public List<String> binaryTreePaths(TreeNode root) {

List<String> res = new ArrayList<>();

String path = “”;

dfs(root, path, res);

return res;

}

private void dfs (TreeNode root, String path, List<String> res) {

String newPath = “”;

if (path.length() == 0 ) {

newPath += root.val;

} else {

newPath = path + “->”+ root.val;

}

if (root.left == null && root.right == null) {

res.add(newPath);

} else {

if (root.left != null) {

dfs(root.left, newPath, res);

}

if (root.right != null) {

dfs(root.right, newPath, res);

}

}

}

int max = 0;

public int diameterOfBinaryTree(TreeNode root) {

depth(root);

return max;

}

private int depth(TreeNode root) {

if (root == null) {

return 0;

}

int left = depth(root.left);

int right = depth(root.right);

max = Math.max(max, (left + right));

return Math.max(left, right) + 1;

}

public List<Double> averageOfLevels(TreeNode root) {

ArrayList<Double> res = new ArrayList<>();

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

if (root == null) {

return res;

}

queue.offer(root);

while (queue.size() != 0) {

int len = queue.size();

double rowSum = 0;

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

TreeNode cur = queue.poll();

if (cur.left != null) {

queue.offer(cur.left);

}

if (cur.right != null) {

queue.offer(cur.right);

}

rowSum += cur.val;

}

res.add(rowSum/len);

}

return res;

}

public boolean leafSimilar(TreeNode root1, TreeNode root2) {

List<Integer> leafs1 = new ArrayList<>();

List<Integer> leafs2 = new ArrayList<>();

dfs(root1, leafs1);

dfs(root2, leafs2);

return leafs1.equals(leafs2);

}

private void dfs(TreeNode root List<Integer> leafs) {

if (root == null) {

return;

}

If (root.left == null. && root.right == null) {

Leafs.add(root.val);

}

dfs(root.left, leafs);

dfs(root.right, leafs);

}

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if (root == null || root == p || root == q) {

return root;

}

TreeNode left = lowestCommonAncestor(root.left, p, q);

TreeNode right = lowestCommonAncestor(root.right, p, q);

if (left != null && right != null ) {

return root;

} else {

if (left != null) {

return left;

}

if (right != null) {

return right;

}

}

return null;

}

public int[] twoSum(int[] nums, int target) {

int[] res = new int[2];

HashMap<Integer, Integer> map = new HashMap<>();

for (int I = 0, I < nums.length; i++) {

int v = target – nums[i];

if (map.containsKey(v)) {

res[0] = map.get(v);

res[1] = i;

}

map.put(nums[i], i);

}

return res;

}

public boolean isPalindrome(String s) {

s = s.toLowerCase();

int len = s.length();

int left = 0;

int right = length() – 1;

while (left < right) {

while (left < right && !Character.isLetterOrDigit(s.charAt(left))) {

left++;

}

while (left < right && !Character.isLetterOrDigit(s.charAt(right))) {

right--;

}

if (s.charAt(left) != s.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}