Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

41% completed

Search Course

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)

# Zigzag Traversal (medium)

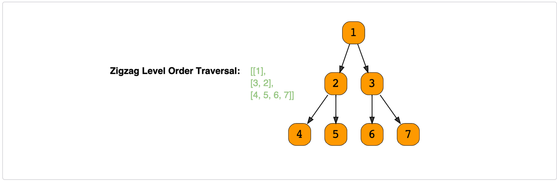
We'll cover the following



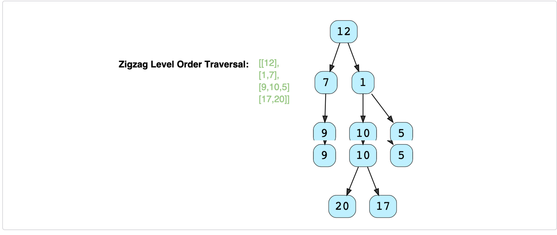
## Problem Statement

Given a binary tree, populate an array to represent its zigzag level order traversal. You should populate the values of all **nodes of the first level from left to right**, then **right to left for the next level** and keep alternating in the same manner for the following levels.

**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 ZigzagTraversal {
 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);
  root.right.left.left = new TreeNode(20);
  root.right.left.right = new TreeNode(17);
```

Run | Save | Reset

## Solution

This problem follows the Binary Tree Level Order Traversal pattern. We can follow the same **BFS** approach. The only additional step we have to keep in mind is to alternate the level order traversal, which means that for every other level, we will traverse similar to Reverse Level Order Traversal.

## Code

Here is what our algorithm will look like, only the highlighted lines have changed:

Java | Python3 | C++ | JS

```
import java.util.*;

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

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

class ZigzagTraversal {
 public static List<List<Integer>> traverse(TreeNode root) {
  List<List<Integer>> result = new ArrayList<List<Integer>>();
  if (root == null)
   return result;

  Queue<TreeNode> queue = new LinkedList<>();
  queue.offer(root);
  boolean leftToRight = true;
  while (!queue.isEmpty()) {
   int levelSize = queue.size();
   List<Integer> currentLevel = new LinkedList<>();
   for (int i = 0; i < levelSize; i++) {
    TreeNode currentNode = queue.poll();

    // add the node to the current level based on the traverse direction
```

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.