**25-11-2024**

**CODING PRACTICE QUESTIONS**

**BINARY SEARCH TREE**

**1.Code (java):**  
import java.util.\*;

class BinarySearchTree {

class Node {

int key;

Node left, right;

Node(int key) {

this.key = key;

left = right = null;

}

}

Node root;

BinarySearchTree() {

root = null;

}

void insert(int key) {

root = insertRec(root, key);

}

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);

}

void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

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

inorderRec(root.right);

}

}

public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

bst.insert(50);

bst.insert(30);

bst.insert(70);

bst.insert(20);

bst.insert(40);

bst.insert(60);

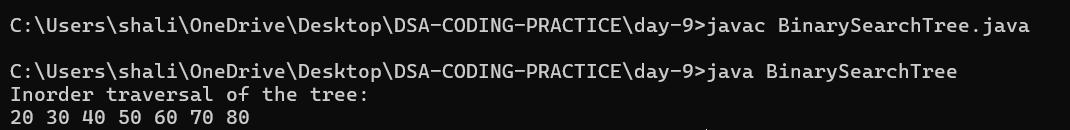
bst.insert(80);

System.out.println("Inorder traversal of the tree:");

bst.inorder();

}

}



**2.Selections in java**

* **Find kth Smallest Element**

import java.util.\*;

class BSTKthSmallest {

class Node {

int key;

Node left, right;

Node(int key) { this.key = key; }

}

Node root;

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node root, int key) {

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

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

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

return root;

}

int findKthSmallest(Node root, int[] k) {

if (root == null) return -1;

int left = findKthSmallest(root.left, k);

if (left != -1) return left;

if (--k[0] == 0) return root.key;

return findKthSmallest(root.right, k);

}

int kthSmallest(int k) {

return findKthSmallest(root, new int[]{k});

}

public static void main(String[] args) {

BSTKthSmallest bst = new BSTKthSmallest();

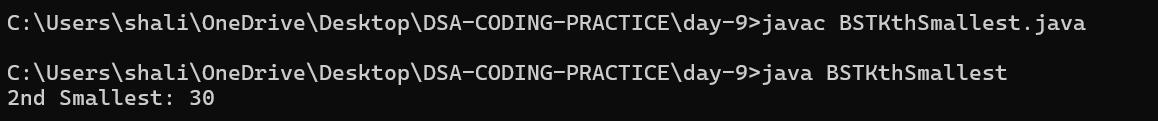
int[] keys = {50, 30, 70, 20, 40, 60, 80};

for (int key : keys) bst.insert(key);

System.out.println("2nd Smallest: " + bst.kthSmallest(2));

}

}



* **Find kth Largest Element**

import java.util.\*;

class BSTKthLargest {

class Node {

int key;

Node left, right;

Node(int key) { this.key = key; }

}

Node root;

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node root, int key) {

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

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

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

return root;

}

int findKthLargest(Node root, int[] k) {

if (root == null) return -1;

int right = findKthLargest(root.right, k);

if (right != -1) return right;

if (--k[0] == 0) return root.key;

return findKthLargest(root.left, k);

}

int kthLargest(int k) {

return findKthLargest(root, new int[]{k});

}

public static void main(String[] args) {

BSTKthLargest bst = new BSTKthLargest();

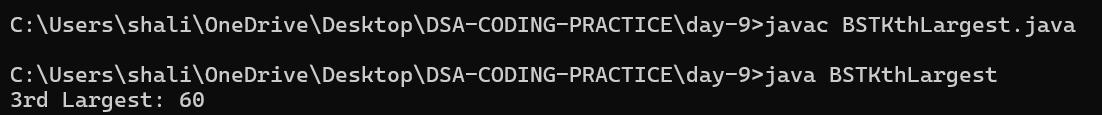
int[] keys = {50, 30, 70, 20, 40, 60, 80};

for (int key : keys) bst.insert(key);

System.out.println("3rd Largest: " + bst.kthLargest(3));

}

}



**3.Code from scratch (node class and pointers)**

import java.util.Scanner;

class BinarySearchTree2 {

class Node {

int key;

Node left, right;

Node(int key) { this.key = key; }

}

Node root;

void insert(int key) {

root = insertRec(root, key);

}

Node insertRec(Node root, int key) {

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

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

else if (key > root.key) root.right = insertRec(root.right, key);

return root;

}

void delete(int key) {

root = deleteRec(root, key);

}

Node deleteRec(Node root, int key) {

if (root == null) return null;

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;

if (root.right == null) return root.left;

root.key = findMin(root.right);

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

}

return root;

}

int findMin(Node root) {

while (root.left != null) root = root.left;

return root.key;

