PRACTICE SET – 9 **25-11-2024**

DSA

1.Validate A Binary Search Tree

```
import java.util.*;

class Node {
    int value;
    Node left, right;

    Node(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}

public class Problem1 {
    private static Node buildTree(Scanner scanner) {
        System.out.println("Enter node value (or 'null' if no node):");
        String input = scanner.next();
        if (input.equals("null")) return null;

        Node node = new Node(Integer.parseInt(input));
        System.out.println("Enter left child of " + node.value + ":");
        node.left = buildTree(scanner);
        System.out.println("Enter right child of " + node.value + ":");
```

```
        node.right = buildTree(scanner);
        return node;
    }

    private static boolean isValidBST(Node node, Integer min, Integer max) {
        if (node == null) return true;
        if ((min != null && node.value <= min) || (max != null && node.value >= max)) return false;
        return isValidBST(node.left, min, node.value) && isValidBST(node.right, node.value, max);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Build the binary tree:");
        Node root = buildTree(scanner);
        boolean result = isValidBST(root, null, null);
        System.out.println(result ? "The tree is a valid BST." : "The tree is NOT a valid BST.");
        scanner.close();
    }
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem1.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem1
Build the binary tree:
Enter node value (or 'null' if no node):
2
Enter left child of 2:
Enter node value (or 'null' if no node):
null
Enter right child of 2:
Enter node value (or 'null' if no node):
null
The tree is a valid BST.

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem1.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem1
Build the binary tree:
Enter node value (or 'null' if no node):
2
Enter left child of 2:
Enter node value (or 'null' if no node):
1
Enter left child of 1:
Enter node value (or 'null' if no node):
null
Enter right child of 1:
Enter node value (or 'null' if no node):
null
Enter right child of 2:
Enter node value (or 'null' if no node):
3
Enter left child of 3:
Enter node value (or 'null' if no node):
null
Enter right child of 3:
Enter node value (or 'null' if no node):
null
The tree is a valid BST.
```

2.Convert to BST

```
import java.util.*;


class Node {
    int value;
    Node left, right;


    Node(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}
```

```
public class Problem2 {

    private static Node buildTree(Scanner scanner) {
        System.out.println("Enter node value (or 'null' if no node):");
        String input = scanner.next();
        if (input.equals("null")) return null;

        Node node = new Node(Integer.parseInt(input));
        System.out.println("Enter left child of " + node.value + ":");
        node.left = buildTree(scanner);
        System.out.println("Enter right child of " + node.value + ":");
        node.right = buildTree(scanner);
        return node;
    }

    private static void inOrderTraversal(Node node, List<Integer> values) {
        if (node == null) return;
        inOrderTraversal(node.left, values);
        values.add(node.value);
        inOrderTraversal(node.right, values);
    }

    private static Node convertToBST(List<Integer> values, int start, int end) {
        if (start > end) return null;

        int mid = (start + end) / 2;
        Node node = new Node(values.get(mid));

        node.left = convertToBST(values, start, mid - 1);
        node.right = convertToBST(values, mid + 1, end);
```

```
    return node;
}

private static Node binaryTreeToBST(Node root) {
    List<Integer> values = new ArrayList<>();
    inOrderTraversal(root, values);  // Get all node values in in-order
    Collections.sort(values);  // Sort the node values
    return convertToBST(values, 0, values.size() - 1);  // Rebuild tree as BST
}

private static void printInOrder(Node node) {
    if (node == null) return;
    printInOrder(node.left);
    System.out.print(node.value + " ");
    printInOrder(node.right);
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Build the binary tree:");
    Node root = buildTree(scanner);

    System.out.println("Original tree (in-order):");
    printInOrder(root);
    System.out.println();

    Node bstRoot = binaryTreeToBST(root);

    System.out.println("Converted Binary Search Tree (in-order):");
```

```
        printInOrder(bstRoot);
        System.out.println();

        scanner.close();
    }
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem2.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem2
Build the binary tree:
Enter node value (or 'null' if no node):
10
Enter left child of 10:
Enter node value (or 'null' if no node):
5
Enter left child of 5:
Enter node value (or 'null' if no node):
3
Enter left child of 3:
Enter node value (or 'null' if no node):
null
Enter right child of 3:
Enter node value (or 'null' if no node):
null
Enter right child of 5:
Enter node value (or 'null' if no node):
20
Enter left child of 20:
Enter node value (or 'null' if no node):
null
Enter right child of 20:
Enter node value (or 'null' if no node):
null
Enter right child of 10:
Enter node value (or 'null' if no node):
15
Enter left child of 15:
Enter node value (or 'null' if no node):
null
Enter right child of 15:
Enter node value (or 'null' if no node):
30
Enter left child of 30:
Enter node value (or 'null' if no node):
null
Enter right child of 30:
Enter node value (or 'null' if no node):
null
Original tree (in-order):
3 5 20 10 15 30
Converted Binary Search Tree (in-order):
3 5 10 15 20 30
```

