IARE

IARE
INSTITUTE OF AERONAUTICAL ENGINEERING
(An Autonomous Institute affiliated to JNTUH, Hyderabad)
Dundigal, Hyderabad - 500 043

LABORATORY WORK BOOK

Name of the Student : Racherla Santhosh

Class IT-B Semester 03

Course Code : ACSD11 Course Name : DS Laboratory

Name of the Course Faculty Ms. K. Laxminarayanamma Faculty ID : IARE10033

Exercise Number : 09 Week Number : 09 Date : 05/11/2024

Roll Number									
2	3	9	5	1	A	1	2	C	3

S. No.	Exercise Number	EXERCISE NAME	MARKS AWARDED					
			Aim/ Preparation	Algorithm / Procedure / Performance in the Lab	Source Code / Calculations and Graphs	Program Execution / Results and Error Analysis	Viva - Voce	Total
			4	4	4	4	4	20
1	9.1	Tree Creation						
2	9.2	Binary Tree Traversal Techniques						
3	9.3	Insertion in a binary tree in a level order.						
4	9.4	Finding the max Height of a binary tree.						
5	9.5	Deletion in a Binary Tree						
6			4		4	4	4	20
7								
8								
9								
10								
11								
12								

Signature of the Student

Signature of the Faculty

9. Trees.

9.1 Tree Creation and Basic Tree Terminologies:

AIM:- Write a Program to Create a Basic Terminologies in Tree.

PROGRAM:-

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
Public class TreeBasicTerminologies {
  Static void printChildren(int root,
      List<List<Integer>> adj) {
   System.out.print("Children of node" + root + ":");
   for(int child : adj.get(root)) {
     System.out.print(child + " ");
   }
   System.out.println();
}
```

```
Static void printLeafNodes (int root,
    List < List < Integer >> adj) {
System.out.print ("Leaf nodes : ");
for (int i=0; i< adj.Size(); i++) {
  if ( adj.get (i). isEmpty() ) {
    System.out.print (i + " ");
  }
}
System.out.println();
}
Static void printDegrees (int root,
    List < List < Integer >> adj) {
for ( int i=0; i< adj.Size(); i++) {
  System.out.println ("Degree of node" + i +
      " : " + adj.get(i).Size());
}
}
Public Static void main (String[] args) {
  Scanner Scanner = new Scanner (System.in);
```

```
System.out.print("Enter the number of nodes
                  in the tree : ");
int numberOfNodes = Scanner.nextInt();
List<List<Integer>> adj = new ArrayList<>();
for(int i=0; i<numberOfNodes; i++){
  adj.add(new ArrayList<>());
}
System.out.println("Enter the edges (parent child)
  in the format : 'parent child' Enter -1 -1 to Stop");
while(true){
  int parent = Scanner.nextInt();
  int child = Scanner.nextInt();
  if(parent == -1 && child == -1){
    break;
  }
  adj.get(parent).add(child);
}
int root = 0;
System.out.print("Enter the node to find
```

```
its children :");
int nodeForChildren = Scanner.nextInt();
PrintChildren (nodeForChildren, adj);
PrintLeafNodes (root, adj);
PrintDegrees (root, adj);
Scanner.close();
  }
}
```

OUTPUT :-

```
Enter the number of nodes in the tree : 7
Enter the edges (Parent child) in format :
'Parent child'. Enter -1 -1 to Stop.
0 1
0 2
1 3
1 4
2 5
2 6
-1 -1
```

ROLL NUMBER :

Enter the node to find its children : 1

Children of node 1 : 3 4

Leaf nodes : 3 4 5 6

Degree of node 0 : 2

Degree of node 1 : 2

Degree of node 2 : 2

Degree of node 3 : 0

Degree of node 4 : 0

Degree of node 5 : 0

Degree of node 6 : 0

9.2 Binary Tree Traversal Techniques :-

Aim :- Write a Program for Binary Tree Traversal Techniques. It can be traversed in Inorder, Preorder, PostOrder Traversal.

PROGRAM :-

```
import java.util.Scanner;
class Node {
  int data;
  Node left, right;
  Public Node (int item) {
    data = item;
    left = right = null;
  }
}
class BT {
  Node root;
  BT() {
    this.root = null;
  }
  void insert (int data) {
    root = insertRec (root, data);
  }
  void inorder (Node root) {
    if (root != null) {
      inorder (root.left);
```

ROLL NUMBER :

