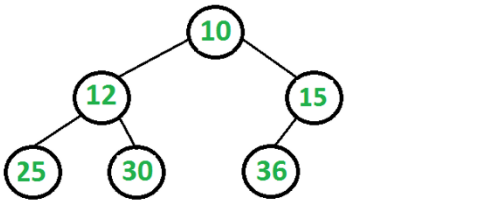
# Assignment 22 Solution - Tree| DSA

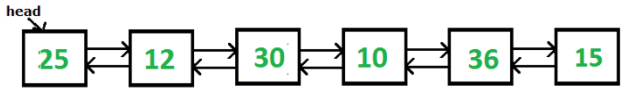
**Question-1:**

Given a Binary Tree (Bt), convert it to a Doubly Linked List(DLL). The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be the same as in Inorder for the given Binary Tree. The first node of Inorder traversal (leftmost node in BT) must be the head node of the DLL.

Example:




**The above tree should be in-place converted to following Doubly Linked List(DLL).**




**Solution Code:**

```
package in.ineuron.pptAssignment22;

class Node {
	int data;
	Node left, right;

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

class BinaryTreeToDLL_1 {
	Node root;
	Node head;
	Node prev;

	// Helper function to convert binary tree to DLL
	void binaryTreeToDLL(Node root) {
		// Base case
		if (root == null)
			return;

		// Recursively convert the left subtree
		binaryTreeToDLL(root.left);

		// Process the current node
```

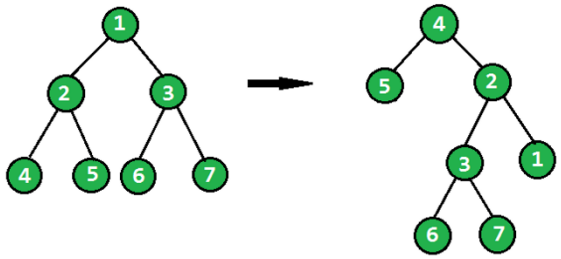
```
            if (head == null) {
                // If head is null, it means we are visiting the leftmost node in the inorder
                // traversal,
                // so make it the head of the DLL
                head = root;
            } else {
            // Otherwise, set the right pointer of the previous node to the current node
                prev.right = root;

                // Set the left pointer of the current node to the previous node
                root.left = prev;
            }

            // Update the previous node to the current node
            prev = root;

            // Recursively convert the right subtree
            binaryTreeToDLL(root.right);
        }

        // Helper function to print the DLL
        void printDLL(Node head) {
            Node curr = head;
            while (curr != null) {
                System.out.print(curr.data + " ");
                curr = curr.right;
            }
        }

        // Driver method
        public static void main(String[] args) {
            BinaryTreeToDLL_1 tree = new BinaryTreeToDLL_1();
            tree.root = new Node(10);
            tree.root.left = new Node(12);
            tree.root.right = new Node(15);
            tree.root.left.left = new Node(25);
            tree.root.left.right = new Node(30);
            tree.root.right.left = new Node(36);

            // Convert the binary tree to DLL
            tree.binaryTreeToDLL(tree.root);

            // Print the DLL
            tree.printDLL(tree.head);
        }
    }
```

**Question-2**

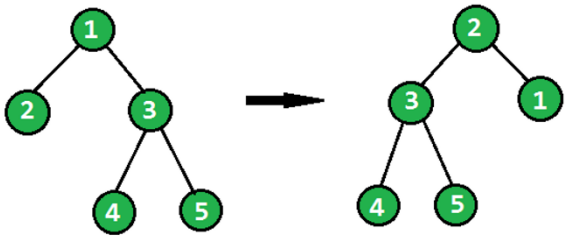
A Given a binary tree, the task is to flip the binary tree towards the right direction that is clockwise. See the below examples to see the transformation.

In the flip operation, the leftmost node becomes the root of the flipped tree and its parent becomes its right child and the right sibling becomes its left child and the same should be done for all left most nodes recursively.

