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)
- Minimum Depth of a Binary Tree (easy)

Binary Tree Level Order Traversal (easy)

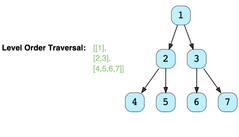
We'll cover the following

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

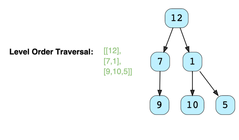
Problem Statement

Given a binary tree, populate an array to represent its level-by-level traversal. You should populate the values of all **nodes of 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;

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

class LevelOrderTraversal {
  public static List<List<Integer>> traverse(TreeNode root) {
    List<List<Integer>> result = new ArrayList<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 = LevelOrderTraversal.traverse(root);
    System.out.println("Level order traversal: " + result);
```

Run Save Reset

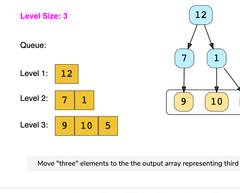
Solution

Since we need to traverse all nodes of each level before moving onto the next level, we can use the **Breadth First Search (BFS)** technique to solve this problem.

We can use a Queue to efficiently traverse in BFS fashion. Here are the steps of our algorithm:

1. Start by pushing the `root` node to the queue.
2. Keep iterating until the queue is empty.
3. In each iteration, first count the elements in the queue (let's call it `levelSize`). We will have these many nodes in the current level.
4. Next, remove `levelSize` nodes from the queue and push their `value` in an array to represent the current level.
5. After removing each node from the queue, insert both of its children into the queue.
6. If the queue is not empty, repeat from step 3 for the next level.

Let's take the example-2 mentioned above to visually represent our algorithm:



Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```
    for (int i = 0; i < levelSize; i++) {
      TreeNode currentNode = queue.poll();
      // add the node to the current level
      currentLevel.add(currentNode.val);
      // insert the children of current node in the queue
      if (currentNode.left != null)
        queue.offer(currentNode.left);
      if (currentNode.right != null)
        queue.offer(currentNode.right);
    }
    result.add(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 = LevelOrderTraversal.traverse(root);
  System.out.println("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.