}

void inorder() {

inorderRec(root);

System.out.println();

}

void inorderRec(Node root) {

if (root != null) {

inorderRec(root.left);

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

inorderRec(root.right);

}

}

public static void main(String[] args) {

BinarySearchTree bst = new BinarySearchTree();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\n1. Insert");

System.out.println("2. Delete");

System.out.println("3. Display Inorder Traversal");

System.out.println("4. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter value to insert: ");

int value = scanner.nextInt();

bst.insert(value);

System.out.println(value + " inserted.");

break;

case 2:

System.out.print("Enter value to delete: ");

int delValue = scanner.nextInt();

bst.delete(delValue);

System.out.println(delValue + " deleted.");

break;

case 3:

System.out.println("Inorder Traversal:");

bst.inorder();

break;

case 4:

System.out.println("Exiting...");

scanner.close();

return;

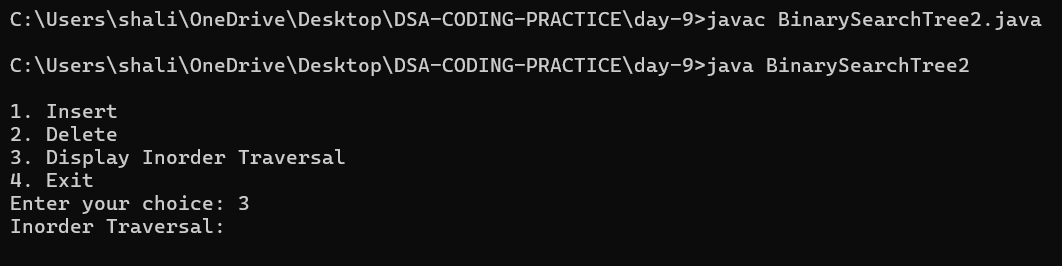
default:

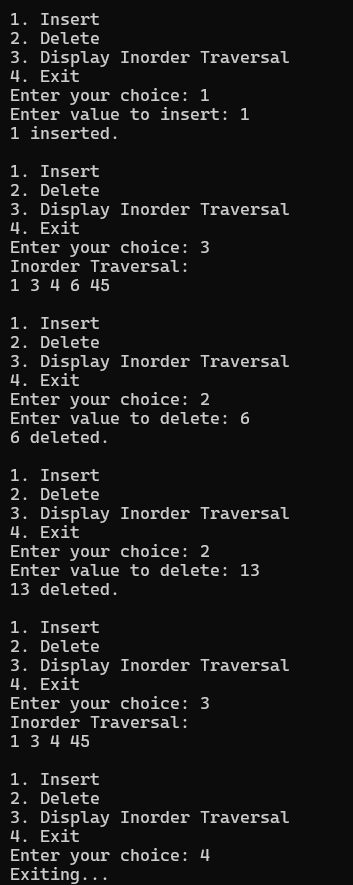
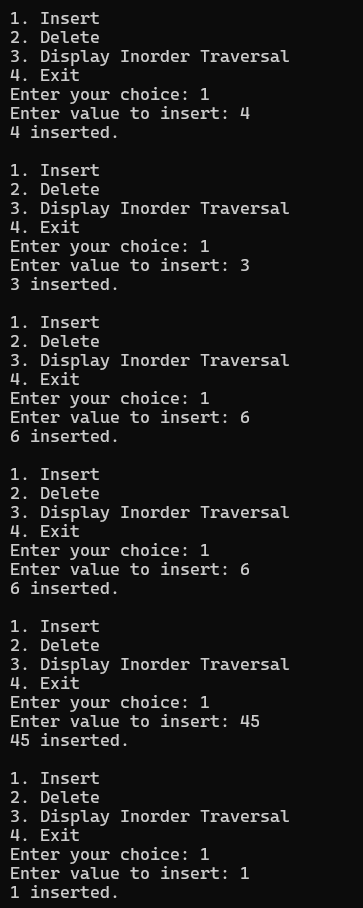
System.out.println("Invalid choice! Try again.");

}

}

}

}



**PROBLEMS**

**1.Validate if BST or not**

import java.util.Scanner;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class ValidateBST {

public boolean isValidBST(TreeNode root) {

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

}

private boolean validate(TreeNode node, long min, long max) {

if (node == null) return true;

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

return validate(node.left, min, node.val) && validate(node.right, node.val, max);

}

public static TreeNode createTree(Scanner scanner) {

System.out.print("Enter value (-1 for null): ");

int val = scanner.nextInt();

if (val == -1) return null;

TreeNode node = new TreeNode(val);

