DSA PRACTICE PROBLEM

PROBLEM 1:

VALID BST:

CODE:

class Solution {

public boolean isValidBST(TreeNode root) {

if (root == null) {

return true;

}

return helper(root, Long.MIN\_VALUE, Long.MAX\_VALUE);

}

private boolean helper(TreeNode root, long left, long right) {

if (root == null) {

return true;

}

if (root.val <= left || root.val >= right) {

return false;

}

return helper(root.left, left, root.val) && helper(root.right, root.val, right);

}

}

PROBLEM 2:

BOTTOM VIEW OF THE BST

CODE:

class Pair<U, V> {

public final U first;

public final V second;

public Pair(U first, V second) {

this.first = first;

this.second = second;

}

public U getKey() {

return first;

}

public V getValue() {

return second;

}

}

class Solution

{

public ArrayList <Integer> bottomView(Node root)

{

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

if (root == null) return result;

Map<Integer, Integer> mpp = new TreeMap<>();

Queue<Pair<Node, Integer>> queue = new LinkedList<>();

queue.offer(new Pair<>(root, 0));

while (!queue.isEmpty()) {

Pair<Node, Integer> p = queue.poll();

Node node = p.getKey();

int line = p.getValue();

mpp.put(line, node.data);

if (node.left != null) {

queue.offer(new Pair<>(node.left, line - 1));

}

if (node.right != null) {

queue.offer(new Pair<>(node.right, line + 1));

}

}

for (int value : mpp.values()) {

result.add(value);

}

return result;

}

}

PROBLEM 3:

TOP VIEW OF THE BST:

CODE:

class Pair<U, V> {

public final U first;

public final V second;

public Pair(U first, V second) {

this.first = first;

this.second = second;

}

public U getKey() {

return first;

}

public V getValue() {

return second;

}

}

class Solution {

static ArrayList<Integer> topView(Node root) {

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

if (root == null) return result;

Map<Integer, Integer> mpp = new TreeMap<>();

Queue<Pair<Node, Integer>> queue = new LinkedList<>();

queue.offer(new Pair<>(root, 0));

while (!queue.isEmpty()) {

Pair<Node, Integer> p = queue.poll();

Node node = p.getKey();

int line = p.getValue();

if (!mpp.containsKey(line)) {

mpp.put(line, node.data);

}

if (node.left != null) {

queue.offer(new Pair<>(node.left, line - 1));

}

if (node.right != null) {

queue.offer(new Pair<>(node.right, line + 1));

}

}

for (int value : mpp.values()) {

result.add(value);

}

return result;

}

}

PROBLEM 4:

RIGHT VIEW OF BST:

CODE:

class Solution {

public List<Integer> rightSideView(TreeNode root) {

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

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

if(root==null) return result;

queue.offer(root);

while(!queue.isEmpty()){

int size = queue.size();

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

TreeNode current = queue.poll();

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

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

if(i==size-1) result.add(current.val);

}

}

return result;

}

}

PROBLEM 5:

LEFT VIEW:

CODE:

class Solution {

public List<Integer> leftSideView(TreeNode root) {

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

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

if(root==null) return result;

queue.offer(root);

while(!queue.isEmpty()){

int size = queue.size();

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

TreeNode current = queue.poll();

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

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

if(i==0) result.add(current.val);

}

}

return result;

}

}