PROBLEM

ZigZag Tree Traversal

Medium Accuracy: 54.05% Submissions: 144K+ Points: 4

Given a binary tree with n nodes. Find the zig-zag level order traversal of the binary tree.

Example 1:

Input:

Output:
1 3 2 4 5 6 7

Example 2:

Input:

Output:
7 7 9 8 8 6 9 10

Your Task:
You don't need to read input or print anything. Your task is to complete the function **zigZagTraversal()** which takes the root node of the Binary Tree as its input and returns a list containing the node values as they appear in the zig-zag level-order traversal of the tree.

Expected Time Complexity: O(n).
Expected Auxiliary Space: O(n).

Constraints:
$1 <= n <= 10^5$

CODE

```
/*class Node
{
   int data;
   Node left,right;
   Node(int d)
   {
      data=d;
      left=right=null;
   }
}*/
```

```
class GFG {
   // Function to store the zigzag order traversal of tree in a list.
   ArrayList<Integer> zigZagTraversal(Node root) {
      ArrayList<Integer> result = new ArrayList<Integer>();
      if (root == null) {
         return result;
      }

      Queue<Node> q = new LinkedList<Node>();
      q.add(root);
      boolean flag = false;

      while (!q.isEmpty()) {
         int n = q.size();
         ArrayList<Integer> currentLevel = new ArrayList<>();

         for (int i = 0; i < n; i++) {
            Node curr = q.poll();
            currentLevel.add(curr.data);

            if (curr.left != null) {
               q.add(curr.left);
            }

            if (curr.right != null) {
               q.add(curr.right);
            }
         }

         if (flag) {
            Collections.reverse(currentLevel);
         }

         result.addAll(currentLevel);
         flag = !flag;
      }

      return result;
   }
}
```

OUTPUT

Compilation Completed

For Input:

3 2 1

Your Output:

3 1 2

Expected Output:

3 1 2

EXPLANATION

Purpose:

This code is for traversing a binary tree in a zigzag pattern, meaning it alternates between going left to right and right to left at each level.

How it Works:

- We start from the root of the tree.
- We use a queue to keep track of the nodes we need to visit.
- We process each level of the tree one by one.

At each level:

- We visit each node from left to right.
- If there's a left child, we add it to the queue.
- If there's a right child, we add it to the queue.
- We store the values of the nodes in the current level.
- We switch the direction of traversal (zigzag) for the next level.
- Finally, we return all the values collected during traversal in zigzag order.

Explanation:

- We start with an empty result list.
- If the tree is empty (root is null), we return the empty result list.
- We initialize a queue and add the root node to it.
- We use a flag to keep track of the traversal direction.
- While the queue is not empty:
- We process each level:
- We dequeue nodes and add their values to the current level list.
- If the dequeued node has children, we add them to the queue.
- After processing all nodes at the current level:

- If the flag is true, we reverse the current level list to achieve zigzag direction.
- We add all values from the current level list to the result list.
- We toggle the flag for the next level.

Finally, we return the result list containing the zigzag traversal order.