System.out.println("Enter left child of " + val);

node.left = createTree(scanner);

System.out.println("Enter right child of " + val);

node.right = createTree(scanner);

return node;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Create the binary tree:");

TreeNode root = createTree(scanner);

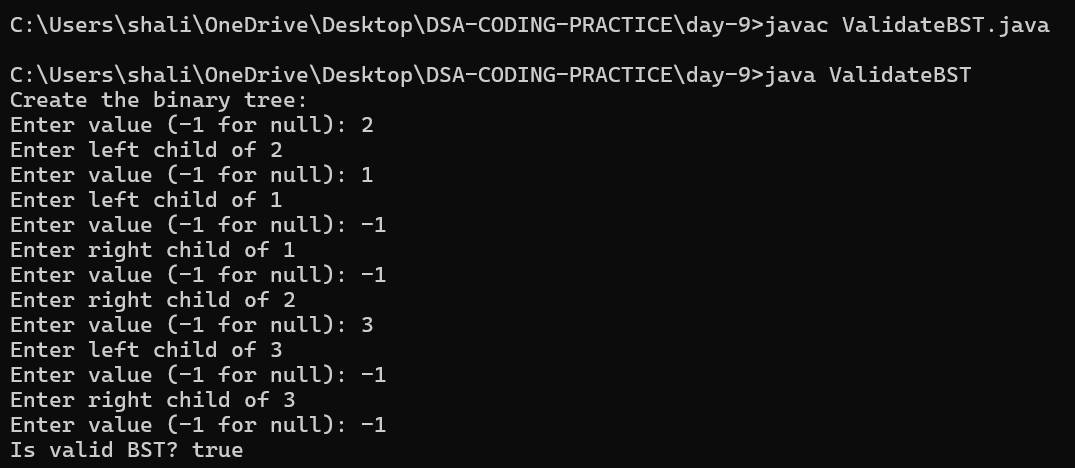
ValidateBST validator = new ValidateBST();

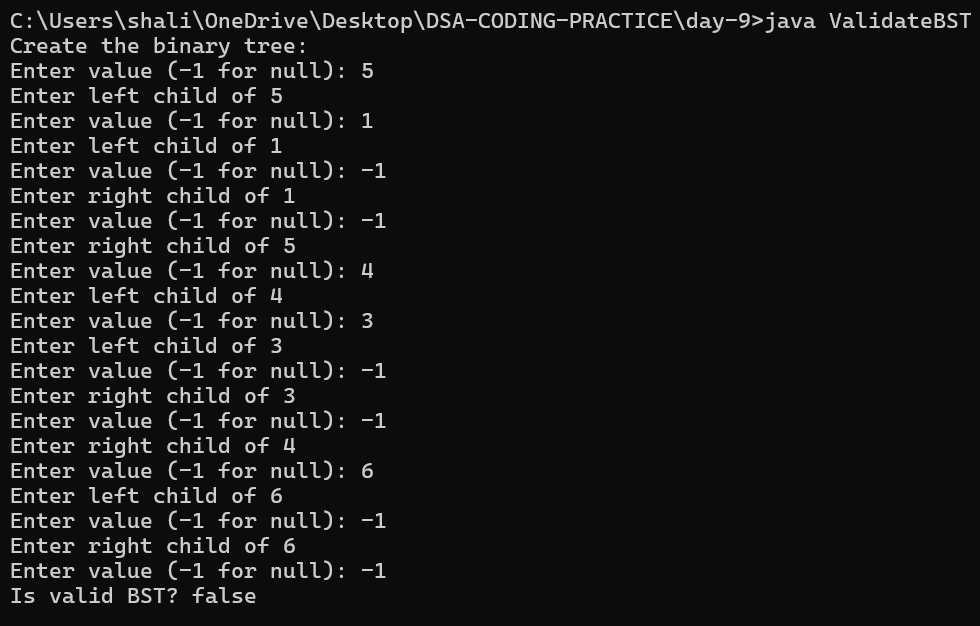
System.out.println("Is valid BST? " + validator.isValidBST(root));

scanner.close();

}

}





**2.If not a BST make it a BST**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

}

}

public class ValidateAndPrintBST {

public boolean isValidBST(TreeNode root) {

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

}

private boolean validate(TreeNode node, long min, long max) {

if (node == null) return true;

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

return validate(node.left, min, node.val) && validate(node.right, node.val, max);

}

public TreeNode convertToBST(TreeNode root) {

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

storeInorder(root, values);

Collections.sort(values);

return buildBST(values, 0, values.size() - 1);

}

private void storeInorder(TreeNode node, List<Integer> values) {

if (node == null) return;

storeInorder(node.left, values);

values.add(node.val);

storeInorder(node.right, values);

