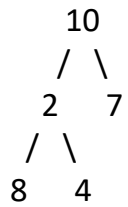
Assignment 21 Solution - Tree| DSA

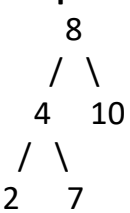
Question-1

You are given a binary tree. The binary tree is represented using the TreeNode class. Each TreeNode has an integer value and left and right children, represented using the TreeNode class itself. Convert this binary tree into a binary search tree.

Input:



Output:



Solution Code:

```
package in.ineuron.pptAssignment21;

import java.util.*;

class TreeNode {
	int val;
	TreeNode left;
	TreeNode right;

	TreeNode(int val) {
		this.val = val;
		left = null;
		right = null;
	}
}
public class BinaryTreeToBST_1 {
	// Perform an INORDER traversal of the binary tree
	private static void inorderTraversal(TreeNode root, List<Integer> list) {
		if (root == null)
			return;

		inorderTraversal(root.left, list);
		list.add(root.val);
		inorderTraversal(root.right, list);
	}
```

```
// Build a binary search tree using the sorted values
private static TreeNode buildBST(List<Integer> sortedValues, int start, int end) {
        if (start > end)
                return null;

        int mid = (start + end) / 2;
        TreeNode node = new TreeNode(sortedValues.get(mid));

        node.left = buildBST(sortedValues, start, mid - 1);
        node.right = buildBST(sortedValues, mid + 1, end);

        return node;
}

// Convert binary tree to binary search tree
public static TreeNode convertBinaryTreeToBST(TreeNode root) {
        // Step 1: Perform an INORDER traversal to obtain sorted values
        List<Integer> sortedValues = new ArrayList<>();
        inorderTraversal(root, sortedValues);

        // Step 2: Build a new binary search tree using the sorted values
        int n = sortedValues.size();
        return buildBST(sortedValues, 0, n - 1);
}

// Utility function to print the INORDER traversal of a binary tree
private static void printInorder(TreeNode root) {
        if (root == null)
                return;

        printInorder(root.left);
        System.out.print(root.val + " ");
        printInorder(root.right);
}

public static void main(String[] args) {
        // Construct the binary tree
        TreeNode root = new TreeNode(10);
        root.left = new TreeNode(2);
        root.right = new TreeNode(7);
        root.left.left = new TreeNode(8);
        root.left.right = new TreeNode(4);

        // Convert binary tree to binary search tree
        TreeNode bstRoot = convertBinaryTreeToBST(root);
```

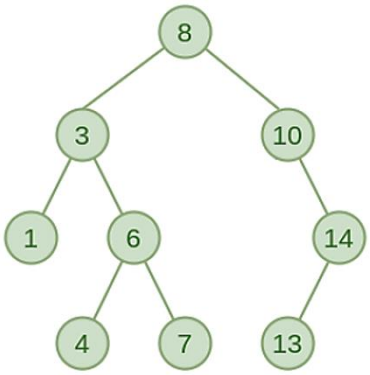
```
                // Print the INORDER traversal of the binary search tree
                System.out.print("Inorder traversal of the converted BST: ");
                printInorder(bstRoot);
        }
    }
```

Question-2:

Given a Binary Search Tree with all unique values and two keys. Find the distance between two nodes in BST. The given keys always exist in BST.

Example:

Consider the following BST:



Input-1:

n = 9
values = [8, 3, 1, 6, 4, 7, 10, 14,13]
node-1 = 6
node-2 = 14

Output-1:

The distance between the two keys = 4

Input-2:

n = 9
values = [8, 3, 1, 6, 4, 7, 10, 14,13]
node-1 = 3
node-2 = 4

Output-2:

The distance between the two keys = 2

Solution Code:

```
package in.ineuron.pptAssignment21;

class TreeNode {
	int val;
	TreeNode left;
	TreeNode right;

	TreeNode(int val) {
		this.val = val;
		left = null;
		right = null;
	}
}
```

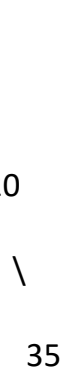
```
public class DistanceBetweenNodesInBST_2 {

    // Find the Lowest Common Ancestor (LCA) of the two nodes
    private static TreeNode findLCA(TreeNode root, int node1, int node2) {
        if (root == null)
            return null;

        if (root.val > node1 && root.val > node2)
            return findLCA(root.left, node1, node2);
        else if (root.val < node1 && root.val < node2)
            return findLCA(root.right, node1, node2);

        return root;
    }

    // Calculate the distance from the LCA to the given node
    private static int calculateDistance(TreeNode root, int node) {
        if (root.val == node)
            return 0;

        if (root.val > node)
            return 1 + calculateDistance(root.left, node);
        else
            return 1 + calculateDistance(root.right, node);
    }

    // Find the distance between two nodes in BST
    public static int findDistance(TreeNode root, int node1, int node2) {
        // Find the Lowest Common Ancestor (LCA)
        TreeNode lca = findLCA(root, node1, node2);

        // Calculate the distance from LCA to each node
        int dist1 = calculateDistance(lca, node1);
        int dist2 = calculateDistance(lca, node2);

        // Return the total distance
        return dist1 + dist2;
    }

    public static void main(String[] args) {
        // Construct the BST
        TreeNode root = new TreeNode(8);
        root.left = new TreeNode(3);
        root.right = new TreeNode(10);
        root.left.left = new TreeNode(1);
        root.left.right = new TreeNode(6);
```

```
        root.left.right.left = new TreeNode(4);
        root.left.right.right = new TreeNode(7);
        root.right.right = new TreeNode(14);
        root.right.right.left = new TreeNode(13);

        // Define the nodes for which to find the distance
        int node1 = 6;
        int node2 = 14;

        // Find the distance between the nodes
        int distance = findDistance(root, node1, node2);

        // Print the distance
        System.out.println("Distance between " + node1 + " and " + node2 + " is: " +
        distance);
    }
}
```

Question-3:

Write a program to convert a binary tree to a doubly linked list.