**Example1:**




**Example2:**




**Solution Code:**

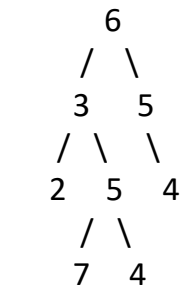
```
package in.ineuron.pptAssignment22;

class Node02 {
	int data;
	Node02 left, right;

	public Node02(int item) {
		data = item;
		left = right = null;
	}
}
class FlipBinaryTree {
	Node02 root;

	// Helper function to flip the binary tree
	Node02 flipBinaryTree(Node02 node) {
		// Base case
		if (node == null || (node.left == null && node.right == null)) {
			return node;
		}
```

```
            // Recursively flip the left and right subtrees
            Node02 flippedRoot = flipBinaryTree(node.left);

            // Set the left child of the flipped root as the right child of the original
            // node
            node.left.left = node.right;
            node.left.right = node;

            // Set the original node's left and right pointers to null
            node.left = node.right = null;

            // Return the flipped root
            return flippedRoot;
        }

        // Helper function to print the binary tree in inorder traversal
        void printInorder(Node02 node) {
            if (node == null) {
                return;
            }
            printInorder(node.left);
            System.out.print(node.data + " ");
            printInorder(node.right);
        }

        // Driver method
        public static void main(String[] args) {
            FlipBinaryTree tree = new FlipBinaryTree();
            tree.root = new Node02(1);
            tree.root.left = new Node02(2);
            tree.root.right = new Node02(3);
            tree.root.left.left = new Node02(4);
            tree.root.left.right = new Node02(5);

            // Print the original binary tree
            System.out.println("Original Binary Tree:");
            tree.printInorder(tree.root);

            // Flip the binary tree
            tree.root = tree.flipBinaryTree(tree.root);

            // Print the flipped binary tree
            System.out.println("\nFlipped Binary Tree:");
            tree.printInorder(tree.root);
        }
    }
```

**Question-3:**

Given a binary tree, print all its root-to-leaf paths without using recursion. For example, consider the following Binary Tree.

**Input:**



**Output:**

There are 4 leaves, hence 4 root to leaf paths -

6->3->2

6->3->5->7

6->3->5->4

6->5>4

**Solution Code:**

```
package in.ineuron.pptAssignment22;

import java.util.Stack;
class Node03 {
	int data;
	Node03 left, right;

	public Node03(int item) {
		data = item;
		left = right = null;
	}
}
public class BinaryTreePaths_3 {
	Node03 root;

	// Helper class to store node and its corresponding path
	class NodeWithPath {
		Node03 node;
		String path;

		public NodeWithPath(Node03 node, String path) {
			this.node = node;
			this.path = path;
		}
	}
```

```
        // Function to print all root-to-leaf paths without recursion
        void printRootToLeafPaths() {
            if (root == null)
                return;

            Stack<NodeWithPath> stack = new Stack<>();
            stack.push(new NodeWithPath(root, ""));

            while (!stack.isEmpty()) {
                NodeWithPath current = stack.pop();
                Node03 currentNode = current.node;
                String currentPath = current.path;

                if (currentNode.left == null && currentNode.right == null) {
                    // Leaf node reached, print the path
                    System.out.println(currentPath + currentNode.data);
                }

                if (currentNode.right != null) {
                    // Push the right child with updated path
                    stack.push(new NodeWithPath(currentNode.right, currentPath +
                    currentNode.data + "->"));
                }

                if (currentNode.left != null) {
                    // Push the left child with updated path
                    stack.push(new NodeWithPath(currentNode.left, currentPath +
                    currentNode.data + "->"));
                }
            }
        }
        // Driver method
        public static void main(String[] args) {
            BinaryTreePaths_3 tree = new BinaryTreePaths_3();
            tree.root = new Node03(6);
            tree.root.left = new Node03(3);
            tree.root.right = new Node03(5);
            tree.root.left.left = new Node03(2);
            tree.root.left.right = new Node03(5);
            tree.root.right.right = new Node03(4);
            tree.root.left.right.left = new Node03(7);
            tree.root.left.right.right = new Node03(4);

            // Print all root-to-leaf paths
            tree.printRootToLeafPaths();
        }
    }
```

**Question-4:**

Given Preorder, Inorder and Postorder traversals of some tree. Write a program to check if they all are of the same tree.