}

private TreeNode buildBST(List<Integer> values, int start, int end) {

if (start > end) return null;

int mid = start + (end - start) / 2;

TreeNode node = new TreeNode(values.get(mid));

node.left = buildBST(values, start, mid - 1);

node.right = buildBST(values, mid + 1, end);

return node;

}

public void inorderTraversal(TreeNode root) {

if (root == null) return;

inorderTraversal(root.left);

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

inorderTraversal(root.right);

}

public void printTree(TreeNode root, String prefix, boolean isLeft) {

if (root != null) {

System.out.println(prefix + (isLeft ? "├── " : "└── ") + root.val);

printTree(root.left, prefix + (isLeft ? "│ " : " "), true);

printTree(root.right, prefix + (isLeft ? "│ " : " "), false);

}

}

public static TreeNode createTree(Scanner scanner) {

System.out.print("Enter value (-1 for null): ");

int val = scanner.nextInt();

if (val == -1) return null;

TreeNode node = new TreeNode(val);

System.out.println("Enter left child of " + val);

node.left = createTree(scanner);

System.out.println("Enter right child of " + val);

node.right = createTree(scanner);

return node;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Create the binary tree:");

TreeNode root = createTree(scanner);

ValidateAndPrintBST validator = new ValidateAndPrintBST();

System.out.println("\nOriginal Tree:");

validator.printTree(root, "", false);

if (validator.isValidBST(root)) {

System.out.println("\nThe given tree is a valid BST.");

} else {

System.out.println("\nThe given tree is not a valid BST.");

root = validator.convertToBST(root);

System.out.println("\nThe tree has been converted into a valid BST.");

}

System.out.println("\nInorder traversal of the BST:");

validator.inorderTraversal(root);

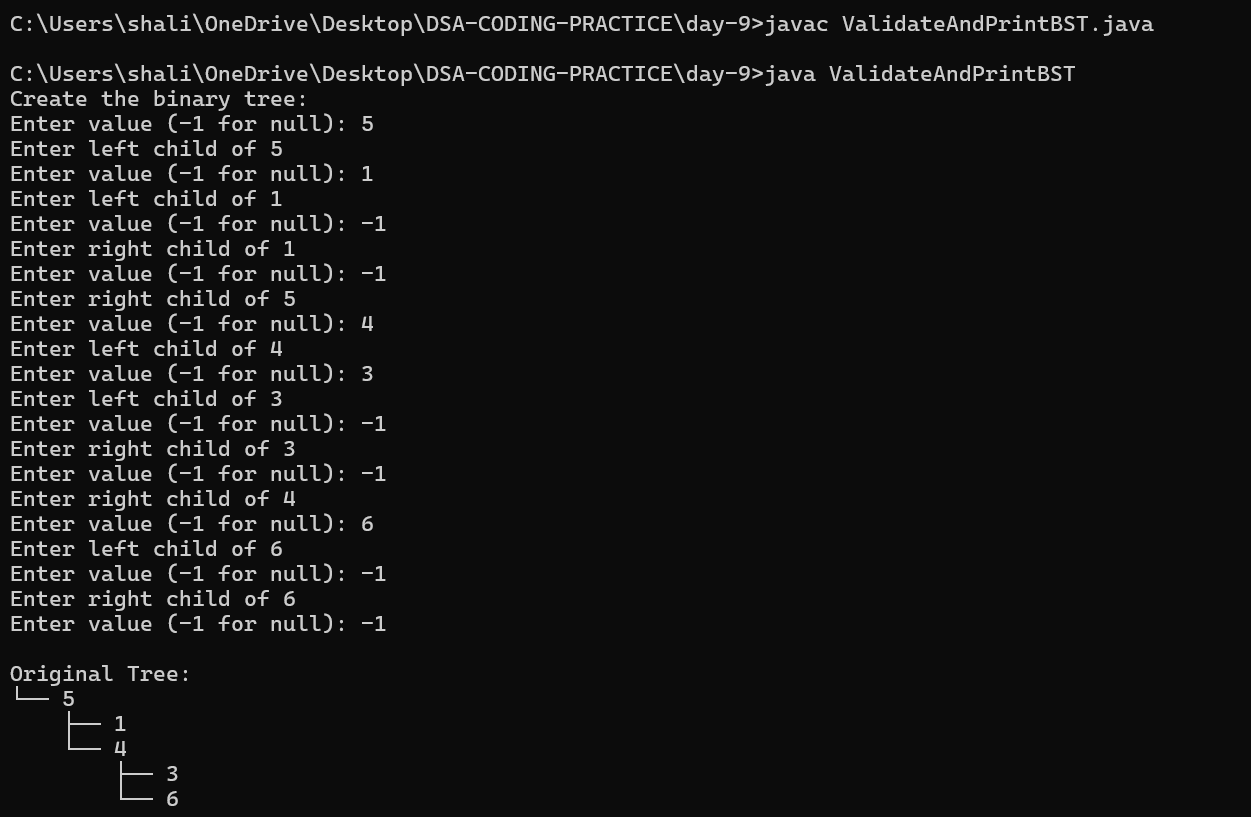
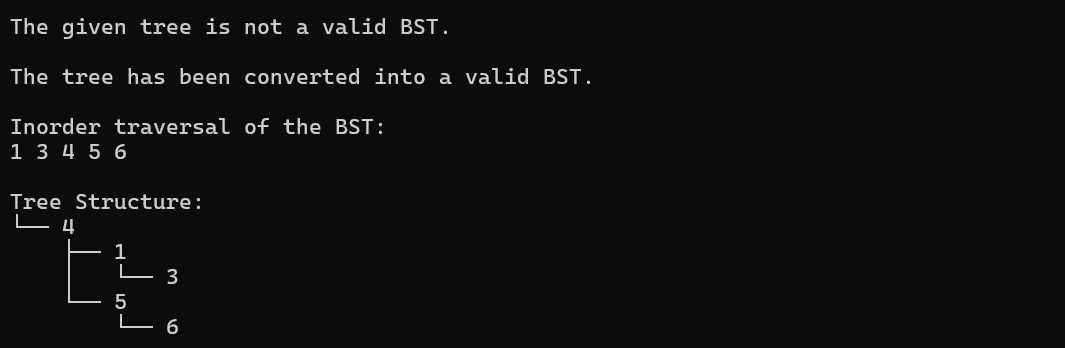
System.out.println("\n\nTree Structure:");

