Tress Contest question

1. Given a Binary Tree, find Right view of it. Right view of a Binary Tree is set of nodes visible when tree is viewed from right side.

Important Note:

Hackerrank seems to have some technical challenges to include driver code, so use the below code to generate the tree before proceeding with the coding logic.

//java

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int x) {

val = x;

left = right = null;

}

}

public class BinaryTreeFromList {

// Function to create a binary tree from a list of integers

public static TreeNode createBinaryTree(List<Integer> values) {

if (values == null || values.isEmpty()) return null;

TreeNode root = new TreeNode(values.get(0));

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

queue.offer(root);

int i = 1;

while (i < values.size()) {

TreeNode current = queue.poll();

// Add left child

if (i < values.size()) {

TreeNode leftChild = new TreeNode(values.get(i++));

current.left = leftChild;

queue.offer(leftChild);

}

// Add right child

if (i < values.size()) {

TreeNode rightChild = new TreeNode(values.get(i++));

current.right = rightChild;

queue.offer(rightChild);

}

}

return root;

}

Solution:

import java.io.\*;

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int x) {

val = x;

left = right = null;

}

}

class BinaryTreeFromList {

public static TreeNode createBinaryTree(List<Integer> values) {

if (values == null || values.isEmpty() || values.get(0) == -1) return null;

TreeNode root = new TreeNode(values.get(0));

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

queue.offer(root);

int i = 1;

while (i < values.size()) {

TreeNode current = queue.poll();

// Left child

if (i < values.size() && values.get(i) != -1) {

TreeNode left = new TreeNode(values.get(i));

current.left = left;

queue.offer(left);

}

i++;

// Right child

if (i < values.size() && values.get(i) != -1) {

TreeNode right = new TreeNode(values.get(i));

current.right = right;

queue.offer(right);

}

i++;

}

return root;

}

}

public class Solution {

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

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

if (root == null) return result;

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

queue.offer(root);

while (!queue.isEmpty()) {

int size = queue.size();

TreeNode rightmost = null;

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

TreeNode current = queue.poll();

rightmost = current;

if (current.left != null) queue.offer(current.left);

if (current.right != null) queue.offer(current.right);

}

result.add(rightmost.val);

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String line = scanner.nextLine();

String[] nodes = line.split(" ");

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

for (int i = 1; i < nodes.length; i++) {

values.add(Integer.parseInt(nodes[i]));

}

TreeNode root = BinaryTreeFromList.createBinaryTree(values);

List<Integer> rightView = rightView(root);

for (int val : rightView) {

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

}

}

}

2.Problem:

Given a binary search tree of N nodes and a number K, find the greatest number in the binary search tree that is less than or equal to K.

Important Note:

Hackerrank seems to have some technical challenges to include driver code, so use the below code to generate the tree before proceeding with the coding logic.

//java

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int x) {

val = x;

left = right = null;

}

}

public class BinaryTreeFromList {

// Function to create a binary tree from a list of integers

public static TreeNode createBinaryTree(List<Integer> values) {

if (values == null || values.isEmpty()) return null;

TreeNode root = new TreeNode(values.get(0));

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

queue.offer(root);

int i = 1;

while (i < values.size()) {

TreeNode current = queue.poll();

// Add left child

if (i < values.size()) {

TreeNode leftChild = new TreeNode(values.get(i++));

current.left = leftChild;

queue.offer(leftChild);

}

// Add right child

if (i < values.size()) {

TreeNode rightChild = new TreeNode(values.get(i++));

current.right = rightChild;

queue.offer(rightChild);

}

}

return root;

}

Solution:

import java.io.\*;

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int x) {

val = x;

left = right = null;

}

}

class BinaryTreeFromList {

// Correct BST builder

public static TreeNode insertIntoBST(TreeNode root, int val) {

if (root == null) return new TreeNode(val);

if (val < root.val) root.left = insertIntoBST(root.left, val);

else root.right = insertIntoBST(root.right, val);

return root;

}

public static TreeNode createBST(List<Integer> values) {

TreeNode root = null;

for (int val : values) {

root = insertIntoBST(root, val);

}

return root;

}

}

public class Solution {

public static int findMaxLessThanOrEqual(TreeNode root, int K) {

int result = -1;

while (root != null) {

if (root.val <= K) {

result = root.val;

root = root.right;

} else {

root = root.left;

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: space-separated integers (BST values), followed by K

String line = scanner.nextLine();

String[] parts = line.trim().split(" ");

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

for (String s : parts) {

values.add(Integer.parseInt(s));

}

int K = scanner.nextInt();

TreeNode root = BinaryTreeFromList.createBST(values);

int result = findMaxLessThanOrEqual(root, K);

System.out.println(result);

}

}