**DSA-PRACTICE-10 – BINARY SEARCH TREE**

BASIC CODE STRUCTURE  
class BST {

class Node {

int key;

Node left, right;

Node(int key) {

this.key = key;

left = right = null;

}

}

private Node root;

BST() {

root = null;

}

void insert(int key) {

root = insertRec(root, key);

}

private Node insertRec(Node root, int key) {

if (root == null) {

root = new Node(key);

return root;

}

if (key < root.key) {

root.left = insertRec(root.left, key);

} else if (key > root.key) {

root.right = insertRec(root.right, key);

}

return root;

}

void inorder() {

inorderRec(root);

}

private void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

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

inorderRec(root.right);

}

}

boolean search(int key) {

return searchRec(root, key);

}

private boolean searchRec(Node root, int key) {

if (root == null) {

return false;

}

if (root.key == key) {

return true;

}

return key < root.key ? searchRec(root.left, key) : searchRec(root.right, key);

}

void delete(int key) {

root = deleteRec(root, key);

}

private Node deleteRec(Node root, int key) {

if (root == null) {

return root;

}

if (key < root.key) {

root.left = deleteRec(root.left, key);

} else if (key > root.key) {

root.right = deleteRec(root.right, key);

} else {

if (root.left == null) {

return root.right;

} else if (root.right == null) {

return root.left;

}

root.key = minValue(root.right);

root.right = deleteRec(root.right, root.key);

}

return root;

}

private int minValue(Node root) {

int min = root.key;

while (root.left != null) {

min = root.left.key;

root = root.left;

}

return min;

}

}

**VALIDATE BST**class Solution {

    public boolean isValidBST(TreeNode root) {

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

        inOrderTraversal(root, inOrderList);

        for (int i = 1; i < inOrderList.size(); i++) {

            if (inOrderList.get(i) <= inOrderList.get(i - 1)) {

                return false;

            }

        }

        return true;

    }

    private void inOrderTraversal(TreeNode node, List<Integer> inOrderList) {

        if (node == null) {

            return;

        }

        inOrderTraversal(node.left, inOrderList);

        inOrderList.add(node.val);

        inOrderTraversal(node.right, inOrderList);

    }

}

**MAKE IT BST**import java.util.ArrayList;

import java.util.Collections;

class BSTValidator {

class Node {

int key;

Node left, right;

Node(int key) {

this.key = key;

left = right = null;

}

}

Node root;

BSTValidator() {

root = null;

}

boolean isBST() {

return isBSTUtil(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE);

}

private boolean isBSTUtil(Node node, int min, int max) {

if (node == null) return true;

if (node.key <= min || node.key >= max) return false;

return isBSTUtil(node.left, min, node.key) && isBSTUtil(node.right, node.key, max);

}

void convertToBST() {

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

extractValues(root, values);

Collections.sort(values);

int[] index = {0};

buildBST(root, values, index);

}

private void extractValues(Node node, ArrayList<Integer> values) {

if (node == null) return;

extractValues(node.left, values);

values.add(node.key);

extractValues(node.right, values);

}

private void buildBST(Node node, ArrayList<Integer> values, int[] index) {

if (node == null) return;

buildBST(node.left, values, index);

node.key = values.get(index[0]++);

buildBST(node.right, values, index);

}

void insert(int key) {

root = insertRec(root, key);

}

private Node insertRec(Node root, int key) {

if (root == null) {

root = new Node(key);

return root;

}

if (key < root.key) {

root.left = insertRec(root.left, key);

} else if (key > root.key) {

root.right = insertRec(root.right, key);

}

return root;

}

void inorder() {

inorderRec(root);

}

private void inorderRec(Node node) {

if (node != null) {

inorderRec(node.left);

System.out.print(node.key + " ");

inorderRec(node.right);

}

}

}

public class Main {

public static void main(String[] args) {

BSTValidator tree = new BSTValidator();

tree.insert(10);

tree.insert(5);

tree.insert(20);

tree.root.left.right = tree.new Node(15); // Introducing a violation

System.out.println("Is the tree a BST? " + tree.isBST());

if (!tree.isBST()) {

tree.convertToBST();