validator.printTree(root, "", false);

scanner.close();

}

}

**3.Top view of a BST**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

}

}

public class TopViewBST {

static class Pair {

TreeNode node;

int hd;

Pair(TreeNode node, int hd) {

this.node = node;

this.hd = hd;

}

}

public void printTopView(TreeNode root) {

if (root == null) return;

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

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

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

while (!queue.isEmpty()) {

Pair current = queue.poll();

if (!topViewMap.containsKey(current.hd)) {

topViewMap.put(current.hd, current.node.val);

}

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

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

}

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

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

}

}

public static TreeNode createTree(Scanner scanner) {

System.out.print("Enter value (-1 for null): ");

int val = scanner.nextInt();

if (val == -1) return null;

TreeNode node = new TreeNode(val);

System.out.println("Enter left child of " + val);

node.left = createTree(scanner);

System.out.println("Enter right child of " + val);

node.right = createTree(scanner);

return node;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Create the binary tree:");

TreeNode root = createTree(scanner);

TopViewBST topView = new TopViewBST();

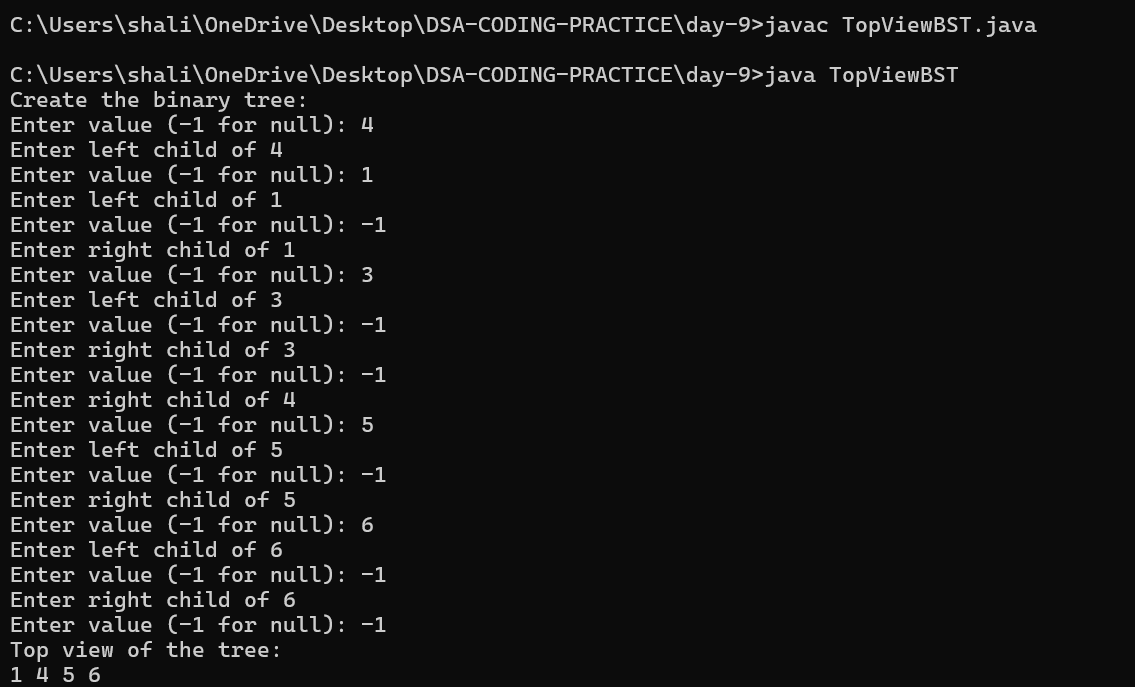
System.out.println("Top view of the tree:");

