

PRACTICAL NO:09

Aim : Write a Program in to Swap Nodes pairwise.

PROGRAM:

```
class Node
{
    int data;
    Node next;
    Node(int data)
    {
        this.data=data;
        next=null;
    }
}

class linkedlist
{
    Node head;
    Node tail;
    linkedlist()
    {
        head=null;
        tail=null;
    }

    void insert(int data)
    {
        Node hello=new Node(data);
        if(head==null)
        {
            head=hello;
            tail=hello;
            System.out.println(data+" is inserted");
        }
    }
}
```

```
}  
else  
{  
    tail.next=hello;  
    tail=hello;  
    System.out.println(data+" is inserted");  
}  
}  
void swap()  
{  
    Node temp=head;  
    while(temp!=null )  
    {  
        int a=temp.data;  
        temp.data=temp.next.data;  
        temp.next.data=a;  
  
        temp=temp.next.next;  
    }  
    System.out.println("Swaped");  
}  
void display()  
{  
    Node n=head;  
    while(n!= null)  
    {  
        System.out.print(n.data+"->");  
        n=n.next;  
    }  
}
```

}

```
public class MAIN
{
    public static void main(String args[])
    {
        linkedlist list=new linkedlist();
        list.insert(10);
        list.insert(20);
        list.insert(30);
        list.insert(40);
        list.insert(50);
        list.insert(60);
        list.display();
        list.swap();
        list.display();
    }
}
```

}Algorithm :

1. Check if the input head is null or has only one node. If so, return the head as there's no need to perform any swaps.
2. Initialize a dummy node and set its next pointer to the head of the list. This dummy node will simplify handling edge cases where the head needs to be swapped.
3. Initialize a prev pointer to the dummy node.
4. Iterate through the list while there are at least two nodes left to swap (head and head.next are not null):
 - Store references to the first (firstNode) and second (secondNode) nodes.
 - Update the next pointers to perform the swap: prev.next points to secondNode, firstNode.next points to secondNode.next, and secondNode.next points to firstNode.

- Move prev to firstNode.
 - Move head to firstNode.next.
5. Return the head of the modified list.

Output :

Original List: 10 20 30 40 50 60

List after pairwise swapping: 20 10 40 30 50 60

PRACTICAL NO:10

AIM: Write a Program to Understand and implement Tree traversals i.e. Pre- Order Post-Order, In-Order

ALGORITHM:

Here's the algorithm to perform different types of tree traversals (preorder, inorder, and postorder) in a binary tree:

1. Define three recursive functions, printPreorder, printInorder, and printPostorder, to perform preorder, inorder, and postorder traversals respectively.
2. In each traversal function:
 - If the current node is null, return.
 - Print the key of the current node.
 - Recursively call the traversal function for the left subtree.
 - Recursively call the traversal function for the right subtree.
3. In the main function:
 - Create an instance of the BinaryTree class.
 - Populate the binary tree with nodes as per the given example.
 - Call each traversal function (printPreorder, printInorder, and printPostorder) with the root node of the tree to perform the respective traversals.
 - Print the result of each traversal.

PROGRAM:

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

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

class BinaryTree {

    Node root;

    BinaryTree() { root = null; }
```

```
void printPreorder(Node node)
{
    if (node == null)
        return;

    System.out.print(node.key + " ");
    printPreorder(node.left);
    printPreorder(node.right);
}

void printPostorder(Node node)
{
    if (node == null)
        return;

    printPostorder(node.left);
    printPostorder(node.right);
    System.out.print(node.key + " ");
}

void printInorder(Node node)
{
    if (node == null)
        return;

    printInorder(node.left);
    System.out.print(node.key + " ");
    printInorder(node.right);
}

public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Preorder traversal of binary tree is ");
    tree.printPreorder(tree.root);
    System.out.println();
    System.out.println("Postorder traversal of binary tree is ");
    tree.printPostorder(tree.root);
}
```

```
System.out.println();  
System.out.println("Inorder traversal of binary tree is ");  
tree.printInorder(tree.root);  
}  
}
```

OUTPUT:

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.2.3\lib\idea_r  
Preorder traversal of binary tree is  
1 2 4 5 3  
Postorder traversal of binary tree is  
4 5 2 3 1  
Inorder traversal of binary tree is  
4 2 5 1 3  
Process finished with exit code 0  
|
```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:-11

AIM: Write a program to verify and validate mirrored trees or not.

