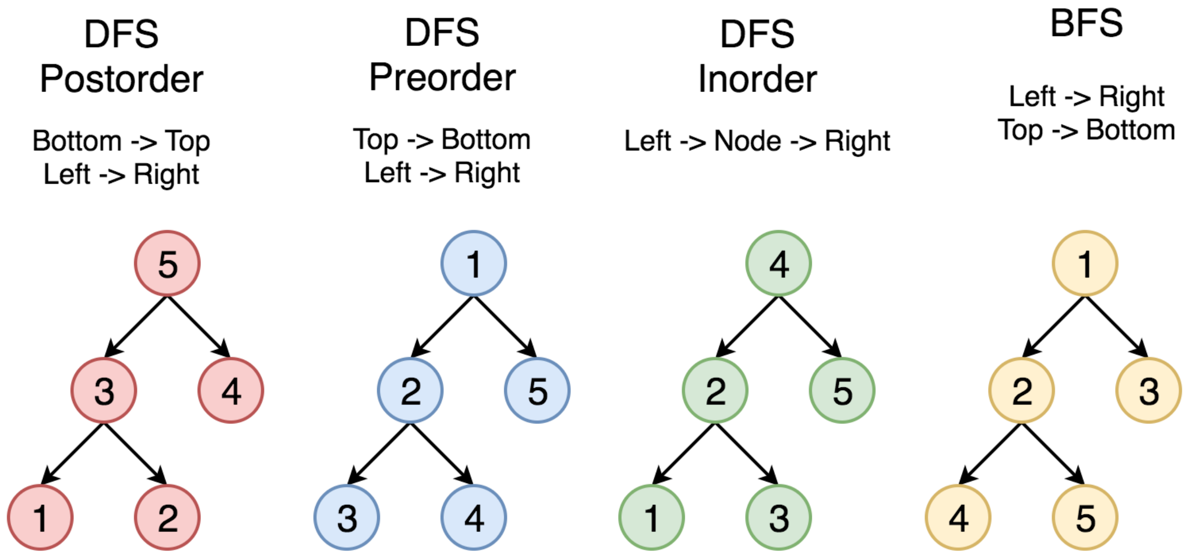
  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach all the way down to certain leaf, and then back to root to reach another branch.

  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.

- *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.

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 implement split-level BFS traversal : `[[1], [2, 3], [4, 5]]`.

Approach 1: Recursion

Algorithm

The simplest way to solve the problem is to use a recursion. Let's first ensure that the tree is not empty, and then call recursively the function `helper(node, level)`, which takes the current node and its level as the arguments.

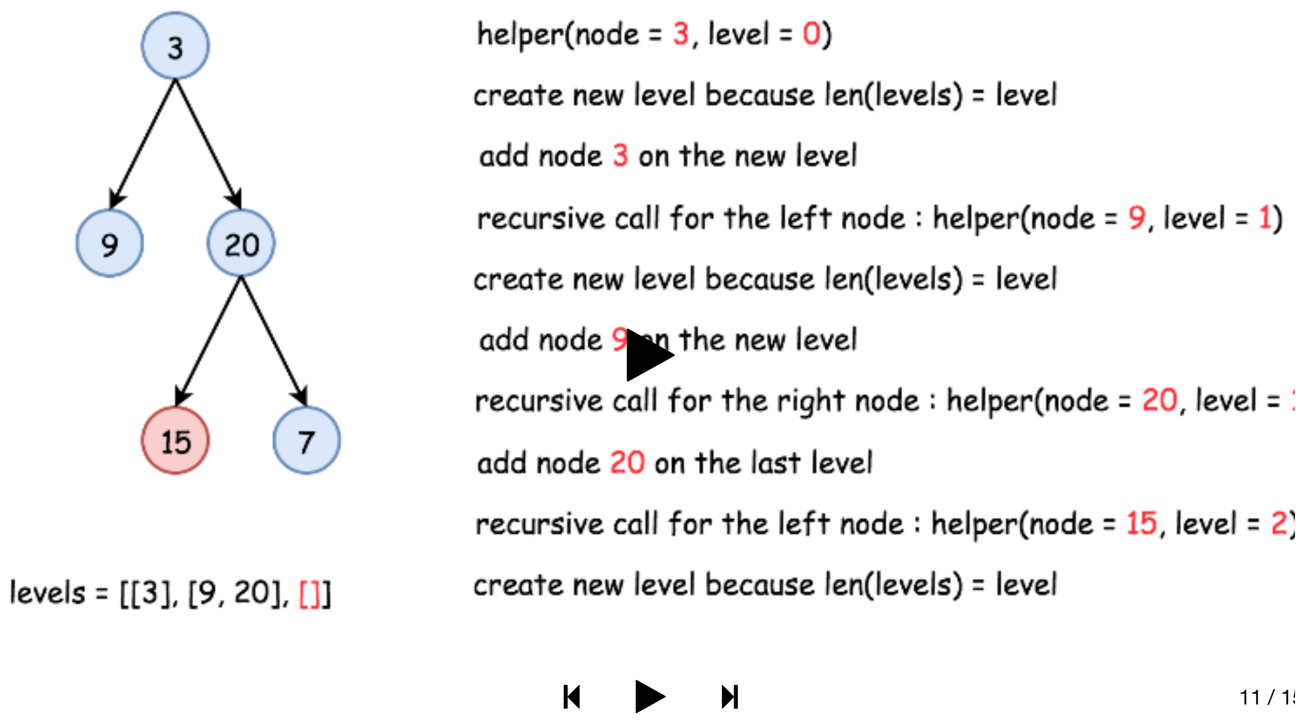
This function does the following :

- The output list here is called `levels`, and hence the current level is just a length of this list `len(levels)`. Compare the number of a current level `len(levels)` with a node level

`level`. If you're still on the previous level - add the new one by adding a new list into `levels`.