topView.printTopView(root);

scanner.close();

}

}



**4. Bottom view of a BST**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

}

}

public class BottomViewBST {

static class Pair {

TreeNode node;

int hd;

Pair(TreeNode node, int hd) {

this.node = node;

this.hd = hd;

}

}

public void printBottomView(TreeNode root) {

if (root == null) return;

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

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

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

while (!queue.isEmpty()) {

Pair current = queue.poll();

bottomViewMap.put(current.hd, current.node.val);

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

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

}

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

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

}

}

public static TreeNode createTree(Scanner scanner) {

System.out.print("Enter value (-1 for null): ");

int val = scanner.nextInt();

if (val == -1) return null;

TreeNode node = new TreeNode(val);

System.out.println("Enter left child of " + val);

node.left = createTree(scanner);

System.out.println("Enter right child of " + val);

node.right = createTree(scanner);

return node;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Create the binary tree:");

TreeNode root = createTree(scanner);

BottomViewBST bottomView = new BottomViewBST();

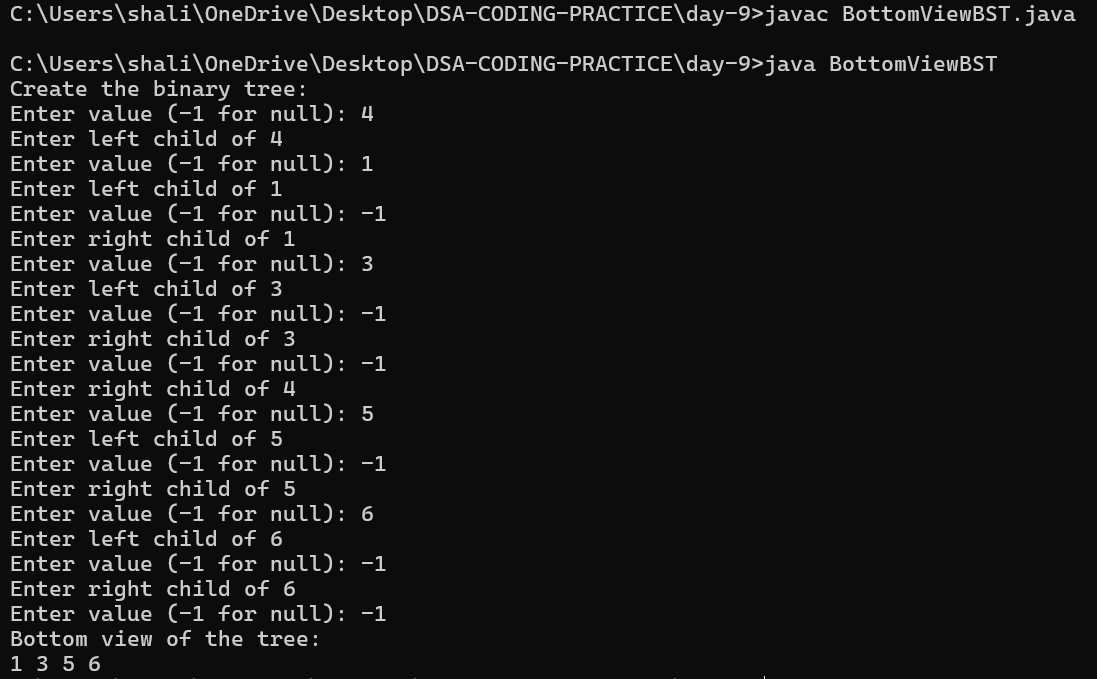
System.out.println("Bottom view of the tree:");

bottomView.printBottomView(root);

scanner.close();