```
        System.out.print (root.data + " ");
        inorder (root.right);
    }
}
void Preorder (Node root) {
    if (root != NULL) {
        System.out.print (root.data + " ");
        Preorder (root.left);
        Preorder (root.right);
    }
}
void Postorder (Node root) {
    if (root != null) {
        Postorder (root.left);
        Postorder (root.right);
        System.out.print (root.data + " ");
    }
}
}
Public class BinaryTreeTraversal {
Public Static void main (String[] args) {
```

```
BT tree = new BTC);
Scanner Scanner = new Scanner ( System . in);
System . out . print ("Enter the number of nodes
                        to insert : ");
int n = Scanner . next Int ();
System . out . println ( "Enter the nodes : ");
for (int i = 0; i < n; i++) {
  int data = Scanner . nextInt();
  tree . insert (data);
}
System . out . println ("Inorder traversal : ");
tree . inorder ( tree . root);
System . out . println();
System . out . println ("Preorder traversal : ");
tree . preorder (tree . root);
System . out . println();
System . out . println ("postorder traversal : ");
tree . Postorder ( tree . root);
System . out . println();
```

```
        Scanner.close();
    }
}
```

OUTPUT :-

```
Enter the number of nodes to insert : 5
Enter the nodes : 10 5 20 3 7
Inorder traversal : 3 5 7 10 20
Preorder traversal : 10 5 3 7 20
Postorder traversal : 3 7 5 20 10
```

9.3 Insertion In A Binary Tree In Level Order :-

AIM :- Given a binary tree and a key, insert the key into the binary tree at the first position available in level order.

PROGRAM :-

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
class Node {
  int data;
  Node left, right;
  Public Node (int item) {
    data = item;
    left = right = null;
  }
}
Public class BinaryTree Insertion {
  Node root;
  Static Node insert (Node root, int key) {
    if (root == null) {
      return new Node (key);
    }
```

```
Queue<Node> queue = new LinkedList<>();
queue.add(root);
while (!queue.isEmpty()) {
    Node temp = queue.poll();
    if (temp.left == null) {
        temp.left = new Node(Key);
        break;
    } else {
        queue.add(temp.left);
    }
    if (temp.right == null) {
        temp.right = new Node(Key);
        break;
    } else {
        queue.add(temp.right);
    }
}
return root;
}
```

```
public static void main (String[] args) {
    Scanner Scanner = new Scanner (System.in);
    BinaryTree Insertion tree = new BinaryTree_Insertion();
    System.out.println ("Enter the number of nodes
                        in the tree : ");
    int n = Scanner.nextInt();
    System.out.println ("Enter the nodes: ");
    for (int i=0; i<n; i++) {
        int value = Scanner.nextInt();
        tree.root = insert (tree.root, value);
    }
    System.out.println ("Given traversal before insertion:");
    inorder (tree.root);
    System.out.println();
    System.out.println ("Enter the value to insert: ");
    int key = Scanner.nextInt();
    tree.root = insert (tree.root, key);
    System.out.println ("Inorder traversal after insertion:");
```

```
        inorder(tree.root);
        System.out.println();
        Scanner.close();
    }
}
```

OUTPUT :-

Enter the number of nodes in the tree : 6

Enter the nodes : 10 11 9 7 15 8

Inorder traversal before insertion : 7 11 15 10 8 9

Enter the value to insert : 12

Inorder traversal after insertion :

7 11 15 10 8 9 12.

9.4 Finding the Maximum Height Or Depth Of A Binary Tree :-

AIM :- Given a Binary Tree, The task is to find the height of the tree. The height of

the tree is the number of edges in the tree from the root to the deepest node.

PROGRAM :-

```
import java.util.*;
class Node {
  int data;
  Node left, right;
  Node (int data) {
    this.data = data;
    this.left = null;
    this.right = null;
  }
}
Public class MaximumDepthOfTree {
  Public Static int maxDepth (Node root) {
    if (root == null) {
      return 0;
    }
    int leftDepth = maxDepth (root.left);
```

```
int rightDepth = maxDepth(root.right);
return Math.max(leftDepth, rightDepth) + 1;
}
Public Static Node buildTree() {
Scanner Scanner = new Scanner(System.in);
System.out.println("Enter root value:");
int rootValue = Scanner.nextInt();
Queue<Node> queue = new LinkedList<>();
Node root = new Node(rootValue);
queue.add(root);
While (!queue.isEmpty()) {
Node current = queue.poll();
System.out.println("Enter left child for" +
current.data + " (-1 for no node):");
int leftValue = Scanner.nextInt();
if (leftValue != -1) {
current.left = new Node(leftValue);
```

```
        queue.add(current.left);
    }
    System.out.println("Enter right child for " + current.data +"
                        (-1 - for no node):");
    int rightValue = Scanner.nextInt();
    if (rightValue != -1) {
        current.right = new Node(rightValue);
        queue.add(current.right);
    }
}
return root;
}
Public Static void main(String[] args) {
    System.out.println("Build your binary tree");
    System.out.println("Enter -1 to indicate no child for a node");
    Node root = BuildTree();
    PrintTree(root);
    System.out.println("\n Height of the binary tree is: "+
                        maxDepth(root));
    }
}
```

OUTPUT:- Build Your Binary Tree

Enter -1 to indicate no child for a node.