**Examples:**

Input :

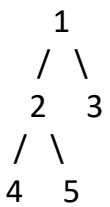
Inorder -> 4 2 5 1 3
Preorder -> 1 2 4 5 3
Postorder -> 4 5 2 3 1

Output :

Yes

Explanation :

All of the above three traversals are of the same tree



Input :

Inorder -> 4 2 5 1 3
Preorder -> 1 5 4 2 3
Postorder -> 4 1 2 3 5

Output :

No

**Solution Code:**

```
package in.ineuron.pptAssignment22;

class Node04 {
	int data;
	Node04 left, right;

	public Node04(int item) {
		data = item;
		left = right = null;
	}
}

public class TraversalCheck_4 {
	Node04 constructTree(int[] inorder, int[] preorder, int inStart, int inEnd, int preIndex) {
		if (inStart > inEnd) {
			return null;
		}

		// Create a new node with the current element from the preorder traversal
```

```
        Node04 node = new Node04(preorder[preIndex]);
        preIndex++;

        // If the current node has no children, return the node
        if (inStart == inEnd) {
            return node;
        }

        // Find the index of the current node in the inorder traversal
        int inIndex = search(inorder, inStart, inEnd, node.data);

        // Construct the left and right subtrees recursively
        node.left = constructTree(inorder, preorder, inStart, inIndex - 1, preIndex);
        node.right = constructTree(inorder, preorder, inIndex + 1, inEnd, preIndex);

        return node;
    }

    // Helper function to search for an element in the inorder traversal
    int search(int[] arr, int start, int end, int value) {
        for (int i = start; i <= end; i++) {
            if (arr[i] == value) {
                return i;
            }
        }
        return -1; // Element not found
    }

    // Function to check if the given traversals are of the same tree
    boolean checkSameTree(int[] inorder, int[] preorder, int[] postorder) {
        int n = inorder.length;
        int preIndex = 0;

        // Construct the tree using the inorder and preorder traversals
        Node04 root = constructTree(inorder, preorder, 0, n - 1, preIndex);

        // Check if the constructed tree matches the postorder traversal
        return checkPostorder(root, postorder, n, preIndex);
    }

    // Helper function to check if the constructed tree matches the postorder
    // traversal
    boolean checkPostorder(Node04 node, int[] postorder, int n, int preIndex) {
        if (node == null) {
            return true;
        }
```

```
                // Traverse the tree in postorder and compare with the postorder traversal array
                boolean leftSubtree = checkPostorder(node.left, postorder, n, preIndex);
                boolean rightSubtree = checkPostorder(node.right, postorder, n, preIndex);

                if (leftSubtree && rightSubtree && node.data == postorder[preIndex]) {
                        preIndex++;
                        return true;
                }

                return false;
        }

        // Driver method
        public static void main(String[] args) {
                TraversalCheck_4 tree = new TraversalCheck_4();
                int[] inorder = { 4, 2, 5, 1, 3 };
                int[] preorder = { 1, 2, 4, 5, 3 };
                int[] postorder = { 4, 5, 2, 3, 1 };

                boolean isSameTree = tree.checkSameTree(inorder, preorder, postorder);

                if (isSameTree) {
                        System.out.println("The traversals are of the same tree.");
                } else {
                        System.out.println("The traversals are not of the same tree.");
                }
        }
}
```



**GitHub Link :** https://github.com/devavratwadekar/ineuron_ppt_ProgramAssignmentCode/tree/master/PPTAssignment-22