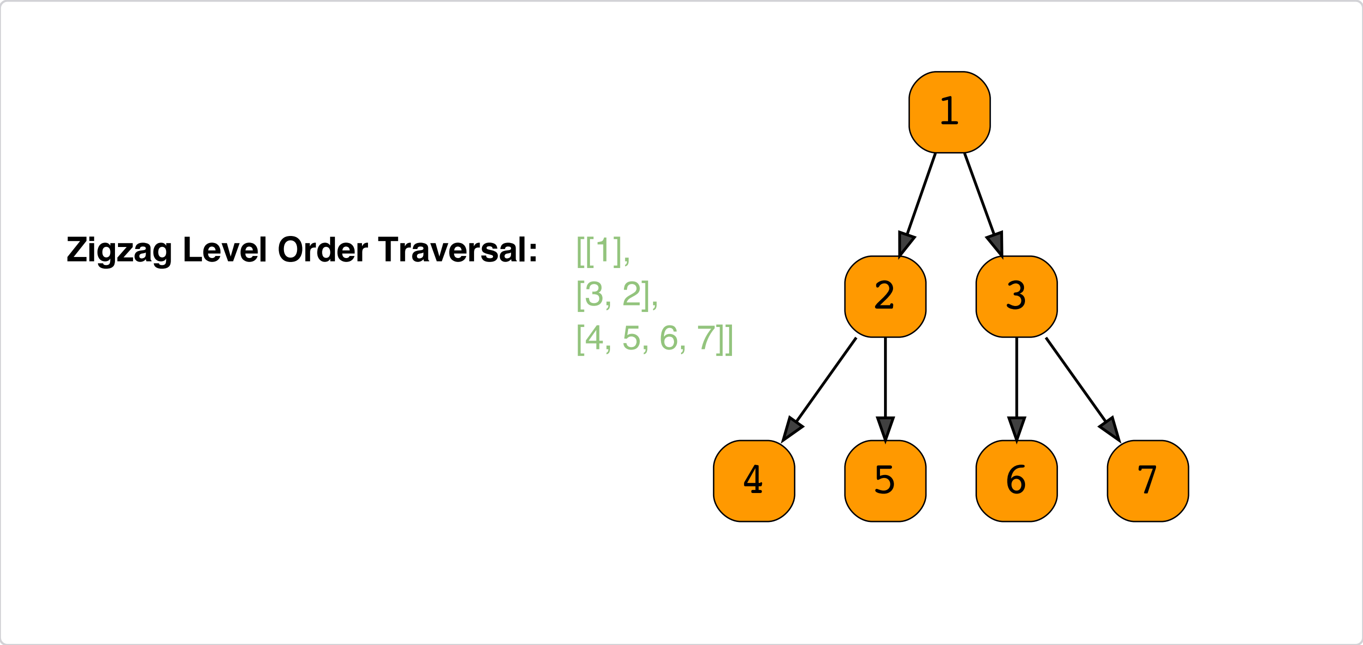
Zigzag Traversal (medium)

