DSA BST Practice

By Jashwanth SA (22CS066)

CSE – C

1. **Create BST:**

```
class Node{

    int key;
    Node left, right;

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

public class createBST {

    public static void main(String[] args) {

        Node root = null;

//                  50
//                /    \
//              30      70
//             /  \    /  \
//           20    40 60   80

        // 50 30 20 40 70 60 80


        root = insert(root, 50);
        root = insert(root, 30);
        root = insert(root, 70);
        root = insert(root, 20);
        root = insert(root, 40);
        root = insert(root, 60);
        root = insert(root, 80);

        System.out.print("inorder : " );
        inorder(root);
        System.out.println();

        System.out.print("preorder : " );
        preorder(root);
        System.out.println();

        System.out.print("postorder : " );
        postorder(root);
        System.out.println();
    }

    static Node insert(Node root, int key){

        // If no tree exists, create one.
        if(root == null){
            return new Node(key);
        }

        // if number already exists means the given key is a duplicate .
        if(root.key == key){
            return root;
        }

        if(key < root.key){
            root.left = insert(root.left, key);
        } else {
            root.right = insert(root.right, key);
        }

        return root;
    }

    static void inorder(Node root){

        if(root == null){
            return;
        }

        inorder(root.left);
        System.out.print(root.key + " ");
        inorder(root.right);

    }

    static void preorder(Node root){

        if(root == null)
            return;
        }

        System.out.print(root.key + " ");
        preorder(root.left);
        preorder(root.right);

    }
    static void postorder(Node root){

        if(root == null){
            return;
        }

        postorder(root.left);
        postorder(root.right);
        System.out.print(root.key + " ");


    }
```

Output:
inorder : 20 30 40 50 60 70 80
preorder : 50 30 20 40 70 60 80
postorder : 20 40 30 60 80 70 50

2. Validate BST:

```
class Node{

    int id;
    Node left, right;

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

public class validateBST {

    public static void main(String[] args) {

        Node root = new Node(50);
        root.left = new Node(30);
        root.left.left = new Node(20);
        root.left.right = new Node(40);

        root.right = new Node(70);
        root.right.left = new Node(60);
        root.right.right = new Node(80);

        System.out.println(isBst(root));
    }

    static boolean isBst(Node root){

        if(root == null) { return true; }

        if(root.left != null && MaxValueInSubTree(root.left)>=(root.id)){ return false; }

        if(root.right != null && MinValueInSubTree(root.right)<=(root.id)){ return false; }

        return isBst(root.left) && isBst(root.right);
    }

    static int MaxValueInSubTree(Node root){

        if(root == null){
            return Integer.MIN_VALUE;
        }

        return Math.max(root.id, Math.max( MaxValueInSubTree(root.left), MaxValueInSubTree(root.right) )
);
    }

    static int MinValueInSubTree(Node root){

        if(root == null){
            return Integer.MAX_VALUE;
        }

        return Math.min( root.id, Math.min( MinValueInSubTree(root.left),
MinValueInSubTree(root.right)));
    }

}
```

Output:
True

3. Left and Right view of BST:

```java
import java.util.ArrayList;
import java.util.List;

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

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

public class LeftAndRightView {

    public List<Integer> rightsideView(Node root) {
        // List to store the result
        List<Integer> res = new ArrayList<>();

        // Call the recursive function
        // to populate the right-side view
        recursionRight(root, 0, res);

        return res;
    }

    public List<Integer> leftsideView(Node root) {

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

        recursionLeft(root, 0, res);

        return res;
    }

    private void recursionLeft(Node root, int level, List<Integer> res) {
        // Check if the current node is null
        if (root == null) {
            return;
        }

        if (res.size() == level) {
            res.add(root.data);
        }

        recursionLeft(root.left, level + 1, res);

        recursionLeft(root.right, level + 1, res);
    }


    private void recursionRight(Node root, int level, List<Integer> res) {

        if (root == null) {
            return;
        }

        if (res.size() == level) {

            res.add(root.data);

            recursionRight(root.right, level + 1, res);

            recursionRight(root.left, level + 1, res);
        }
    }

    public static void main(String[] args) {

        Node root = new Node(1);
        root.left = new Node(2);
        root.left.left = new Node(4);
        root.left.right = new Node(10);
        root.left.left.right = new Node(5);
        root.left.left.right.right = new Node(6);
        root.right = new Node(3);
        root.right.right = new Node(10);
        root.right.left = new Node(9);

        LeftAndRightView solution = new LeftAndRightView();

        List<Integer> rightView = solution.rightsideView(root);

        System.out.print("Right View Traversal: ");
        for (int node : rightView) {
            System.out.print(node + " ");
        }
        System.out.println();

        List<Integer> leftView = solution.leftsideView(root);

        System.out.print("Left View Traversal: ");
        for (int node : leftView) {
            System.out.print(node + " ");
        }
        System.out.println();
    }
}
```