Enter root value : 1

Enter left child for 1 (-1 for no node) : 2

Enter right child for 1 (-1 for no node) : 3

Enter left child for 2 (-1 for no node) : 4

Enter right child for 2 (-1 for no node) : 5

Enter left child for 3 (-1 for no node) : -1

Enter right child for 3 (-1 for no node) : -1

Enter left child for 4 (-1 for no node) : -1

Enter right child for 4 (-1 for no node) : -1

Enter left child for 5 (-1 for no node) : -1

Enter right child for 5 (-1 for no node) : -1

Tree structure :

1

2 3

4 5

Height of the binary tree is : 3.

9.5 Deletion In A Binary Tree :-

AIM :- Given a binary tree, delete a node from it by making sure that the tree shrinks from the bottom.

PROGRAM :-

```
import java.util.*;
class Node {
  int data;
  Node left, right;
  Node (int data) {
    this.data = data;
    this.left = null;
    this.right = null;
  }
}
Public class BinaryTreeDeletion {
  Static void deleteDeepest (Node root, Node delNode) {
    Queue <Node> q = new LinkedList <> ();
    q.add (root);
    Node temp = null;
    While (! q.isEmpty()) {
```

```
        temp = q.poll();
        if (temp == delNode) {
            temp = null;
            return;
        }
        if (temp.right != null) {
            if (temp.right == delNode) {
                temp.right = null;
                return;
            } else {
                q.add(temp.right);
            }
        }
        if (temp.left != null) {
            if (temp.left == delNode) {
                temp.left = null;
                return;
            } else {
                q.add(temp.left);
            }
        }
    }
}
```

```
Public static void main (String[] args) {
    Scanner scanner = new Scanner (System.in);
    System.out.println (" Enter root value : ");
    Node root = new Node ( scanner.nextInt());
    Queue <Node > q = new LinkedList<>();
    q.add (root);
    while ( !q.isEmpty()) {
      Node temp = q.poll();
      System.out.println (" Enter left child for " + temp.data + "
                          (-1 for no node) : ");
      int leftValue = scanner.nextInt();
      if ( leftValue != -1) {
        temp.left = new Node (leftValue);
        q.add (temp.left);
      }
      System.out.println (" Enter right child for " +
                          temp.data + " (-1 for no node) : ");
      int rightValue = scanner.nextInt();
      if (rightValue != -1) {
```

```
        temp.right = new Node(rightValue);
        q.add(temp.right);
      }
    }
    System.out.println("\nOriginal Tree:");
    levelOrder(root);
    System.out.println("\nEnter the value to delete:");
    int keyToDelete = scanner.nextInt();
    root = deletion(root, keyToDelete);
    System.out.println("\nTree after deleting " + keyToDelete + ":");
    levelOrder(root);
  }
}
```

OUTPUT :-

```
Enter root value : 10
Enter left child for 10 (-1 for no node) : 20
Enter right child for 10 (-1 for no node) : 30
Enter left child for 20 (-1 for no node) : -1
Enter right child for 20 (-1 for no node) : -1
Enter left child for 30 (-1 for no node) : -1
Enter right child for 30 (-1 for no node) : -1
```

Original Tree :

Level Order Traversal : 10
20 30

Enter the value to delete : 10

Tree after deleting 10 :

Level Order Traversal : 30
20

VIVA VOCE :-

1) What is Tree Data Structure ?

A) A Tree is a hierarchical data structure with a root, nodes, and edges connecting them. It has parent-child relationships and leaf nodes. Trees are widely used for organizing & processing data with inherent hierarchies.

2) What is a Binary Tree Data Structure ?

A) A binary tree is a tree data structure where each node has at most two child nodes - a left child & a right child.

3) What is the Maximum Height Of Binary Tree ?

A) The Maximum Height of a binary tree is the number of edges in the longest path from the root node to a leaf node.

4) What is Deletion in a Binary Tree ?

A) Deletion in a binary tree refers to the process of removing a node from the tree while preserving the tree's structural properties.