Input:



Output:

5 10 30 20 35

Solution Code:

```
package in.ineuron.pptAssignment21;

class TreeNode03 {
	int val;
	TreeNode03 left;
	TreeNode03 right;

	TreeNode03(int val) {
		this.val = val;
		left = null;
		right = null;
	}
}

class DoublyLinkedListNode {
	int val;
	DoublyLinkedListNode prev;
	DoublyLinkedListNode next;

	DoublyLinkedListNode(int val) {
		this.val = val;
		prev = null;
		next = null;
	}
}
```

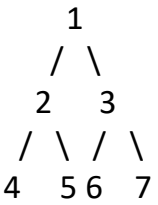
```
public class BinaryTreeToDoublyLinkedList_3 {
    // Helper function to perform INORDER traversal and convert to DLL
    private static DoublyLinkedListNode convertToDLL(TreeNode03 root) {
        if (root == null)
            return null;

        // Recursively convert left subtree
        DoublyLinkedListNode left = convertToDLL(root.left);

        // Create a new DLL node for the current root
        DoublyLinkedListNode current = new DoublyLinkedListNode(root.val);

        // If there is a left subtree, update the references
        if (left != null) {
            left.prev = current;
            current.next = left;
        }

        // Recursively convert right subtree
        DoublyLinkedListNode right = convertToDLL(root.right);

        // If there is a right subtree, update the references
        if (right != null) {
            right.prev = current;
            current.next = right;
        }

        // Return the head of the doubly linked list
        return (left != null) ? left : current;
    }

    // Utility function to print the doubly linked list
    private static void printDLL(DoublyLinkedListNode head) {
        DoublyLinkedListNode current = head;
        while (current != null) {
            System.out.print(current.val + " ");
            current = current.next;
        }
        System.out.println();
    }
```

```
    public static void main(String[] args) {
        // Construct the binary tree
        TreeNode03 root = new TreeNode03(10);
        root.left = new TreeNode03(5);
        root.right = new TreeNode03(20);
        root.right.left = new TreeNode03(30);
        root.right.right = new TreeNode03(35);

        // Convert binary tree to doubly linked list
        DoublyLinkedListNode head = convertToDLL(root);

        // Print the doubly linked list
        System.out.print("Doubly linked list: ");
        printDLL(head);
    }
}
```

Question-4:

Write a program to connect nodes at the same level.

Input:



Output:

1 → -1
2 → 3
3 → -1
4 → 5
5 → 6
6 → 7
7 → -1

Solution Code:

```
package in.ineuron.pptAssignment21;

class TreeNode04 {
	int val;
	TreeNode04 left;
	TreeNode04 right;
	TreeNode04 next; // This will be used to establish the connection

	TreeNode04(int val) {
		this.val = val;
		left = null;
		right = null;
		next = null;
	}
}

public class ConnectNodesAtSameLevel_4 {

	// Perform a level-order traversal and connect nodes at the same level
	public static void connectNodes(TreeNode04 root) {
		if (root == null)
			return;

		// Start with the root node
		TreeNode04 levelStart = root;

		while (levelStart != null) {
```

```
                TreeNode04 current = levelStart;
                TreeNode04 prev = null;
                levelStart = null; // Reset levelStart for the next level

                // Traverse the current level and establish the next pointers
                while (current != null) {
                        if (current.left != null) {
                                if (prev != null) {
                                        prev.next = current.left;
                                } else {
                                        levelStart = current.left;
                                }
                                prev = current.left;
                        }

                        if (current.right != null) {
                                if (prev != null) {
                                        prev.next = current.right;
                                } else {
                                        levelStart = current.right;
                                }
                                prev = current.right;
                        }

                        current = current.next;
                }
        }
}

// Utility function to print the connected nodes
private static void printConnectedNodes(TreeNode04 root) {
        TreeNode04 current = root;
        while (current != null) {
                System.out.print(current.val + " → ");
                current = current.next;
        }
        System.out.println("-1");
}

public static void main(String[] args) {
        // Construct the binary tree
        TreeNode04 root = new TreeNode04(1);
        root.left = new TreeNode04(2);
        root.right = new TreeNode04(3);
        root.left.left = new TreeNode04(4);
        root.left.right = new TreeNode04(5);
        root.right.left = new TreeNode04(6);
```

```
        root.right.right = new TreeNode04(7);

        // Connect nodes at the same level
        connectNodes(root);

        // Print the connections
        System.out.println("Output:");
        System.out.print("1 → ");
        printConnectedNodes(root.left);
        System.out.print("2 → ");
        printConnectedNodes(root.right.left);
        System.out.print("3 → ");
        printConnectedNodes(null); // No nodes to the right of 3
        System.out.print("4 → ");
        printConnectedNodes(root.left.right);
        System.out.print("5 → ");
        printConnectedNodes(root.right.left.right);
        System.out.print("6 → ");
        printConnectedNodes(root.right.right);
        System.out.print("7 → ");
        printConnectedNodes(null); // No nodes to the right of 7
    }
}
```


GitHub Link : https://github.com/devavratwadekar/ineuron_ppt_ProgramAssignmentCode/tree/master/PPTAssignment-21