- Append the node value to the last list in `levels`.
- Process recursively child nodes if they are not `None` : `helper(node.left / node.right, level + 1)`.

Implementation



helper(node = 3, level = 0)
create new level because len(levels) = level
add node 3 on the new level
recursive call for the left node : helper(node = 9, level = 1)
create new level because len(levels) = level
add node 9 on the new level
recursive call for the right node : helper(node = 20, level =
add node 20 on the last level
recursive call for the left node : helper(node = 15, level = 2
create new level because len(levels) = level

Java Python Articles > 102. Binary Tree Level Order Traversal Copy

```
class Solution:
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        levels = []
        if not root:
            return levels

        def helper(node, level):
            # start the current level
            if len(levels) == level:
                levels.append([])

            # append the current node value
            levels[level].append(node.val)

            # process child nodes for the next level
            if node.left:
                helper(node.left, level + 1)
            if node.right:
                helper(node.right, level + 1)

        helper(root, 0)
        return levels
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ since each node is processed exactly once.
- Space complexity : $\mathcal{O}(N)$ to keep the output structure which contains `N` node values.

Approach 2: Iteration

Algorithm

The recursion above could be rewritten in the iteration form.

Let's keep nodes of each tree level in the *queue* structure, which typically orders elements in a FIFO (first-in-first-out) manner. In Java one could use `LinkedList` implementation of the `Queue` interface (https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html). In Python using `Queue` structure (https://docs.python.org/3/library/queue.html) would be an overkill since it's designed for a safe exchange between multiple threads and hence requires locking which leads to a performance loose. In Python the queue implementation with a fast atomic `append()` and `popleft()` is `deque` (https://docs.python.org/3/library/collections.html#collections.deque).

The zero level contains only one node `root`. The algorithm is simple :

- Initiate queue with a `root` and start from the level number `0` : `level = 0`.

- While queue is not empty :
  - Start the current level by adding an empty list into output structure `levels`.
  - Compute how many elements should be on the current level : it's a queue length.
  - Pop out all these elements from the queue and add them into the current level.
  - Push their child nodes into the queue for the next level.
  - Go to the next level `level++`.

Implementation

Java Python Copy

```
from collections import deque
class Solution:
    def levelOrder(self, root):
        """
        :type root: TreeNode
        :rtype: List[List[int]]
        """
        levels = []
        if not root:
            return levels

        level = 0
        queue = deque([root,])
        while queue:
            # start the current level
            levels.append([])
            # number of elements in the current level
            level_length = len(queue)

            for i in range(level_length):
                node = queue.popleft()
                # fulfill the current level
                levels[level].append(node.val)

                # add child nodes of the current level
                # in the queue for the next level
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ since each node is processed exactly once.
- Space complexity : $\mathcal{O}(N)$ to keep the output structure which contains `N` node values.

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

Rate this article:

Previous (/articles/gas-station/) Articles > 102. Binary Tree Level Order Traversal Next (/articles/sudoku-solver/)

Comments: 6 Sort By

Type comment here... (Markdown is supported)

Preview Post

axelramar9 (axelramar9) ★ 68 April 17, 2019 10:54 PM

(/axelramar9)

Approach 1 can be simplified as:

```
class Solution(object):
    def levelOrder(self, root):
        levels = []
```

Read More

4 Share Reply

yoursungjin (yoursungjin) ★ 10 March 30, 2019 9:51 AM

(/yoursungjin)

For the solution 2, you don't need the level variable.
you can go with

```
        levels.get(levels.size()-1).add(node.val);
```

3 Share Reply

starsheep (starsheep) ★ 2 August 2, 2019 11:56 AM

(/starsheep)

For both approaches, it was my understanding that the output structure is not taken into account when calculating space complexity in these problems. Am I wrong? If not, should the correct space complexity be O(H) where H is the height of the tree (recursion stack)?

1 Share Reply

rheinze08 (rheinze08) ★ 2 July 25, 2019 7:21 AM

(/rheinze08)

Python Stack Implementation -- follows the same methodology as the levels in Approach 1, albeit probably higher complexity due to level_dict scan at the end (equal to number of levels)

```
class Solution:
```

Read More

0 Share Reply

(/kohok47)

kohok47 (kohok47) ★ 0 April 9, 2019 9:52 AM

Is it possible to create a recursive function that does a breadth-first binary tree traversal in JavaScript? It seems like it should be similar to the approach above, but my attempt is not working thus far, and this post (https://stackoverflow.com/questions/33703019/breadth-first-traversal-of-a-tree-in-javascript (https://stackoverflow.com/questions/33703019/breadth-first-traversal-of-a-tree-in-javascript)) claims that it is not possible: “DFS is easy to

Read More

0 Share Reply

SHOW 1 REPLY

(/zhang-peter)

zhang-peter (zhang-peter) ★ 6 March 25, 2019 11:10 PM

as for python solution of Iteration, why i use list as a queue is much slower than using deque?

0 Share Reply

SHOW 2 REPLIES