ALGORITHM:

Here's an algorithm to check if a binary tree is symmetric (i.e., a mirror image of itself):

1. Define a recursive function `isMirror(node1, node2)` that takes two nodes as input and returns true if they are mirrors of each other and false otherwise.
2. In the `isMirror` function:
 - If both `node1` and `node2` are null, return true.
 - If either `node1` or `node2` is null (but not both), return false.
 - If the keys of `node1` and `node2` are equal, recursively check if the left subtree of `node1` is a mirror of the right subtree of `node2` and if the right subtree of `node1` is a mirror of the left subtree of `node2`.
 - Otherwise, return false.
3. Define a function `isSymmetric()` that calls `isMirror(root, root)` to check if the binary tree is symmetric.
4. In the main function:
 - Create an instance of the `BinaryTree` class.
 - Populate the binary tree with nodes as per the given example.
 - Call the `isSymmetric` function to determine if the tree is symmetric.
 - Print "Symmetric" if the tree is symmetric; otherwise, print "Not symmetric".

PROGRAM:

```
class Node {
    int key;
    Node left, right;
    Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree {
    Node root;
    boolean isMirror(Node node1, Node node2)
    {
        if (node1 == null && node2 == null)
            return true;
```



```

        if (node1 != null && node2 != null
            && node1.key == node2.key)
            return (isMirror(node1.left, node2.right)
                && isMirror(node1.right, node2.left));

        return false;
    }
    boolean isSymmetric()
    {
        return isMirror(root, root);
    }
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(4);
        tree.root.right.left = new Node(4);
        tree.root.right.right = new Node(3);
        boolean output = tree.isSymmetric();
        if (output == true)
            System.out.println("Symmetric");
        else
            System.out.println("Not symmetric");
    }
}

```

OUTPUT:



```

C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Ed...
Symmetric
Process finished with exit code 0

```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:-12

AIM: Write a Program to determine the depth of agiven Tree by
Implementing MAXDEPTH

ALGORITHM:

Here's the algorithm to find the maximum depth (height) of a binary tree:

1. Define a recursive function `maxDepth(node)` that takes a node as input and returns the maximum depth of the subtree rooted at that node.
2. In the `maxDepth` function:
 - If the node is null, return 0.
 - Recursively calculate the maximum depth of the left subtree (`lDepth`) and the right subtree (`rDepth`).
 - Return the maximum depth among `lDepth` and `rDepth`, incremented by 1 (to account for the current node).
3. In the main function:
 - Create an instance of the `BinaryTree` class.
 - Populate the binary tree with nodes as per the given example.
 - Call the `maxDepth` function with the root node of the tree to find the maximum depth.
 - Print the maximum depth of the tree.

PROGRAM:

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

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

class BinaryTree {
    Node root;

    int maxDepth(Node node)
    {
        if (node == null)
```

```
        return 0;
    else {

        int lDepth = maxDepth(node.left);
        int rDepth = maxDepth(node.right);


        /* use the larger one */
        if (lDepth > rDepth)
            return (lDepth + 1);
        else
            return (rDepth + 1);
    }
}

public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();

    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Height of tree is "
        + tree.maxDepth(tree.root));
}
}
```

OUTPUT:



```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Commu
Height of tree is 3

Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:13

AIM: Write a program for LowestCommon Ancestors.

ALGORITHM:

Here's the algorithm to find the Lowest Common Ancestor (LCA) of two nodes in a binary tree:

1. Define a function findLCA(n1, n2) that takes the values of two nodes as input and returns the LCA of those nodes.
2. Inside the findLCA function:
 - If the current node is null, return null.
 - If the current node's data matches either of the two node values (n1 or n2), return the current node.
 - Recursively search for the LCA in the left subtree and the right subtree.
 - If both left and right subtrees return non-null values, it means that the current node is the LCA. Return the current node.
 - If only one of the subtrees returns a non-null value, return that value.
3. In the main function:
 - Create an instance of the BinaryTree class.
 - Populate the binary tree with nodes as per the given example.
 - Call the findLCA function with pairs of node values to find their LCA.
 - Print the LCA for each pair of nodes.

