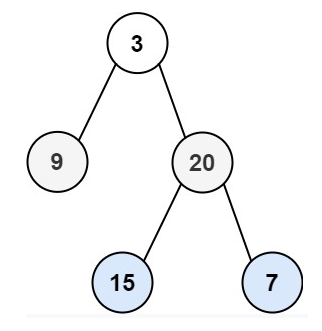
<https://leetcode.com/problems/binary-tree-level-order-traversal-ii/>

Given the root of a binary tree, return *the bottom-up level order traversal of its nodes' values*. (i.e., from left to right, level by level from leaf to root).

**Example 1:**



Input: root = [3,9,20,null,null,15,7]

Output: [[15,7],[9,20],[3]]

**Example 2:**

Input: root = [1]

Output: [[1]]

**Example 3:**

Input: root = []

Output: []

**Constraints:**

* The number of nodes in the tree is in the range [0, 2000].
* -1000 <= Node.val <= 1000

**Attempt 1: 2022-11-02**

**Solution 1: Queue with queue size calculation(10 min, just a mirror to L102/P8.1.Binary Tree Level Order Traversal, mirror at both result contained ArrayList and ArrayList contained node value)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<List<Integer>> levelOrderBottom(TreeNode root) {

List<List<Integer>> result = new ArrayList<List<Integer>>();

if(root == null) {

return result;

}

Queue<TreeNode> q = new LinkedList<TreeNode>();

q.offer(root);

while(!q.isEmpty()) {

List<Integer> list = new ArrayList<Integer>();

int size = q.size();

for(int i = 0; i < size; i++) {

TreeNode node = q.poll();

list.add(0, node.val);

if(node.right != null) {

q.offer(node.right);

}

if(node.left != null) {

q.offer(node.left);

}

}

result.add(0, list);

}

return result;

}

}

Time Complexity: O(N), where N is number of nodes in the Binary Tree

Space Complexity: O(N)

**Solution 2: Recursive traversal (10 min, just a mirror to L102/P8.1.Binary Tree Level Order Traversal, mirror at both result contained ArrayList and ArrayList contained node value)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<List<Integer>> levelOrderBottom(TreeNode root) {

List<List<Integer>> result = new ArrayList<List<Integer>>();

helper(root, result, 0);

return result;

}

private void helper(TreeNode root, List<List<Integer>> result, int depth) {

if(root == null) {

return;

}

if(result.size() == depth) {

result.add(0, new ArrayList<Integer>());

}

result.get(result.size() - 1 - depth).add(0, root.val);

helper(root.right, result, depth + 1);

helper(root.left, result, depth + 1);

}

}

Time Complexity: O(N), where N is number of nodes in the Binary Tree

Space Complexity: O(N)

**Step by step how mirror to L102/P8.1.Binary Tree Level Order Traversal works**

e.g

3

/ \

9 20

/ \

15 7

result={{}} -> right path from root to leaf first -> add new ArrayList at beginning -> result={{}} -> add 3 at beginning on index=1-1-0=0 ArrayList -> result={{3}} -> add new ArrayList at beginning -> result={{},{3}} ->

add 20 at beginning on index=2-1-1=0 ArrayList -> result={{20},{3}} -> add new ArrayList at beginning -> result={{},{20},{3}} -> add 7 at beginning on index=3-1-2=0 ArrayList -> result={{7},{20},{3}} -> add 15 at beginning on index=3-1-2=0 ArrayList -> result={{15,7},{20},{3}} -> add 9 at beginning on index=3-1-1=1 ArrayList -> result={{15,7},{9,20},{3}}