}

}



**5.Left view of a BST**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

}

}

public class LeftViewBST {

public void printLeftView(TreeNode root) {

if (root == null) return;

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

queue.add(root);

while (!queue.isEmpty()) {

int size = queue.size();

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

TreeNode current = queue.poll();

if (i == 0) {

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

}

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

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

}

}

}

public static TreeNode createTree(Scanner scanner) {

System.out.print("Enter value (-1 for null): ");

int val = scanner.nextInt();

if (val == -1) return null;

TreeNode node = new TreeNode(val);

System.out.println("Enter left child of " + val);

node.left = createTree(scanner);

System.out.println("Enter right child of " + val);

node.right = createTree(scanner);

return node;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Create the binary tree:");

TreeNode root = createTree(scanner);

LeftViewBST leftView = new LeftViewBST();

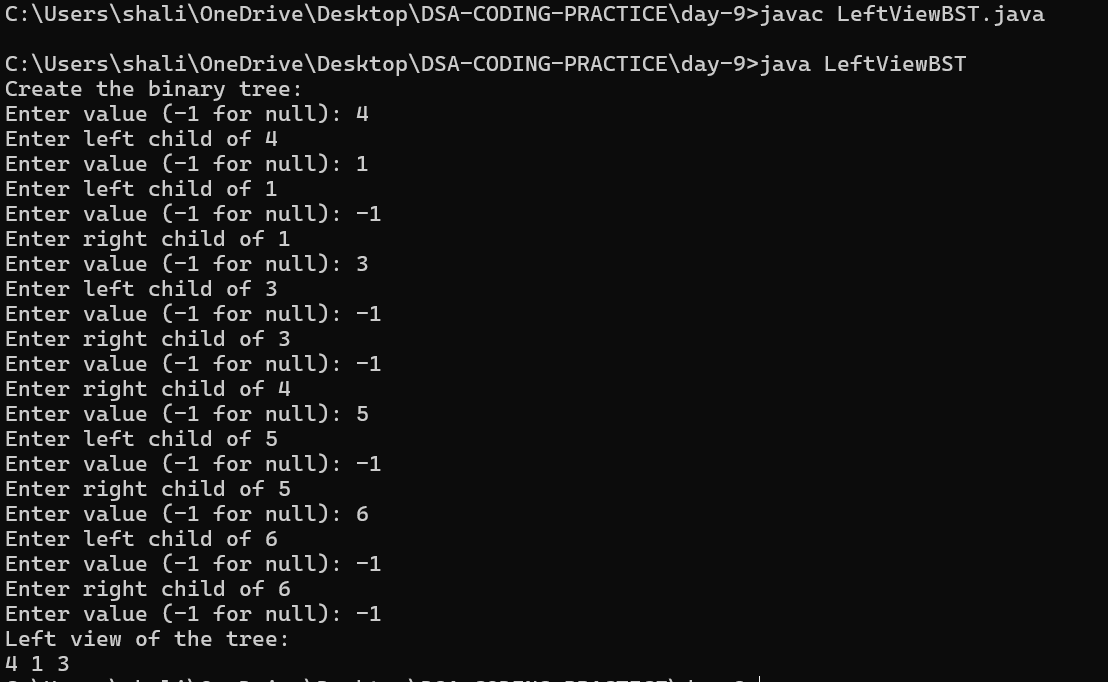
System.out.println("Left view of the tree:");

leftView.printLeftView(root);

scanner.close();

}

}



**6.Right view of a BST**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

}

}

public class RightViewBST {

public void printRightView(TreeNode root) {

if (root == null) return;

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

queue.add(root);

while (!queue.isEmpty()) {

int size = queue.size();

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

TreeNode current = queue.poll();

if (i == size - 1) {

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

}

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

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

}

}

}

public static TreeNode createTree(Scanner scanner) {

System.out.print("Enter value (-1 for null): ");

int val = scanner.nextInt();

if (val == -1) return null;

TreeNode node = new TreeNode(val);

System.out.println("Enter left child of " + val);

node.left = createTree(scanner);

System.out.println("Enter right child of " + val);

node.right = createTree(scanner);

return node;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Create the binary tree:");

TreeNode root = createTree(scanner);

RightViewBST rightView = new RightViewBST();