PROGRAM:

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

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

public class BinaryTree {

    Node root;

    Node findLCA(int n1, int n2)
    {
        return findLCA(root, n1, n2);
    }

    Node findLCA(Node node, int n1, int n2)
    {
```

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

if (node.data == n1 || node.data == n2)
    return node;

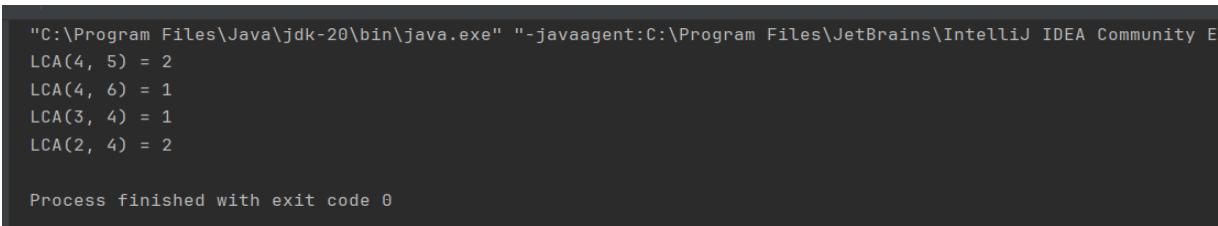
Node left_lca = findLCA(node.left, n1, n2);
Node right_lca = findLCA(node.right, n1, n2);

if (left_lca != null && right_lca != null)
    return node;

return (left_lca != null) ? left_lca : right_lca;
}
```

```
public static void main(String args[])
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);
    tree.root.right.left = new Node(6);
    tree.root.right.right = new Node(7);
    System.out.println("LCA(4, 5) = " + tree.findLCA(4, 5).data);
    System.out.println("LCA(4, 6) = " + tree.findLCA(4, 6).data);
    System.out.println("LCA(3, 4) = " + tree.findLCA(3, 4).data);
    System.out.println("LCA(2, 4) = " + tree.findLCA(2, 4).data);
}
}
```

OUTPUT:



```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community E
LCA(4, 5) = 2
LCA(4, 6) = 1
LCA(3, 4) = 1
LCA(2, 4) = 2

Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:14

AIM: Write a Program to Build BST

ALGORITHM:

Here's the algorithm for the provided Java program to build a Binary Search Tree (BST):

1. Define the Node class:

- Define a class **Node** to represent each node of the BST.
- Each node has three fields: **key** to store the value of the node, **left** to point to the left child, and **right** to point to the right child.

2. Define the BinarySearchTree class:

- Define a class **BinarySearchTree** to implement the BST.
- It contains a field **root** to store the root node of the BST.

3. Constructor:

- Define a constructor for the **BinarySearchTree** class to initialize the root node as null when a new BST object is created.

4. Insertion Method:

- Define an **insert** method to insert a new key into the BST.
- It calls a private recursive method **insertRec** passing the root node and the key to insert.

5. Recursive Insertion Method (insertRec):

- Base case: If the root is null, create a new node with the key and return it.
- Recur down the tree:
 - If the key is less than the current node's key, go to the left subtree.
 - If the key is greater than the current node's key, go to the right subtree.
- After insertion, return the (unchanged) node pointer.

6. Inorder Traversal Method:

- Define an **inorder** method to perform an inorder traversal of the BST.
- It calls a private recursive method **inorderRec** passing the root node.

7. Recursive Inorder Traversal Method (inorderRec):

- Base case: If the root is null, return.
- Recur for the left subtree.
- Print the key of the current node.
- Recur for the right subtree.

8. Main Method:

- Create a **BinarySearchTree** object.
- Insert some keys into the BST using the **insert** method.
- Print the inorder traversal of the BST using the **inorder** method.

This algorithm outlines the steps taken by the provided Java program to build and traverse a Binary Search Tree.

PROGRAM:

