Reverse Level Order Traversal (easy)

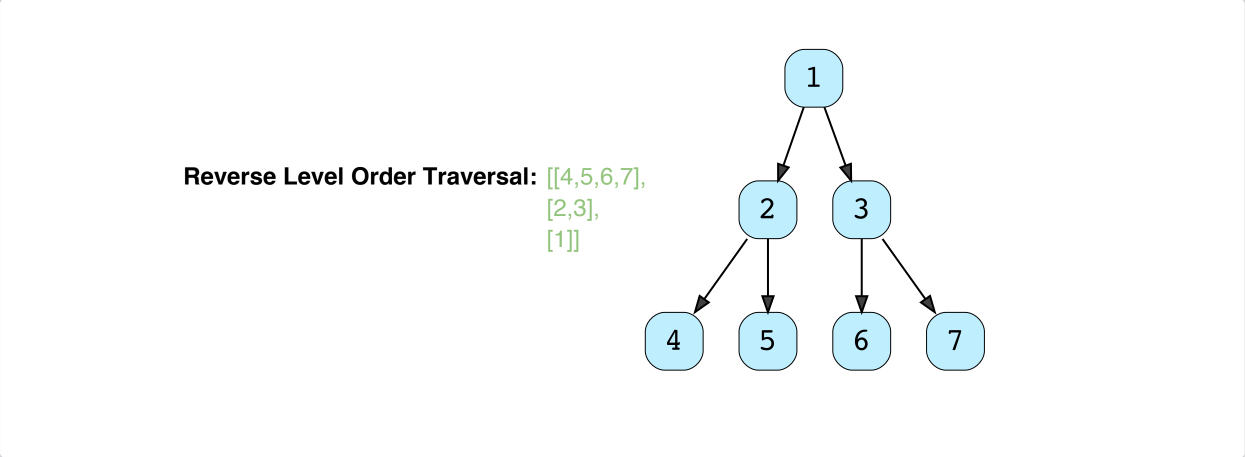
We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
  - Time complexity
  - Space complexity

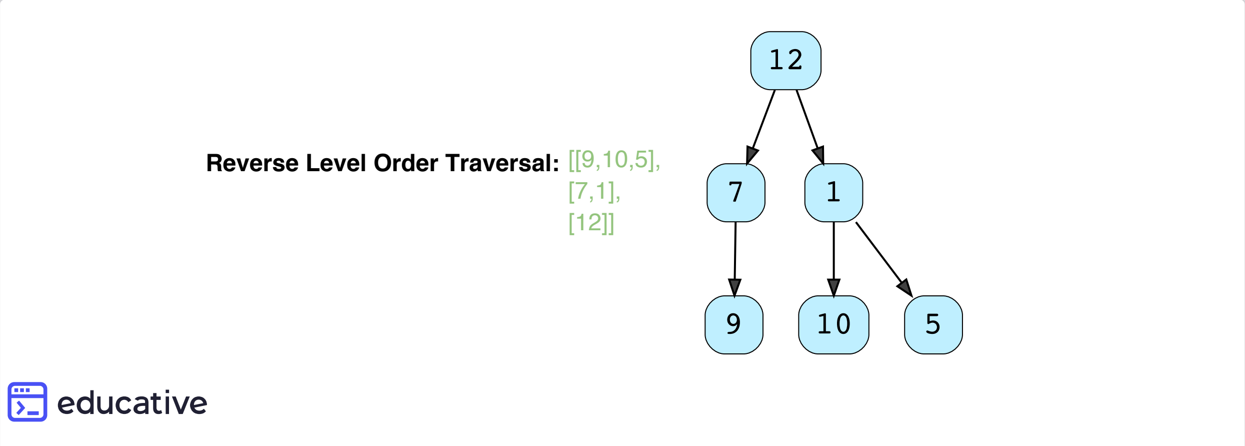
Problem Statement

Given a binary tree, populate an array to represent its level-by-level traversal in reverse order, i.e., the **lowest level comes first**. You should populate the values of all nodes in each level from left to right in separate sub-arrays.

Example 1:



Example 2:



Try it yourself

Try solving this question here:

Java Python3 JS C++

```
from collections import deque

class TreeNode:
  def __init__(self, val):
    self.val = val
    self.left, self.right = None, None

def traverse(root):
  result = deque()
  # TODO: Write your code here
  return result

def main():
  root = TreeNode(12)
  root.left = TreeNode(7)
  root.right = TreeNode(1)
  root.left.left = TreeNode(9)
  root.right.left = TreeNode(10)
  root.right.right = TreeNode(5)
  print("Reverse level order traversal: " + str(traverse(root)))


main()

```

Solution

This problem follows the Binary Tree Level Order Traversal (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5726607939469312/) pattern. We can follow the same **BFS** approach. The only difference will be that instead of appending the current level at the end, we will append the current level at the beginning of the result list.

Code

Here is what our algorithm will look like; only the highlighted lines have changed. Please note that, for **Java**, we will use a `LinkedList` instead of an `ArrayList` for our result list. As in the case of `ArrayList`, appending an element at the beginning means shifting all the existing elements. Since we need to append the level array at the beginning of the result list, a `LinkedList` will be better, as this shifting of elements is not required in a `LinkedList`. Similarly, we will use a double-ended queue (deque) for **Python**, **C++**, and **JavaScript**.

Java Python3 C++ JS

```
from collections import deque


```

```
class TreeNode:
  def __init__(self, val):
    self.val = val
    self.left, self.right = None, None


def traverse(root):
  result = deque()
  if root is None:
    return result

  queue = deque()
  queue.append(root)
  while queue:
    levelSize = len(queue)
    currentLevel = []
    for _ in range(levelSize):
      currentNode = queue.popleft()
      # add the node to the current level
      currentLevel.append(currentNode.val)
      # insert the children of current node in the queue
      if currentNode.left:
        queue.append(currentNode.left)
      if currentNode.right:
        queue.append(currentNode.right)
```

Time complexity

The time complexity of the above algorithm is $O(N)$, where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

Space complexity

The space complexity of the above algorithm will be $O(N)$ as we need to return a list containing the level order traversal. We will also need $O(N)$ space for the queue. Since we can have a maximum of $N/2$ nodes at any level (this could happen only at the lowest level), therefore we will need $O(N)$ space to store them in the queue.

← Back

Binary Tree Level Order Traversal (easy)

Next →

Zigzag Traversal (medium)

☑ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/reverse-level-order-traversal-easy__pattern-tree-breadth-first-search__grokking-the-coding-interview-patterns-for-coding-questions)