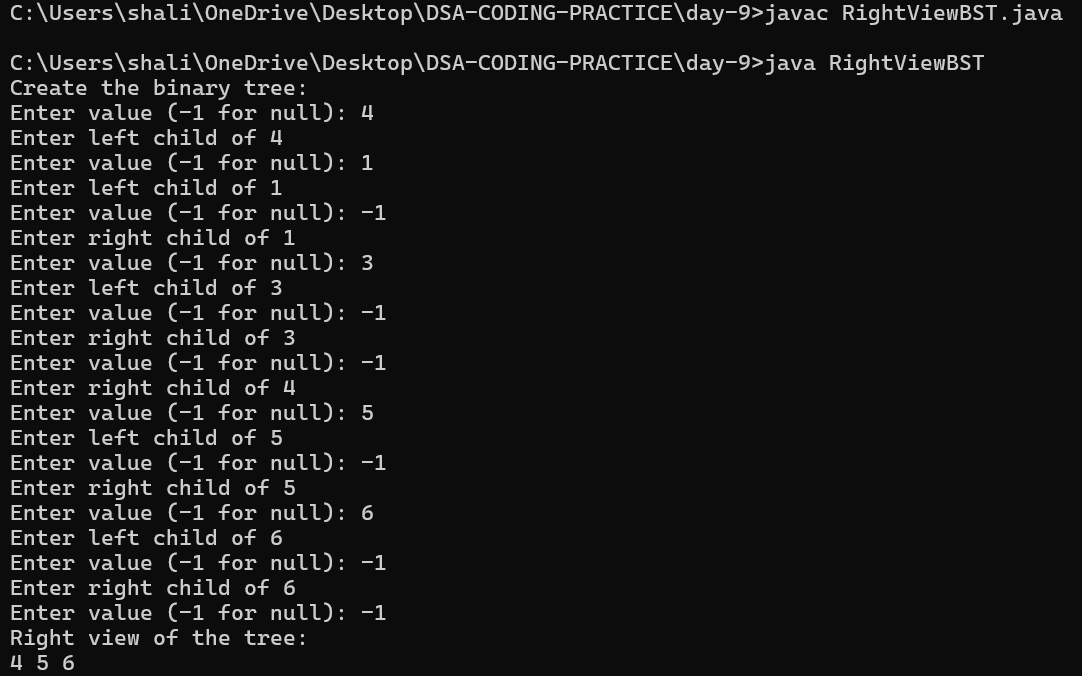
System.out.println("Right view of the tree:");

rightView.printRightView(root);

scanner.close();

}

}



**SEGMENT TREE**

A **Segment Tree** is a binary tree used for storing intervals or segments. It allows querying the sum (or other operations like min, max, etc.) of elements in a range efficiently. Segment trees are typically used for problems that involve range queries and updates.

**Scratch Code**

import java.util.Scanner;

class SegmentTree {

private int[] tree;

private int n;

public SegmentTree(int size) {

n = size;

tree = new int[4 \* n];

}

public void build(int[] arr, int node, int start, int end) {

if (start == end) {

tree[node] = arr[start];

} else {

int mid = (start + end) / 2;

build(arr, 2 \* node + 1, start, mid);

build(arr, 2 \* node + 2, mid + 1, end);

tree[node] = tree[2 \* node + 1] + tree[2 \* node + 2];

}

}

public int query(int node, int start, int end, int l, int r) {

if (r < start || end < l) {

return 0;

}

if (l <= start && end <= r) {

return tree[node];

}

int mid = (start + end) / 2;

int leftQuery = query(2 \* node + 1, start, mid, l, r);

int rightQuery = query(2 \* node + 2, mid + 1, end, l, r);

return leftQuery + rightQuery;

}

public void update(int node, int start, int end, int idx, int value) {

if (start == end) {

tree[node] = value;

} else {

int mid = (start + end) / 2;

if (start <= idx && idx <= mid) {

update(2 \* node + 1, start, mid, idx, value);

} else {

update(2 \* node + 2, mid + 1, end, idx, value);

}

tree[node] = tree[2 \* node + 1] + tree[2 \* node + 2];

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = sc.nextInt();

}

SegmentTree segmentTree = new SegmentTree(n);

segmentTree.build(arr, 0, 0, n - 1);

System.out.println("Segment Tree built successfully.");

System.out.println("Enter the range for query (l, r):");

int l = sc.nextInt();

int r = sc.nextInt();

int result = segmentTree.query(0, 0, n - 1, l, r);

System.out.println("Sum of the range [" + l + ", " + r + "] is: " + result);

System.out.println("Enter index and new value for update (idx, value):");

int idx = sc.nextInt();

int value = sc.nextInt();

segmentTree.update(0, 0, n - 1, idx, value);

System.out.println("Enter the range for query (l, r) after update:");

l = sc.nextInt();

r = sc.nextInt();

result = segmentTree.query(0, 0, n - 1, l, r);

System.out.println("Sum of the range [" + l + ", " + r + "] after update is: " + result);

sc.close();

}

}

