Solution: Binary Tree Zigzag Level Order Traversal

Let's solve the Binary Tree Zigzag Level Order Traversal problem using the Tree Breadth First Search pattern.

We'll cover the following

Statement

Given a binary tree, return its zigzag level order traversal. The zigzag level order traversal corresponds to traversing nodes from left to right for one level, right to left for the next level, and so on, reversing the direction after every level.

Constraints:

- The number of nodes in the tree is in the range 0 to 2000.
- $-100 \leq$ `node.data` ≤ 100

Solution

In this problem, we're required to proceed level by level through the tree, performing certain computations along the way. Therefore, the solution to this problem naturally maps to the tree breadth-first search pattern. In this pattern, we maintain a queue to keep track of all the nodes in a level before going to the next level.

As per the problem statement, we need to traverse the tree in a level-by-level zigzag order. We'll need to modify the basic breadth-first search (BFS) to get our desired output, a zigzag order. We can use a deque (a double-ended queue) data structure to add elements on both ends.

We alternate between using the first-in-first-out (FIFO) and the last-in-first-out (LIFO) schemes for each level. To ensure FIFO ordering, we append new elements to the deque's tail. For LIFO, we append the new elements to the head of the deque.

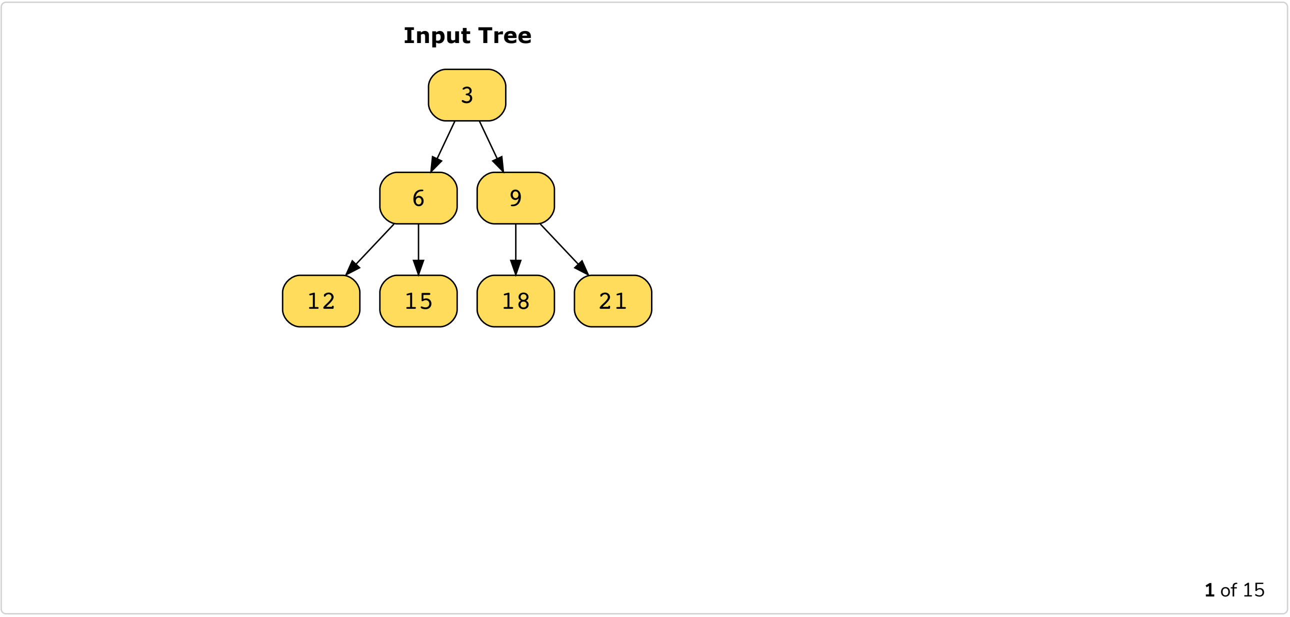
Let's review the steps to implement the solution:

1. Initialize a 2D array, `results`, to store the output.
2. Add the `root` of the tree to the deque for the level order traversal.
3. Initialize the order flag, `reverse`, which we will use to alternate between FIFO and LIFO schemes.
   - The `reverse` flag will be initially set to FALSE but will alternate its value at every other level.
   - If `reverse` is FALSE, we'll pop nodes from the start (left) of the deque and push nodes at the back (right) of the deque. In this manner, the first node to be visited at the next level (when `reverse` will be TRUE) will be the right child of the last node in the current level. This will result in the right-to-left traversal of the tree.

- If `reverse` is TRUE, we pop nodes from the back (right) of the deque and push nodes at the start (left) of the deque. This means that the first node to be visited at the next level (when `reverse` will be FALSE) will be the left child of the first node in the current level. This will result in the left-to-right traversal of the tree.

4. Pop from the deque either from the back or front on an alternate basis according to the value of the `reverse` switch.
5. Append the current node's value to that list in `results`, which corresponds to the current level.
6. Repeat steps 4 and 5 until the deque is empty.

To get a clearer understanding of the algorithm, let's look at the slides below:



Let's look at the code for this solution below:

Java

main.java

BinaryTree.java

TreeNode.java

```java
class ZigzagTraversal {

  public static List<List<Integer>> zigzagLevelOrder(TreeNode<Integer> root)
    // If the root is NULL, return an empty list
    if (root == null) {
      return new ArrayList<List<Integer>>();
    }
```

```java
    dq.add(root);

    // Creating an empty list to store the results
    List<List<Integer>> results = new ArrayList<List<Integer>>();

    // Creating a flag to indicate the direction of the traversal
    boolean reverse = false;

    // Traverse the tree
    while(!dq.isEmpty()){
      // Getting the size of the current level
      int size = dq.size();
      // Insert an empty list at the end of the 'results' list
      results.add(new ArrayList<Integer>());

      // Traverse the nodes in the current level
      for(int i = 0; i < size; i++){
        // Check the direction of the traversal
```

Binary Tree Zigzag Level Order Traversal

Time complexity

The time complexity will be $O(n)$, where n represents the number of nodes in the tree.

Space complexity

The space complexity will be $O(n)$. This is the space occupied by the deque.

Back

Binary Tree Zigzag Le...

Next

Populating Next Right...

✓ Mark as Completed