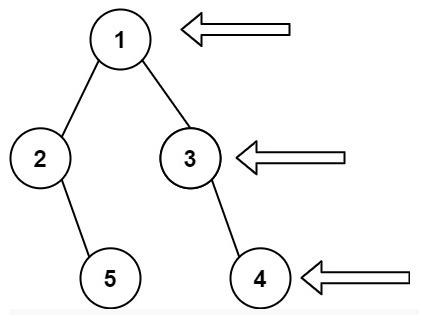
<https://leetcode.com/problems/binary-tree-right-side-view/>

Given the root of a binary tree, imagine yourself standing on the **right side** of it, return *the values of the nodes you can see ordered from top to bottom*.

**Example 1:**



Input: root = [1,2,3,null,5,null,4]

Output: [1,3,4]

**Example 2:**

Input: root = [1,null,3]

Output: [1,3]

**Example 3:**

Input: root = []

Output: []

**Constraints:**

* The number of nodes in the tree is in the range [0, 100].
* -100 <= Node.val <= 100

**Attempt 1: 2022-12-04**

**Solution 1:  Recursive traversal as DFS (10 min)**

/\*\*

\* 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<Integer> rightSideView(TreeNode root) {

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

helper(result, root, 0);

return result;

}

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

if(root == null) {

return;

}

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

result.add(root.val);

}

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

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

}

}

Time Complexity : O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/binary-tree-right-side-view/solutions/56012/my-simple-accepted-solution-java/comments/57608>

Add 3 points:

(1) the traverse of the tree is NOT standard pre-order traverse. It checks the RIGHT node first and then the LEFT

(2) the line to check currDepth == result.size() makes sure the first element of that level will be added to the result list

(3) if reverse the visit order, that is first LEFT and then RIGHT, it will return the left view of the tree.

**Solution 2:  Level order traversal as BFS (10 min)**

**Style 1: From right -> left**

/\*\*

\* 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<Integer> rightSideView(TreeNode root) {

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

if(root == null) {

return result;

}

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

q.offer(root);

while(!q.isEmpty()) {

// Level order traversal

int size = q.size();

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

TreeNode node = q.poll();

// right -> left

if(i == 0) {

result.add(node.val);

}

// First add right node of next level to queue, then left

if(node.right != null) {

q.offer(node.right);

}

if(node.left != null) {

q.offer(node.left);

}

}

}

return result;

}

}

Time Complexity : O(N)

Space Complexity: O(N)

**Style 2: From left -> right**

/\*\*

\* 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<Integer> rightSideView(TreeNode root) {

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

if(root == null) {

return result;

}

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

q.offer(root);

while(!q.isEmpty()) {

// Level order traversal

int size = q.size();

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

TreeNode node = q.poll();

// left -> right

if(i == size - 1) {

result.add(node.val);

}

// First add left node of next level to queue, then right

if(node.left != null) {

q.offer(node.left);

}

if(node.right != null) {

q.offer(node.right);

}

}

}

return result;

}

}

Time Complexity : O(N)

Space Complexity: O(N)

**Refer to**

<https://leetcode.com/problems/binary-tree-right-side-view/solutions/56012/my-simple-accepted-solution-java/comments/239974>

But I think BFS way is much more intuitive. Do the level order traversal, and add the last node on every layer.

class Solution {

public List<Integer> rightSideView(TreeNode root) {

if (root == null)

return new ArrayList();

Queue<TreeNode> queue = new LinkedList();

queue.offer(root);

List<Integer> res = new ArrayList();

while(!queue.isEmpty()){

int size = queue.size();

while (size -- > 0){

TreeNode cur = queue.poll();

if (size == 0)

res.add(cur.val);

if (cur.left != null)

queue.offer(cur.left);

if (cur.right != null)

queue.offer(cur.right);

}

}

return res;

}

}