107. Binary Tree Level Order Traversal II

May 30, 2020 | 10.8K views

Average Rating: 4.64 (11 votes)

Previous Next

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

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

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

return its bottom-up level order traversal as:

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

Solution

How to traverse the tree

There are two general strategies to traverse a tree:

- *Depth First Search* (`DFS`)

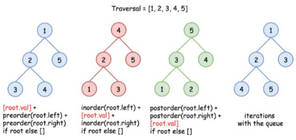
  In this strategy, we adopt the `depth` as the priority, so that one would start from a root and reach all the way down to a certain leaf, and then back to root to reach another branch.

  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 a higher level would be visited before the ones on lower levels.

In the following figure the nodes are enumerated 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 : `[[4, 5], [2, 3], [1]]`. That means we could use one of the `Node->Left->Right` techniques: BFS or DFS Preorder.

We already discussed three different ways to implement iterative BFS traversal with the queue, and compared iterative BFS vs. iterative DFS. Let's use this article to discuss the two most simple and fast techniques:

- Recursive DFS.
- Iterative BFS with two queues.

> Note, that both approaches are root-to-bottom traversals, and we're asked to provide bottom-up output. To achieve that, the final result should be reversed.

Approach 1: Recursion: DFS Preorder Traversal

Intuition

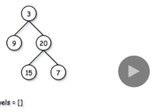
The first step is to ensure that the tree is not empty. The second step is to implement the recursive function `helper(node, level)`, which takes the current node and its level as the arguments.

Algorithm for the Recursive Function

Here is its implementation:

- Initialize the output list `levels`. The length of this list determines which level is currently updated. You should compare this level `len(levels)` with a node level `level`, to ensure that you add the node on the correct level. If you're still on the previous level - add the new level by adding a new list into `levels`.
- Append the node value to the last level in `levels`.
- Process recursively child nodes if they are not `None`: `helper(node.left / node.right, level + 1)`.

Implementation




```
1 / 16
```

Java | Python — Copy

```
class Solution:
    def levelOrderBottom(self, root: TreeNode) -> 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[::-1]
```

Complexity Analysis

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

Approach 2: Iteration: BFS Traversal

Algorithm

The recursion above could be rewritten in the iteration form.

Let's keep each tree level in the *queue* structure, which typically orders elements in a FIFO (first-in-first-out) manner. In Java one could use `ArrayDeque` implementation of the `Queue` interface. In Python using `Queue` structure would be an overkill since it's designed for a safe exchange between multiple threads and hence requires locking which leads to a performance downgrade. In Python the queue implementation with a fast atomic `append()` and `popleft()` is `deque`.

Algorithm

- Initialize two queues: one for the current level, and one for the next. Add root into `nextLevel` queue.
- While `nextLevel` queue is not empty:
  - Initialize the current level `currLevel = nextLevel`, and empty the next level `nextLevel`.
  - Iterate over the current level queue:
    - Append the node value to the last level in `levels`.
    - Add first *left* and then *right child* node into `nextLevel` queue.
- Return reversed `levels`.

Implementation

Java | Python — Copy

```
class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
        levels = []
        next_level = deque([root])

        while root and next_level:
            curr_level = next_level
            next_level = deque()
            levels.append([])

            for node in curr_level:
                # append the current node value
                levels[-1].append(node.val)
                # process child nodes for the next level
                if node.left:
                    next_level.append(node.left)
                if node.right:
                    next_level.append(node.right)

        return levels[::-1]
```

Complexity Analysis

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

Rate this article:

Previous Next

Comments: 6

Sort By

Type comment here... (Markdown is supported)

Preview Post

jyshi302 ★ 1 · July 2, 2020 11:20 PM

BFS, use Linked List to store the result

```
class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        LinkedList<List<Integer>> res=new LinkedList<>();
```

Read More

3 Share Reply

SHOW 1 REPLY

super_saiyan_2 ★ 19 · July 2, 2020 10:28 PM

Any opinions on my solution? Just a classic BFS, where we utilize a linked list to be able to add elements in the beginning fast.

```
class Solution {
```

Read More

0 Share Reply

neospeed03 ★ 3 · 3 days ago

I implemented the first approach using the given algorithm.

the algo says that "Append the node value to the last level in levels." So I wrote levels[-1].append(node.val) in my code. But in the given implementation, the author is adding the current node value to the 'levels' list in the levels structure like this:

Read More

0 Share Reply

swapnil2927 ★ 42 · July 3, 2020 8:42 AM

Why add another question, if you are going to be taking the solution of #102 and call Collections.reverse on the final solution. Maybe we can include better, different or cleverer solutions in this article.

Read More

0 Share Reply

LeetCoding_Master ★ 138 · July 2, 2020 8:00 PM

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

Read More

0 Share Reply

gdkou90 ★ 64 · July 2, 2020 7:45 PM

This problem is exactly the one that I was asked during an interview, which was followed by problem 102.

0 Share Reply

SHOW 3 REPLIES