```
public class BinarySearchTree {
    private static class Node {
        int key;
        Node left, right;

        public Node(int item) {
            key = item;
            left = right = null;
        }
    }
    private Node root;
    public BinarySearchTree() {
        root = null;
    }
    private void insert(int key) {
        root = insertRec(root, key);
    }
    private Node insertRec(Node root, int key) {
        // If the tree is empty, return a new node
        if (root == null) {
            root = new Node(key);
            return root;
        }

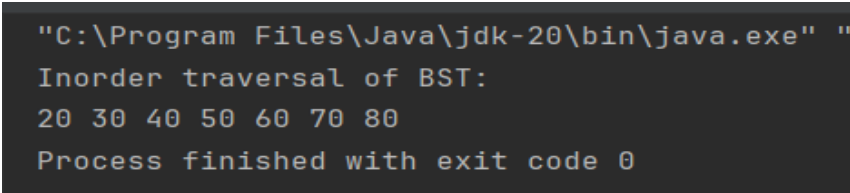
        if (key < root.key)
            root.left = insertRec(root.left, key);
        else if (key > root.key)
            root.right = insertRec(root.right, key);

        return root;
    }
    private void inorder() {
        inorderRec(root);
    }
    private 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(20);
        bst.insert(40);
    }
}
```

```
        bst.insert(70);  
        bst.insert(60);  
        bst.insert(80);  
  
        System.out.println("Inorder traversal of BST:");  
        bst.inorder();  
    }  
}
```

OUTPUT:

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "  
Inorder traversal of BST:  
20 30 40 50 60 70 80  
Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:-15

AIM: Write a Program for Building a Function ISVALID to VALIDATE BST

ALGORITHM:

Here's the algorithm to check if a binary tree is a valid Binary Search Tree (BST):

1. Define a function isValidBST(root) that takes the root node of the binary tree as input and returns true if the tree is a valid BST and false otherwise.
2. Initialize two pointers, curr and prev, to traverse the tree. Set both pointers to the root initially.
3. Use a while loop to traverse the tree:
 - If the current node's left child is null:
 - Check if the value of the previous node (prev) is greater than or equal to the current node's value. If true, return false indicating that the tree is not a valid BST.
 - Update prev to the current node (curr).
 - Move curr to its right child.
 - If the current node's left child is not null:
 - Find the inorder predecessor (pred) of the current node (curr). The predecessor is the rightmost node in the left subtree of curr.
 - If the predecessor's right child is null, link it to the current node (curr) and move curr to its left child.
 - If the predecessor's right child is already linked to the current node (curr), unlink it, update prev, and move curr to its right child.
4. Return true if the traversal completes without any violation of the BST property.

PROGRAM:

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

    TreeNode(int x)
    {
        val = x;
        left = null;
        right = null;
    }
}

class Solution {
    public static boolean isValidBST(TreeNode root)
    {
        TreeNode curr = root;
        TreeNode prev = null;

        while (curr != null) {
            if (curr.left
                == null) {
                if (prev != null && prev.val >= curr.val)
                    return false;
                prev = curr;
            }
        }
    }
}
```

```
        curr = curr.right;
    }
    else {
        TreeNode pred = curr.left;
        while (pred.right != null
            && pred.right != curr)
            pred = pred.right;

        if (pred.right == null) {
            pred.right = curr;
            curr = curr.left;
        }
        else {
            pred.right = null;

            if (prev != null
                && prev.val >= curr.val)
                return false;
            prev = curr;
            curr = curr.right;
        }
    }
}

return true;
}

public static void main(String[] args)
{

    TreeNode root = new TreeNode(4);
    root.left = new TreeNode(2);
    root.right = new TreeNode(5);
    root.left.left = new TreeNode(1);
    root.left.right = new TreeNode(3);

    if (isValidBST(root))
        System.out.println(
            "The binary tree is a valid BST.");
    else
        System.out.println(
            "The binary tree is not a valid BST.");
}
}
```

OUTPUT:

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Progr  
The binary tree is a valid BST.  
  
Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:-16

AIM: Write a Program to Traverse a Tree using Level Order Traversal

ALGORITHM:

Here's an algorithm for the given recursive level order traversal of a binary tree:

1. **printLevelOrder() Function:**

Calculate the height of the binary tree using the height() function.

Iterate through each level of the tree from 1 to the height.

For each level, call the printCurrentLevel() function to print the nodes at that level.

2. **height() Function:**

If the root is null, return 0.

Otherwise, recursively compute the height of the left and right subtrees.

Return the maximum height of the left and right subtrees plus 1.

3. **printCurrentLevel() Function:**

If the root is null, return.

If the current level is 1, print the data of the root.

If the current level is greater than 1, recursively call printCurrentLevel() on the left and right children with level - 1.

4. **Main Function:**

Create a binary tree instance.

Initialize the tree with some nodes.

Print the level order traversal of the binary tree using the printLevelOrder() function.

This algorithm prints the nodes of the binary tree in level order traversal, starting from the root and moving down level by level.

PROGRAM:

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

class BinaryTree {

    Node root;

    public BinaryTree() { root = null; }

    void printLevelOrder()
    {
37| Page
```

```
int h = height(root);
int i;
for (i = 1; i <= h; i++)
    printCurrentLevel(root, i);
}

int height(Node root)
{
    if (root == null)
        return 0;
    else {

        int lheight = height(root.left);
        int rheight = height(root.right);

        if (lheight > rheight)
            return (lheight + 1);
        else
            return (rheight + 1);
    }
}

void printCurrentLevel(Node root, int level)
{
    if (root == null)
        return;
    if (level == 1)
        System.out.print(root.data + " ");
    else if (level > 1) {
        printCurrentLevel(root.left, level - 1);
        printCurrentLevel(root.right, level - 1);
    }
}

public static void main(String args[])
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

    System.out.println("Level order traversal of"
        + "binary tree is ");
    tree.printLevelOrder();
}
```

}

OUTPUT:

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBra
Level order traversal of binary tree is
1 2 3 4 5
Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

PROGRAM NO:-17**AIM:** Write a Program to perform Boundary Traversal on BST**ALGORITHM:**

Below is the algorithm for printing the boundary nodes of a binary tree, including the left boundary, leaf nodes, and right boundary:

1.printLeaves() Function:

- Recursively traverse the tree in an inorder fashion.
- When a leaf node (a node with no left and right children) is encountered, print its data.

2.printBoundaryLeft() Function:

- Recursively traverse the left boundary of the tree.
- Print the data of each node before traversing its children.
- If a node has no left child, traverse its right child.

3.printBoundaryRight() Function:

- Recursively traverse the right boundary of the tree.
- Traverse the right child first before printing the data of each node.
- If a node has no right child, traverse its left child.

4.printBoundary() Function:

- Print the root node's data.
- Print the left boundary of the tree, excluding the leaf nodes.
- Print the leaf nodes.
- Print the right boundary of the tree, excluding the leaf nodes.

5.Main Function:

- Create a binary tree instance.
- Initialize the tree with some nodes.
- Call the printBoundary() function to print the boundary nodes of the tree.

This algorithm ensures that the boundary nodes of the binary tree are printed in the correct order.

PROGRAM:

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

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

class BinaryTree {
    Node root;
```

```
void printLeaves(Node node)
{
    if (node == null)
        return;

    printLeaves(node.left);
    // Print it if it is a leaf node
    if (node.left == null && node.right == null)
        System.out.print(node.data + " ");
    printLeaves(node.right);
}
```

```
void printBoundaryLeft(Node node)
{
    if (node == null)
        return;

    if (node.left != null) {
        System.out.print(node.data + " ");
        printBoundaryLeft(node.left);
    }
    else if (node.right != null) {
        System.out.print(node.data + " ");
        printBoundaryLeft(node.right);
    }
}
```

```
void printBoundaryRight(Node node)
{
    if (node == null)
        return;

    if (node.right != null) {
        printBoundaryRight(node.right);
        System.out.print(node.data + " ");
    }
    else if (node.left != null) {
        printBoundaryRight(node.left);
        System.out.print(node.data + " ");
    }
}
```

```
void printBoundary(Node node)
41| Page
```



```
{
    if (node == null)
        return;

    System.out.print(node.data + " ");

    printBoundaryLeft(node.left);

    printLeaves(node.left);
    printLeaves(node.right);

    printBoundaryRight(node.right);
}

public static void main(String args[])
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(20);
    tree.root.left = new Node(8);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(12);
    tree.root.left.right.left = new Node(10);
    tree.root.left.right.right = new Node(14);
    tree.root.right = new Node(22);
    tree.root.right.right = new Node(25);
    tree.printBoundary(tree.root);
}
}
```