3.Top View Implementation of BST

```
import java.util.*;

class Node {
    int data;
    Node left, right;

    Node(int x) {
        data = x;
        left = right = null;
```

```
    }
}

class Problem3 {

    static ArrayList<Integer> topView(Node root) {
        class Pair {
            Node node;
            int hd;

            Pair(Node node, int hd) {
                this.node = node;
                this.hd = hd;
            }
        }

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

        TreeMap<Integer, Integer> map = 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 (!map.containsKey(hd)) {
                map.put(hd, node.data);
```

```
            }

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

        for (int value : map.values()) {
            result.add(value);
        }
        return result;
    }

    public static void main(String[] args) {
        Node root = new Node(10);

        root.left = new Node(5);
        root.right = new Node(15);
        root.left.left = new Node(3);
        root.left.right = new Node(7);
        root.right.right = new Node(20);

        ArrayList<Integer> result = topView(root);
        System.out.println("Top view of the tree: " + result);
    }
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem3.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem3
Top view of the tree: [3, 5, 10, 15, 20]
```

4.Bottom View Implementation of BST

```
import java.util.*;

class Node {
    int data;
    Node left, right;

    Node(int x) {
        data = x;
        left = right = null;
    }
}

class Problem4 {

    static Node insert(Node root, int value) {
        if (root == null) {
            return new Node(value);
        }

        if (value < root.data) {
            root.left = insert(root.left, value);
        } else if (value > root.data) {
            root.right = insert(root.right, value);
        }

        return root;
    }
```

```
static ArrayList<Integer> bottomView(Node root) {
    class Pair {
        Node node;
        int hd;

        Pair(Node node, int hd) {
            this.node = node;
            this.hd = hd;
        }
    }

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

    TreeMap<Integer, Integer> map = 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;

        map.put(hd, node.data);

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

    for (int value : map.values()) {
```

```
            result.add(value);
        }
        return result;
    }

    public static void main(String[] args) {
        Node root = new Node(10);

        root = insert(root, 5);
        root = insert(root, 15);
        root = insert(root, 3);
        root = insert(root, 7);
        root = insert(root, 12);
        root = insert(root, 18);

        ArrayList<Integer> result = bottomView(root);
        System.out.println(result);
    }
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem4.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem4
[3, 5, 12, 15, 18]
```

5.Left View Implementation of BST

```
import java.util.*;

class Node {
    int data;
    Node left, right;
```

```
    Node(int x) {
        data = x;
        left = right = null;
    }
}

class Problem5 {

    static Node insert(Node root, int value) {
        if (root == null) {
            return new Node(value);
        }

        if (value < root.data) {
            root.left = insert(root.left, value);
        } else if (value > root.data) {
            root.right = insert(root.right, value);
        }

        return root;
    }

    static void leftViewUtil(Node root, int level, int[] maxLevel, ArrayList<Integer> result) {
        if (root == null) return;

        if (level > maxLevel[0]) {
            result.add(root.data);
            maxLevel[0] = level;
        }
```

```
        leftViewUtil(root.left, level + 1, maxLevel, result);
        leftViewUtil(root.right, level + 1, maxLevel, result);
    }

    static ArrayList<Integer> leftView(Node root) {
        ArrayList<Integer> result = new ArrayList<>();
        int[] maxLevel = new int[] {-1};
        leftViewUtil(root, 0, maxLevel, result);
        return result;
    }

    public static void main(String[] args) {
        Node root = new Node(10);

        root = insert(root, 5);
        root = insert(root, 15);
        root = insert(root, 3);
        root = insert(root, 7);
        root = insert(root, 12);
        root = insert(root, 18);

        ArrayList<Integer> result = leftView(root);
        System.out.println(result);
    }
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem5.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem5
[10, 5, 3]
```

