Binary Tree Zigzag Level Order Traversal

Given a binary tree, return the *zigzag level order* traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:
Given binary tree `{3,9,20,#,#,15,7}`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
   1
  / \
 2   3
    /
   4
    \
     5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

Solution 1

```
public class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) 
    {
        List<List<Integer>> sol = new ArrayList<>();
        travel(root, sol, 0);
        return sol;
    }
    
    private void travel(TreeNode curr, List<List<Integer>> sol, int level)
    {
        if(curr == null) return;
        
        if(sol.size() <= level)
        {
            List<Integer> newLevel = new LinkedList<>();
            sol.add(newLevel);
        }
        
        List<Integer> collection  = sol.get(level);
        if(level % 2 == 0) collection.add(curr.val);
        else collection.add(0, curr.val);
        
        travel(curr.left, sol, level + 1);
        travel(curr.right, sol, level + 1);
    }
}
```

1. O(n) solution by using LinkedList along with ArrayList. So insertion in the inner list and outer list are both O(1),
2. Using DFS and creating new lists when needed.

should be quite straightforward. any better answer?

written by wayne.s.lu original link here

Solution 2

Assuming after traversing the 1st level, nodes in queue are {9, 20, 8}, And we are going to traverse 2nd level, which is even line and should print value from right to left [8, 20, 9].

We know there are 3 nodes in current queue, so the vector for this level in final result should be of size 3. Then, queue [i] -> goes to -> vector[queue.size() - 1 - i] i.e. the ith node in current queue should be placed in (queue.size() - 1 - i) position in vector for that line.

For example, for node(9), it's index in queue is 0, so its index in vector should be (3-1-0) = 2.

```
vector<vector<int> > zigzagLevelOrder(TreeNode* root) {
    if (root == NULL) {
        return vector<vector<int> > ();
    }
    vector<vector<int> > result;

    queue<TreeNode*> nodesQueue;
    nodesQueue.push(root);
    bool leftToRight = true;

    while ( !nodesQueue.empty()) {
        int size = nodesQueue.size();
        vector<int> row(size);
        for (int i = 0; i < size; i++) {
            TreeNode* node = nodesQueue.front();
            nodesQueue.pop();

            // find position to fill node's value
            int index = (leftToRight) ? i : (size - 1 - i);

            row[index] = node->val;
            if (node->left) {
                nodesQueue.push(node->left);
            }
            if (node->right) {
                nodesQueue.push(node->right);
            }
        }
        // after this level
        leftToRight = !leftToRight;
        result.push_back(row);
    }
    return result;
}
```

written by StevenCooks original link here

Solution 3

public class Solution { public List<List> zigzagLevelOrder(TreeNode root) { List<List> res = new ArrayList<>(); if(root == null) return res;

```
    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);
    boolean order = true;
    int size = 1;

    while(!q.isEmpty()) {
        List<Integer> tmp = new ArrayList<>();
        for(int i = 0; i < size; ++i) {
            TreeNode n = q.poll();
            if(order) {
                tmp.add(n.val);
            } else {
                tmp.add(0, n.val);
            }
            if(n.left != null) q.add(n.left);
            if(n.right != null) q.add(n.right);
        }
        res.add(tmp);
        size = q.size();
        order = order ? false : true;
    }
    return res;
}
```

}

written by GraceLuLi original link here

From Leetcoder.