**Assignment Binary tree**

Q1 Given the root of a binary tree, return the spiral level order traversal of its nodes' values. The solution should consider the binary tree nodes level by level in spiral order, i.e., all nodes present at level 1 should be processed first from left to right, followed by nodes of level 2 from right to left, followed by nodes of level 3 from left to right and so on... In other words, odd levels should be processed from left to right, and even levels should be processed from right to left.

Tree is:1🡪2🡪3🡪5🡪7🡪8🡪6🡪4

import java.util.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

public class SpiralLevelOrderTraversal {

public static List<List<Integer>> spiralLevelOrder(TreeNode root) {

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

if (root == null) {

return result;

}

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

queue.offer(root);

boolean leftToRight = true;

while (!queue.isEmpty()) {

int levelSize = queue.size();

LinkedList<Integer> levelNodes = new LinkedList<>();

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

TreeNode node = queue.poll();

// Add nodes based on the current direction

if (leftToRight) {

levelNodes.addLast(node.val);

} else {

levelNodes.addFirst(node.val);

}

if (node.left != null) {

queue.offer(node.left);

}

if (node.right != null) {

queue.offer(node.right);

}

}

result.add(new ArrayList<>(levelNodes));

leftToRight = !leftToRight; // Toggle direction

}

return result;

}

public static void main(String[] args) {

// Create the binary tree

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(5);

root.left.right = new TreeNode(7);

root.right.left = new TreeNode(8);

root.right.right = new TreeNode(6);

root.right.left.left = new TreeNode(4);

List<List<Integer>> result = spiralLevelOrder(root);

// Print the result

System.out.println("Spiral Level Order Traversal:");

for (List<Integer> level : result) {

System.out.println(level);

}

}

}

Q2. Given the root of a binary tree, check if it is a complete binary tree or not. A complete binary tree is a binary tree in which every level, except possibly the last, is filled, and all nodes are as far left as possible.

Trees is:1🡪2🡪3🡪4🡪5🡪6

Output is: Complete binary tree

import java.util.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

public class CompleteBinaryTreeCheck {

// Function to check if the binary tree is a complete binary tree

public static boolean isCompleteBinaryTree(TreeNode root) {

if (root == null) {

return true;

}

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

queue.offer(root);

boolean end = false;

while (!queue.isEmpty()) {

TreeNode node = queue.poll();

if (node == null) {

end = true; // If we see a null node, all subsequent nodes must be null

} else {

if (end) {

return false; // If we have seen a null node, but still see non-null nodes, the tree is not complete

}

queue.offer(node.left);

queue.offer(node.right);

}

}

return true;

}

public static void main(String[] args) {

// Create the binary tree

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

root.right.left = new TreeNode(6);

// Check if the tree is a complete binary tree

boolean isComplete = isCompleteBinaryTree(root);

System.out.println("The tree is " + (isComplete ? "a complete binary tree" : "not a complete binary tree"));

}

}

Q3. Given the root of a binary tree, return the reverse level order traversal of its nodes' values. The solution should consider the binary tree nodes level by level in bottom-up order from left to right, i.e., process all nodes of the last level first, followed by all nodes of the second last level, and so on.

Tree:1🡪2🡪3🡪5🡪7🡪8🡪6🡪4

import java.util.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

public class ReverseLevelOrderTraversal {

// Function to get the reverse level order traversal of a binary tree

public static List<List<Integer>> reverseLevelOrder(TreeNode root) {

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

if (root == null) {

return result;

}

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

Stack<List<Integer>> stack = new Stack<>();

queue.offer(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

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

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

TreeNode node = queue.poll();

levelNodes.add(node.val);

if (node.left != null) {

queue.offer(node.left);

}

if (node.right != null) {

queue.offer(node.right);

}

}

stack.push(levelNodes);

}

// Transfer the nodes from the stack to the result list

while (!stack.isEmpty()) {

result.add(stack.pop());

}

return result;

}

// Function to print the list of lists

public static void printList(List<List<Integer>> list) {

for (List<Integer> level : list) {

System.out.println(level);

}

}

public static void main(String[] args) {

// Create the binary tree

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(5);

root.left.right = new TreeNode(7);

root.right.left = new TreeNode(8);

root.right.right = new TreeNode(6);

root.right.left.left = new TreeNode(4);

// Get the reverse level order traversal

List<List<Integer>> result = reverseLevelOrder(root);

System.out.println("Reverse Level Order Traversal:");

printList(result); // Output: [4], [5, 7, 8, 6], [2, 3], [1]

}

}

Q4. Given the root of a binary tree, return the left view of its nodes' values. Assume the left and right child of a node makes a 45–degree angle with the parent.

Tree is:1🡪2🡪3🡪5🡪7🡪8🡪6🡪4

Output :[1,2,5,7]

import java.util.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

public class LeftViewOfBinaryTree {

// Function to get the left view of a binary tree

public static List<Integer> leftView(TreeNode root) {

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

if (root == null) {

return result;

}

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

queue.offer(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

boolean firstNode = true;

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

TreeNode node = queue.poll();

// Capture the first node of each level

if (firstNode) {

result.add(node.val);

firstNode = false;

}

// Add children to the queue

if (node.left != null) {

queue.offer(node.left);

}

if (node.right != null) {

queue.offer(node.right);

}

}

}

return result;

}

// Function to print the list

public static void printList(List<Integer> list) {

System.out.println(list);

}

public static void main(String[] args) {

// Create the binary tree

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(5);

root.left.right = new TreeNode(7);

root.right.left = new TreeNode(8);

root.right.right = new TreeNode(6);

root.right.left.left = new TreeNode(4);

// Get the left view of the binary tree

List<Integer> result = leftView(root);

System.out.println("Left View of the Binary Tree:");

printList(result); // Output: [1, 2, 5, 7]

}

}

Q5. Given the root of a binary tree, convert the binary tree into its mirror and print it’s pre order

Input: 1🡪2🡪3🡪4🡪5🡪6🡪7

import java.util.\*;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

left = null;

right = null;

}

}

public class MirrorBinaryTree {

// Function to mirror the binary tree

public static void mirrorTree(TreeNode root) {

if (root == null) {

return;

}

// Swap the left and right children

TreeNode temp = root.left;

root.left = root.right;

root.right = temp;

// Recursively mirror the left and right subtrees

mirrorTree(root.left);

mirrorTree(root.right);

}

// Function to perform pre-order traversal

public static void preOrderTraversal(TreeNode root) {

if (root == null) {

return;

}

// Print the root node value

System.out.print(root.val + " ");

// Traverse the left subtree

preOrderTraversal(root.left);

// Traverse the right subtree

preOrderTraversal(root.right);

}

public static void main(String[] args) {

// Create the binary tree

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.left.left = new TreeNode(4);

root.left.right = new TreeNode(5);

root.right.left = new TreeNode(6);

root.right.right = new TreeNode(7);

// Mirror the binary tree

mirrorTree(root);

// Print the pre-order traversal of the mirrored tree

System.out.println("Pre-order traversal of the mirrored tree:");

preOrderTraversal(root); // Output should be the pre-order traversal of the mirrored tree

}

}