Output:
Right View Traversal: 1 3 10
Left View Traversal: 1 2 4 5 6

4. Top view of BST:

```java
import java.util.LinkedList;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Queue;
import java.util.TreeMap;

class Node {
    int data;
    Node left, right;

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

class TopViewBST {
    Node root;

    public TopViewBST() { root = null; }

    private void TopView(Node root)
    {
        class QueueObj {
            Node node;
            int hd;

            QueueObj(Node node, int hd)
            {
                this.node = node;
                this.hd = hd;
            }
        }
        Queue<QueueObj> q = new LinkedList<QueueObj>();
        Map<Integer, Node> topViewMap
            = new TreeMap<Integer, Node>();

        if (root == null) {
            return;
        }
        else {
            q.add(new QueueObj(root, 0));
        }

        System.out.println(
            "The top view of the tree is : ");

        while (!q.isEmpty()) {
            QueueObj tmpNode = q.poll();
            if (!topViewMap.containsKey(tmpNode.hd)) {
                topViewMap.put(tmpNode.hd, tmpNode.node);
            }

            if (tmpNode.node.left != null) {
                q.add(new QueueObj(tmpNode.node.left,
                                   tmpNode.hd - 1));
            }
            if (tmpNode.node.right != null) {
                q.add(new QueueObj(tmpNode.node.right,
                                   tmpNode.hd + 1));
            }
        }
        for (Map.Entry<Integer, Node> entry :
            topViewMap.entrySet()) {
            System.out.print(entry.getValue().data + " ");
        }
    }

    public static void main(String[] args)
    {
        /* Create following Binary Tree
        1
    / \
    2 3
    \
        4
        \
        5
            \
            6
*/
        TopViewBST tree = new TopViewBST();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.right = new Node(4);
        tree.root.left.right.right = new Node(5);
        tree.root.left.right.right.right = new Node(6);
        tree.TopView(tree.root);
    }
}
```

Output:
2 1 3 6

5. **Bottom view of BST:**

```java
import java.util.*;

class Node {
    int data;
    Node left, right;

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

class Pair {
        Node node;
        int hd;

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

class BottomViewBST {

    static ArrayList<Integer> bottomView(Node root) {

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

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

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

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

        while (!q.isEmpty()) {

            Node curr = q.peek().node;
            int hd = q.peek().hd;
            q.poll();

            hdMap.put(hd, curr.data);

            if (curr.left != null) {
                q.add(new Pair(curr.left, hd - 1));
            }

            if (curr.right != null) {
                q.add(new Pair(curr.right, hd + 1));
            }
        }

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

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

        return result;
    }

    public static void main(String[] args) {

        // Representation of the input tree:
        //         20
        //        /  \
        //      8     22
        //     / \     \
        //    5   3     25
        //       / \
        //      10 14
        Node root = new Node(20);
        root.left = new Node(8);
        root.right = new Node(22);
        root.left.left = new Node(5);
        root.left.right = new Node(3);
        root.left.right.left = new Node(10);
        root.left.right.right = new Node(14);
        root.right.right = new Node(25);

        ArrayList<Integer> result = bottomView(root);
        System.out.print("Bottom View: ");
        for (int val : result) {
          System.out.print(val + " ");
      }
    }
}
```

Output:
5 10 3 14 25