Problem Statement#

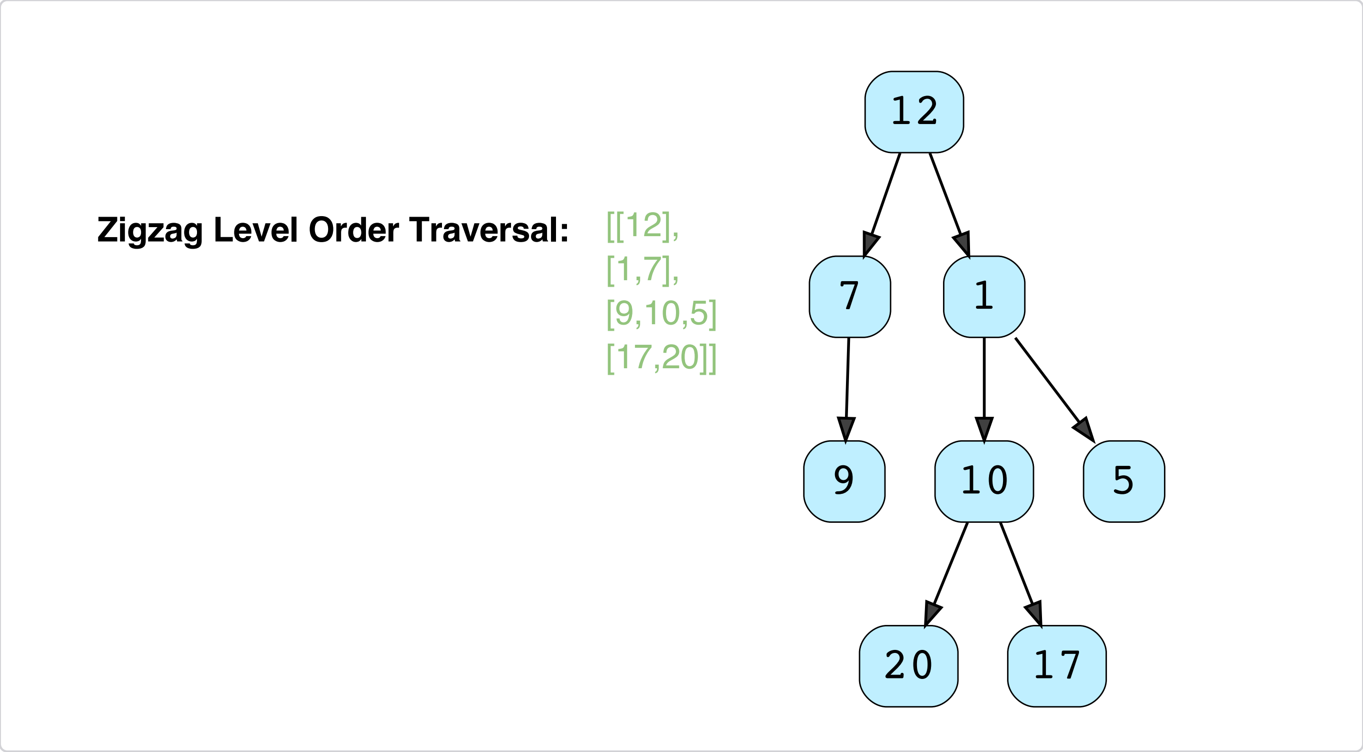
Given a binary tree, populate an array to represent its zigzag level order traversal. You should populate the values of all **nodes of the first level from left to right**, then **right to left for the next level** and keep alternating in the same manner for the following levels.

Example 1:



Zigzag Level Order Traversal: [[1],
[3, 2],
[4, 5, 6, 7]]

Example 2:



Zigzag Level Order Traversal: [[12],
[1,7],
[9,10,5]
[17,20]]

Try it yourself#

Try solving this question here:

Java | Python3 | JS | C++

```
class TreeNode {

  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
  }
};


const traverse = function(root) {
  result = [];
  // TODO: Write your code here
  return result;
};


var 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)
console.log(`Zigzag traversal: ${traverse(root)}`)
```

Run Save Reset

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 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 (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5765606242516992/).

Code#

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

Java | Python3 | C++ | JS

```
const Deque = require('./collections/deque'); //http://www.collectionsjs.com


class TreeNode {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
  }
}

function traverse(root) {
  result = [];
  if (root === null) {
    return result;
  }

  const queue = new Deque();
  queue.push(root);
  leftToRight = true;
  while (queue.length > 0) {
    levelSize = queue.length;
    currentLevel = new Deque();
    for (i = 0; i < levelSize; i++) {
      currentNode = queue.shift();

      // add the node to the current level based on the traverse direction
      if (leftToRight) {
        currentLevel.push(currentNode.val);
      } else {
        currentLevel.unshift(currentNode.val);
      }

      // insert the children of current node in the queue
      if (currentNode.left !== null) {
        queue.push(currentNode.left);
      }
      if (currentNode.right !== null) {
        queue.push(currentNode.right);
      }
    }
    result.push(currentLevel.toArray());
    // reverse the traversal direction
    leftToRight = !leftToRight;
  }

  return result;
}


const 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);
console.log(`Zigzag traversal: ${traverse(root)}`);
```

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.

← Back

Reverse Level Order Traversal (easy)

Next →

Level Averages in a Binary Tree (easy)

☑ Mark as Completed

Report an Issue