OUTPUT:

```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA\bin\idea-agent.jar" -jar C:\Program Files\Java\jdk-20\bin\java.exe
20 8 4 10 14 25 22
Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

PRACTICAL NO:-18

AIM: Write a Program to view a tree from left View

ALGORITHM:

Here's a breakdown of the algorithm:

1. Define a static inner class Node to represent the nodes of the binary tree. Each node has a data value, a left child, and a right child.
2. Define a method printLeftView() that takes the root node of the binary tree as input and prints the left view of the tree.
3. Inside the printLeftView() method, use a queue (implemented using a LinkedList) to perform level order traversal of the binary tree.
4. Start with adding the root node to the queue.
5. Run a loop until the queue is empty:
 - a. Get the number of nodes at the current level by getting the size of the queue.
 - b. Iterate through the nodes at the current level:
 - Poll (remove and retrieve) the node from the queue.
 - If it's the first node at the current level, print its data (this represents the leftmost node at that level).
 - Enqueue the left and right child of the current node if they exist.
6. Finally, in the main() method, create a binary tree and call the printLeftView() method to print the left view.

This algorithm ensures that only the leftmost node at each level of the binary tree is printed, which gives the left view of the tree.

PROGRAM:

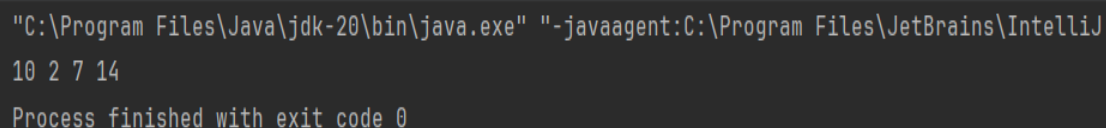
```
import java.util.*;
```

```
public class BinaryTree {  
  
    private static class Node {  
        int data;  
        Node left, right;  
  
        public Node(int data)  
        {  
            this.data = data;  
            this.left = null;  
            this.right = null;  
        }  
    }  
  
    private static void printLeftView(Node root)  
    {  
        if (root == null)  
            return;  
  
        Queue<Node> queue = new LinkedList<>();  
        queue.add(root);
```

```
while (!queue.isEmpty()) {  
    int n = queue.size();  
  
    for (int i = 1; i <= n; i++) {  
        Node temp = queue.poll();  
  
        if (i == 1)  
            System.out.print(temp.data + " ");  
  
        // Add left node to queue  
        if (temp.left != null)  
            queue.add(temp.left);  
  
        // Add right node to queue  
        if (temp.right != null)  
            queue.add(temp.right);  
    }  
}
```

```
public static void main(String[] args)  
{  
  
    Node root = new Node(10);  
    root.left = new Node(2);  
    root.right = new Node(3);  
    root.left.left = new Node(7);  
    root.left.right = new Node(8);  
    root.right.right = new Node(15);  
    root.right.left = new Node(12);  
    root.right.right.left = new Node(14);  
  
    printLeftView(root);  
}
```

OUTPUT:



```
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ  
10 2 7 14  
Process finished with exit code 0
```

CONCLUSION: The above code is successfully executed in Lab.

Practical 19

Aim: Write a Program for a basic **hash function** in a programming language of your choice. Demonstrate its usage to store and retrieve key-value pairs.

Problem Statement:

Implement a simple hash table in C. Define a hash table structure (**HashTable**) that contains an array of hash table items (**Ht_item**). Each hash table item should have a key-value pair. Include functions for creating a hash table (**create_table**), creating hash table items (**create_item**), a hash function (**hash_function**) to calculate the index for an item, and a function to print the contents of the hash table (**print_table**). Test the hash table by creating items with keys and values, inserting them into the hash table using linear probing for collision handling, and then printing the contents of the hash table.

Sample Input and Output:

Sample Input:

```
create_item(123, 48956);  
create_item(456, 89956);
```

Sample Output:

```
Index: 3, Key: 123, Value: 48956  
Index: 6, Key: 456, Value: 89956
```

Algorithm:

- Define a structure **Ht_item** to represent key-value pairs.
- Define a structure **HashTable** with an array of **Ht_item** pointers to store items.
- Implement a hash function (**hash_function**) that calculates the index for an item based on its key using modular arithmetic.
- Implement a function (**create_item**) to dynamically allocate memory for a new item, set its key and value, and return a pointer to the item.
- Implement a function (**create_table**) to create a hash table by dynamically allocating memory for a **HashTable** structure and initializing its array of items to NULL.
- In the **main** function or elsewhere, create a hash table using **create_table**.
- Create items using **create_item** with keys and values.
- Calculate the index for each item using the hash function and insert the items into the hash table.
- Implement a function (**print_table**) to print the contents of the hash table by iterating through the array and printing non-NULL items.

Code:

```
#include <stdio.h>  
#include <stdlib.h>  
#define CAPACITY 10  
  
typedef struct {  
    int key;  
    int value;  
} Ht_item;
```

```
typedef struct{
    Ht_item* items[CAPACITY];
}HashTable;

int hash_function(int key){
    return key % CAPACITY;
}

Ht_item* create_item(int key, int value){
    Ht_item* item=(Ht_item*)malloc(sizeof(Ht_item));

    item->key=key;
    item->value=value;
    return item;
}

HashTable* create_table(){
    HashTable *table=(HashTable*)malloc(sizeof(HashTable));
    for(int i=0;i<CAPACITY;i++){
        table->items[i]=NULL;
    }
    return table;
}

void print_table(HashTable* table){
    printf("_____Hash Table_____\n");
    for(int i=0;i<CAPACITY;i++){
        if(table->items[i]){
            printf("Index: %d, Key: %d, Value: %d",i,table->items[i]->key,table->items[i]->value);
            printf("\n");
        }
    }
}