System.out.println("Tree converted to BST. Inorder traversal:");

tree.inorder();

}

}

}  
**TOP VIEW**

import java.util.\*;

class Node {

int value;

Node left, right;

Node(int value) {

this.value = value;

left = right = null;

}

}

class TopViewBST {

static class Pair {

Node node;

int hd;

Pair(Node node, int hd) {

this.node = node;

this.hd = hd;

}

}

public static List<Integer> topView(Node root) {

if (root == null) return new ArrayList<>();

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

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

queue.add(new Pair(root, 0));

while (!queue.isEmpty()) {

Pair current = queue.poll();

Node node = current.node;

int hd = current.hd;

if (!topNodes.containsKey(hd)) {

topNodes.put(hd, node.value);

}

if (node.left != null) queue.add(new Pair(node.left, hd - 1));

if (node.right != null) queue.add(new Pair(node.right, hd + 1));

}

return new ArrayList<>(topNodes.values());

}

public static Node insert(Node root, int value) {

if (root == null) return new Node(value);

if (value < root.value) root.left = insert(root.left, value);

else root.right = insert(root.right, value);

return root;

}

public static void main(String[] args) {

Node root = null;

int[] values = {10, 5, 15, 3, 7, 12, 18};

for (int value : values) {

root = insert(root, value);

}

System.out.println("Top view of the BST: " + topView(root));

}

}  
**BOTTOM VIEW**

import java.util.\*;

class Node {

int value;

Node left, right;

Node(int value) {

this.value = value;

left = right = null;

}

}

class BottomViewBST {

static class Pair {

Node node;

int hd;

Pair(Node node, int hd) {

this.node = node;

this.hd = hd;

}

}

public static List<Integer> bottomView(Node root) {

if (root == null) return new ArrayList<>();

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

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

queue.add(new Pair(root, 0));

while (!queue.isEmpty()) {

Pair current = queue.poll();

Node node = current.node;

int hd = current.hd;

bottomNodes.put(hd, node.value);

if (node.left != null) queue.add(new Pair(node.left, hd - 1));

if (node.right != null) queue.add(new Pair(node.right, hd + 1));

}

return new ArrayList<>(bottomNodes.values());

}

public static Node insert(Node root, int value) {

if (root == null) return new Node(value);

if (value < root.value) root.left = insert(root.left, value);

else root.right = insert(root.right, value);

return root;

}

public static void main(String[] args) {

Node root = null;

int[] values = {10, 5, 15, 3, 7, 12, 18};

for (int value : values) {

root = insert(root, value);

}

System.out.println("Bottom view of the BST: " + bottomView(root));

}

}

**LEFT VIEW**

import java.util.\*;

class Node {

int value;

Node left, right;

Node(int value) {

this.value = value;

left = right = null;

}

}

class LeftViewBST {

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

if (root == null) return new ArrayList<>();

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

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

queue.add(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

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

Node current = queue.poll();

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

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

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

}

}

return result;

}

public static Node insert(Node root, int value) {

if (root == null) return new Node(value);

if (value < root.value) root.left = insert(root.left, value);

else root.right = insert(root.right, value);

return root;

}

public static void main(String[] args) {

Node root = null;

int[] values = {10, 5, 15, 3, 7, 12, 18};

for (int value : values) {

root = insert(root, value);

}

System.out.println("Left view of the BST: " + leftView(root));

}

}

**RIGHT VIEW**

import java.util.\*;

class Node {

int value;

Node left, right;

Node(int value) {

this.value = value;

left = right = null;

}

}

class RightViewBST {

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

if (root == null) return new ArrayList<>();

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

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

queue.add(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

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

Node current = queue.poll();

if (i == levelSize - 1) result.add(current.value);

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

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

}

}

return result;

}

public static Node insert(Node root, int value) {

if (root == null) return new Node(value);

if (value < root.value) root.left = insert(root.left, value);

else root.right = insert(root.right, value);

return root;

}

public static void main(String[] args) {

Node root = null;

int[] values = {10, 5, 15, 3, 7, 12, 18};

for (int value : values) {

root = insert(root, value);

}

System.out.println("Right view of the BST: " + rightView(root));

}

}