Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions
40% completed
Search Course
Introduction
Binary Tree Level Order Traversal (easy)
Reverse Level Order Traversal (easy)
Zigzag Traversal (medium)
Level Averages in a Binary Tree (easy)

Reverse Level Order Traversal (easy)

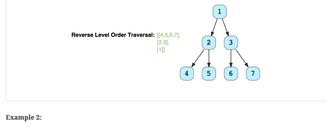
We'll cover the following

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

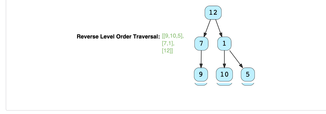
Problem Statement

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++

```
import java.util.*;

class TreeNode {
  int val;
  TreeNode left;
  TreeNode right;
  }
};

class ReverseLevelOrderTraversal {
  public static List<List<Integer>> traverse(TreeNode root) {
    List<List<Integer>> result = new LinkedList<List<Integer>>();
    // TODO: Write your code here
    return result;
  }

  public static void main(String[] args) {
    TreeNode root = new TreeNode(12);
    root.left = new TreeNode(7);
    root.right = new TreeNode(1);
    root.left.left = new TreeNode(9);
    root.right.left = new TreeNode(10);
    root.right.right = new TreeNode(5);
    List<List<Integer>> result = ReverseLevelOrderTraversal.traverse(root);
    System.out.println("Reverse level order traversal: " + result);
  }
}
```

Run | Save | Reset

Solution

This problem follows the Binary Tree Level Order Traversal 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**,

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.

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around. See how

Grokking the Coding Interview: Patterns for Coding Questions
40% completed
Search Course
Introduction
Binary Tree Level Order Traversal (easy)
Reverse Level Order Traversal (easy)
Zigzag Traversal (medium)
Level Averages in a Binary Tree (easy)

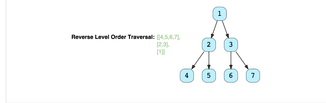
Back
Next

We''ll cover the following
- Problem Statement
- Try it yourself
- Solution
- Code
- Time complexity
- Space complexity

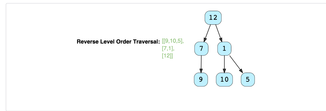
Problem Statement

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++

```
  int val;
  TreeNode left;
  TreeNode right;

  TreeNode(int x) {
    val = x;
  }
};

class ReverseLevelOrderTraversal {
  public static List<List<Integer>> traverse(TreeNode root) {
    List<List<Integer>> result = new LinkedList<List<Integer>>();
    // TODO: Write your code here
    return result;
  }

  public static void main(String[] args) {
    TreeNode root = new TreeNode(12);
    root.left = new TreeNode(7);
    root.right = new TreeNode(1);
    root.left.left = new TreeNode(9);
    root.right.left = new TreeNode(10);
    root.right.right = new TreeNode(5);
    List<List<Integer>> result = ReverseLevelOrderTraversal.traverse(root);
    System.out.println("Reverse level order traversal: " + result);
  }
}
```

Run | Save | Reset

Solution

This problem follows the Binary Tree Level Order Traversal 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

```
    TreeNode currentNode = queue.poll();
    // add the node to the current level
    currentLevel.add(currentNode.val);
    // insert the children of current node to the queue
    if (currentNode.left != null)
      queue.offer(currentNode.left);
    if (currentNode.right != null)
      queue.offer(currentNode.right);
    }
    // append the current level at the beginning
    result.add(0, currentLevel);
    }

    return result;
  }

  public static void main(String[] args) {
    TreeNode root = new TreeNode(12);
    root.left = new TreeNode(7);
    root.right = new TreeNode(1);
    root.left.left = new TreeNode(9);
    root.right.left = new TreeNode(10);
    root.right.right = new TreeNode(5);
    List<List<Integer>> result = ReverseLevelOrderTraversal.traverse(root);
    System.out.println("Reverse level order traversal: " + result);
  }
}
```

Run | Save | Reset

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.

Interviewing soon? We've partnered with Hired so that companies apply to you, instead of the other way around. See how