int main()
{
    HashTable* table=create_table();
    Ht_item* item1=create_item(123,48956);
    table->items[hash_function(123)]=item1;
    Ht_item* item2=create_item(456,89956);
    table->items[hash_function(456)]=item2;
    print_table(table);
    return 0;
}
```



Output:

_____Hash Table_____

Index: 3, Key: 123, Value: 48956

Index: 6, Key: 456, Value: 89956

CONCLUSION: The above code is successfully executed in Lab.

Practical 20

Aim: Write a Program to Implement Two sums using HASHMAP

Problem Statement:

Given an array of integers **nums** and an integer target, return indices of the two numbers such that they add up to the target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

Sample Input and Output:

Input: nums = [2, 6, 5, 8, 11], target = 14

Output: [1, 3]

Explanation: nums[1] + nums[3] = 6 + 8 = 14, so the indices are 1 and 3.

Algorithm:

- Define a structure Ht_item & define a structure HashTable.
- Define a hash function hashFunction(key).
- Define a function createItem(key, value).
- Define a function createTable() that dynamically allocates memory for a new hash table, initializes each slot in the hash table to NULL, and returns a pointer to the newly created hash table.
- Define a function searchTable(table, key) that takes a hash table and a key as input, calculates the hash index using the hash function, and returns the item at that index in the hash table.
- Define a function insertTable(table, key, value) that takes a hash table, key, and value as input, calculates the hash index using the hash function, and inserts the item with the given key and value at that index in the hash table.
- Define a function twoSum(nums, numSize, target, returnSize) that takes an array of integers nums, its size numSize, a target integer target, and a pointer to an integer returnSize.
- Create a hash table using createTable().
- Iterate through each element in the nums array:
 - i. Calculate the complement by subtracting the current number from the target.
 - ii. Search the hash table for the complement using searchTable().
 - iii. If the complement is found in the hash table:
 - Insert the current number and its index into the hash table using insertTable().
 - If no such pair is found, set returnSize to 0 and return NULL.
- Define the main function

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define CAPACITY 10
```

```
typedef struct{
    int key;
    int value;
}Ht_item;
```

```
typedef struct{
    Ht_item* items[CAPACITY];
}HashTable;

int hash_function(int key){
    return key % CAPACITY;
}

Ht_item* create_item(int key, int value){
    Ht_item* item=(Ht_item*)malloc(sizeof(Ht_item));

    item->key=key;
    item->value=value;
    return item;
}

HashTable* create_table(){
    HashTable *table=(HashTable*)malloc(sizeof(HashTable));
    for(int i=0;i<CAPACITY;i++){
        table->items[i]=NULL;
    }
    return table;
}

Ht_item* search_table(HashTable* table,int key){
    int index=hash_function(key);
    return table->items[index];
}

void insert_table(HashTable* table,int key,int value){
    int index = hash_function(key);
    table->items[index]=create_item(key,value);
}

int *twoSum(int* nums,int numSize,int target,int* returnSize){
    HashTable* table=create_table();
    int* result=(int*)malloc(2*sizeof(int));

    for(int i=0;i<numSize;i++){
        int complement=target-nums[i];
        Ht_item* item=search_table(table,complement);
        if(item){
            result[0]=item->value;
            result[1]=i;
            *returnSize=2;
        }
    }
}
```



```
        return result;
    }
    insert_table(table,nums[i],i);
} *returnSize=0;
return NULL;
}

void print_table(HashTable* table){
    printf("_____Hash Table_____\n");
    for(int i=0;i<CAPACITY;i++){
        if(table->items[i]){
            printf("Index: %d, Key: %d, Value: %d",i,table->items[i]->key,table->items[i]->value);
            printf("\n");
        }
    }
}

int main()
{
    int nums[]={2,6,5,8,11};
    int target=14;
    int returnSize;
    int *result=twoSum(nums,sizeof(nums),target,&returnSize);
    if(result != NULL){
        printf("[%d,%d]\n",result[0],result[1]);
        return 0;
    }
}
```

Output:

[1,3]

CONCLUSION: The above code is successfully executed in Lab.

Practical 21

Aim: Write a program to find Distinct substrings in a string

Problem Statement:

You are tasked with writing a program that efficiently finds all distinct substrings within a given string. A substring is a contiguous sequence of characters within a string. However, for this problem, a substring is considered distinct if it does not match any other substring found in the string.

For example, given the string "ababa", the distinct substrings are {"a", "ab", "aba", "abab", "ababa", "b", "ba", "bab"}.

Sample Input and Output:

Sample Input:

“aabc”

Sample Output:

Aa,a,ab,bc,b,aab,abc,c,aabc

Algorithm:

- Initialize an empty set to store distinct substrings.
- Iterate through all possible starting indices 'I' of substrings in the string.
- Iterate through all possible ending indices 'j' greater than or equal to 'I' to form substrings.
- Extract the substring from index 'I' to index 'j'.
- Add the extracted substring to the set if it's not already present.
- Continue until all possible substrings have been explored.
- The set will contain all distinct substrings.

Code:

```
import java.util.HashSet;

public class DistinctSubstrings {

    public static void main(String[] args) {
        String inputString = "aabc";
        HashSet<String> distinctSubstrings = new HashSet<>();

        for (int i = 0; i < inputString.length(); i++) {
            for (int j = i + 1; j <= inputString.length(); j++) {
                distinctSubstrings.add(inputString.substring(i, j));
            }
        }

        System.out.println("Distinct substrings in " + inputString + " :");
        for (String substring : distinctSubstrings) {
            System.out.println(substring);
        }
    }
}
```

OUTPUT:

```
java -cp /tmp/Dw3zERJOM0/DistinctSubstrings  
Distinct substrings in 'aabc':  
aa  
a  
ab  
bc  
b  
aab  
abc  
c  
aabc
```

CONCLUSION: The above code is successfully executed in Lab