6.Right View Implementation of BST

```
import java.util.ArrayList;

class Node {
    int data;
    Node left, right;

    Node(int x) {
        data = x;
        left = right = null;
    }
}

class Problem6 {

    static Node insert(Node root, int value) {
        if (root == null) {
            return new Node(value);
        }

        if (value < root.data) {
            root.left = insert(root.left, value);
        } else if (value > root.data) {
            root.right = insert(root.right, value);
        }

        return root;
    }

    static void findRightView(Node root, int level, int[] maxLevel, ArrayList<Integer> result) {
```

```
    if (root == null) return;

    if (level > maxLevel[0]) {
        result.add(root.data);
        maxLevel[0] = level;
    }

    findRightView(root.right, level + 1, maxLevel, result);
    findRightView(root.left, level + 1, maxLevel, result);
}

static ArrayList<Integer> rightView(Node root) {
    ArrayList<Integer> result = new ArrayList<>();
    int[] maxLevel = new int[] { -1 };
    findRightView(root, 0, maxLevel, result);
    return result;
}

public static void main(String[] args) {
    Node root = new Node(10);

    root = insert(root, 5);
    root = insert(root, 15);
    root = insert(root, 3);
    root = insert(root, 7);
    root = insert(root, 12);
    root = insert(root, 18);

    ArrayList<Integer> result = rightView(root);
    System.out.println(result);
```

```
    }
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem6.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem6
[10, 15, 18]
```

7.Segment Tree Implementation

```
import java.util.Scanner;

public class Problem7 {

    private int[] segmentTree;
    private int n;

    public Problem7(int[] arr) {
        n = arr.length;
        segmentTree = new int[4 * n];
        buildTree(arr, 0, 0, n - 1);
    }

    private void buildTree(int[] arr, int node, int start, int end) {
        if (start == end) {
            segmentTree[node] = arr[start];
        } else {
            int mid = (start + end) / 2;
            int leftChild = 2 * node + 1;
            int rightChild = 2 * node + 2;
```

```
        buildTree(arr, leftChild, start, mid);
        buildTree(arr, rightChild, mid + 1, end);

        segmentTree[node] = segmentTree[leftChild] + segmentTree[rightChild];
    }
}

public int query(int L, int R) {
    return query(0, 0, n - 1, L, R);
}

private 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 segmentTree[node];
    }

    int mid = (start + end) / 2;
    int leftChild = 2 * node + 1;
    int rightChild = 2 * node + 2;

    int leftSum = query(leftChild, start, mid, L, R);
    int rightSum = query(rightChild, mid + 1, end, L, R);

    return leftSum + rightSum;
}
```

```
public void update(int i, int newValue) {
    update(0, 0, n - 1, i, newValue);
}

private void update(int node, int start, int end, int i, int newValue) {
    if (start == end) {
        segmentTree[node] = newValue;
    } else {
        int mid = (start + end) / 2;
        int leftChild = 2 * node + 1;
        int rightChild = 2 * node + 2;

        if (i <= mid) {
            update(leftChild, start, mid, i, newValue);
        } else {
            update(rightChild, mid + 1, end, i, newValue);
        }

        segmentTree[node] = segmentTree[leftChild] + segmentTree[rightChild];
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the number of elements: ");
    int n = scanner.nextInt();
    int[] arr = new int[n];

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

```
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }

        Problem7 segmentTree = new Problem7(arr);

        System.out.print("Enter the range [L, R] for query: ");
        int L = scanner.nextInt();
        int R = scanner.nextInt();
        System.out.println("Sum of values in range [" + L + ", " + R + "]: " + segmentTree.query(L, R));

        System.out.print("Enter the index and new value for update: ");
        int index = scanner.nextInt();
        int newValue = scanner.nextInt();
        segmentTree.update(index, newValue);

        System.out.print("Enter the range [L, R] for query after update: ");
        L = scanner.nextInt();
        R = scanner.nextInt();
        System.out.println("Sum of values in range [" + L + ", " + R + "] after update: " +
segmentTree.query(L, R));

        scanner.close();
    }
```

```
}
```

```
C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>javac Problem7.java

C:\Users\SUNITHARAJ\Downloads\DSA-CODING-PROBLEMS\Day 9>java Problem7
Enter the number of elements: 7
Enter the elements:
3 6 5 9 2 1 7
Enter the range [L, R] for query: 0 2
Sum of values in range [0, 2]: 14
Enter the index and new value for update: 3 4
Enter the range [L, R] for query after update: 2 3
Sum of values in range [2, 3] after update: 9
```