

PROBLEM

ZigZag Tree Traversal

Medium

Accuracy: 54.05%

Submissions: 144K+

Points: 4

Maximize your earnings for your published articles in Dev Scripster 2024!
Gain recognition & extra compensation while elevating your tech profile.

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

Example 1:

Input:

```

      1
     / \
    2   3
   / \ / \
  4  5 6  7

```

Output:

```
1 3 2 4 5 6 7
```

Example 2:

Input:

```

      7
     / \
    9   7
   / \ / \
  8  8 6  9
   / \
  10  9

```

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:

$1 \leq n \leq 10^5$

CODE

#User function Template for python3

```
'''
```

```
class Node:
```

```
    def __init__(self, val):
```

```
        self.data = val
```

```
        self.left = None
```

```
        self.right = None
```

```
'''
```

```
class Solution:
```

```
    #Function to store the zig zag order traversal of tree in a list.
```

```
    def zigZagTraversal(self, root):
```

```
# Your code here
```

```
out = []
```

```
queue = [root]
```

```
next_queue = []
```

```
forward = False
```

```
while queue:
```

```
    for node in queue:
```

```
        out.append(node.data)
```

```
        if not forward:
```

```
            if node.left is not None and node.left.data != "N":
```

```
                next_queue.append(node.left)
```

```
            if node.right is not None and node.right.data != "N":
```

```
                next_queue.append(node.right)
```

```
        else:
```

```
            if node.right is not None and node.right.data != "N":
```

```
                next_queue.append(node.right)
```

```
            if node.left is not None and node.left.data != "N":
```

```
                next_queue.append(node.left)
```

```
    next_queue.reverse()
```

```
    queue = next_queue.copy()
```

```
    next_queue = []
```

```
    forward = not forward
```

```
return out
```

EXPLANATION

Initialization:

```
def zigZagTraversal(self, root):
```

```
    out = []
    queue = [root]
    next_queue = []
    forward = False
```

out is initialized as an empty list to store the zigzag traversal.

queue is initialized with the root node to start the traversal from the root.

next_queue is initialized to store the next level nodes.

forward is initialized as **False** indicating we start from the left side.

Main Loop:

```
while queue:
```

This loop continues until the queue is empty, meaning all nodes have been traversed.

Processing Nodes:

```
for node in queue:
```

```
    out.append(node.data)
    if not forward:
        if node.left is not None and node.left.data != "N":
            next_queue.append(node.left)
        if node.right is not None and node.right.data != "N":
            next_queue.append(node.right)
    else:
        if node.right is not None and node.right.data != "N":
            next_queue.append(node.right)
        if node.left is not None and node.left.data != "N":
            next_queue.append(node.left)
```

For each node in the current level (queue), it appends the node's data to the out list.

If **forward** is **False**, it means we are traversing left to right. So, it appends left child first if it exists and then the right child.

If **forward** is **True**, it means we are traversing right to left. So, it appends right child first if it exists and then the left child.

Switching Direction:

```
next_queue.reverse()
queue = next_queue.copy()
next_queue = []
forward = not forward
```

After processing all nodes in the current level, it reverses next_queue to switch the direction of traversal.

Copies the content of **next_queue** to **queue** for the next iteration.

Clears **next_queue** for the next level traversal.

Switches the direction by toggling the value of **forward**.

Return:

```
return out
```

Returns the zigzag traversal stored in the out list.

This code performs a zigzag traversal of a binary tree, visiting nodes alternately from left to right and right to left at each level, and collects the node values in the traversal order.

Consider the following binary tree:

```

  1
 / \
2   3
/\  /\
4 5 6 7

```

We'll traverse this tree in a zigzag manner using the provided code.

Initialization:

out, queue, next_queue, and forward are initialized.
queue starts with the root node 1.

First Iteration (Level 1):

Current queue: [1]
Process each node in queue:
Append the value of node 1 to out.
Add children of node 1 to next_queue (left to right).

Switch Direction:

Since we processed the nodes from left to right, now we'll reverse the direction to right to left.

Second Iteration (Level 2):

Current queue: [3, 2]
Process each node in queue:
Append the value of node 3 to out.
Add children of node 3 to next_queue (right to left).
Append the value of node 2 to out.
Add children of node 2 to next_queue (right to left).

Switch Direction:

Now, we'll switch the direction again since we processed nodes from right to left.

Third Iteration (Level 3):

Current queue: [4, 5, 6, 7]
Process each node in queue:
Append the value of node 4 to out.
Append the value of node 5 to out.
Append the value of node 6 to out.
Append the value of node 7 to out.

Switch Direction:

Again, we switch the direction for the next level traversal.

Fourth Iteration (End):

No more nodes to process. The traversal is complete.
Finally, the out list will contain the zigzag traversal order [1, 3, 2, 4, 5, 6, 7].