Institute Of Technology, Nirma university

BRANCH :- **Computer Science Engineering**

PRACTICAL SUBMISSION

|*|*STUDENT INFO*|*|

Name :- **Pratik Kansara**

Roll No. :- **20BCE510**

Division :- **E4**

|*|*SUBJECT INFO*|*|

Subject :- **Advanced Data Structures**

Practical No. :- **2**

Practical – 2

AIM :- Implement a balanced binary search tree (AVL) using model 2 (Node tree) structure. The search tree operations create, insert, delete, display should be included. The input for creating the tree should be taken from a text/CSV file. The content of the file should be a unique key-object pair.

Code:

AVLTree.java

```
package Practical2;


public class AVLTree {


    private class Node {

        int data;

        Node left, right;

        int height;

        int key;


        public Node(int key, int data) {

            this.data = data;

            this.height = 1;

            this.key = key;

        }

    }


    private Node root;


    public void insert(int key, int val) {
```

```
        this.root = insert(this.root, key, val);
    }

    private Node insert(Node node, int key, int val) {
        if (node == null) {
            return new Node(key, val);
        }

        if (key > node.key) {
            node.right = insert(node.right, key, val);
        } else if (key < node.key) {
            node.left = insert(node.left, key, val);
        } else {
            node.data = val;
            System.out.println("Key Found, Object Updated");
            return node;
        }
        node.height = Math.max(height(node.left), height(node.right)) + 1;

        int bf = balanceFactor(node);

        // LL
        if (bf > 1 && key < node.left.key) {
            System.out.println("Performing Left Left Rotation");
            return rightRotate(node);
        }
        // RR
        if (bf < -1 && key > node.right.key) {
```

```
            System.out.println("Performing Right Right Rotation");
            return leftRotate(node);
        }
        // LR
        if (bf > 1 && key > node.left.key) {
            System.out.println("Performing Left Right Rotation");
            node.left = leftRotate(node.left);
            return rightRotate(node);
        }
        // RL
        if (bf < -1 && key < node.left.key) {
            System.out.println("Performing Right Left Rotation");
            node.right = rightRotate(node.left);
            return leftRotate(node);
        }

        return node;
    }

    private int height(Node node) {
        if (node == null) {
            return 0;
        }
        return node.height;
    }

    private int balanceFactor(Node node) {
        if (node == null) {
```

```
            return 0;
        }
        return height(node.left) - height(node.right);
    }

    private Node rightRotate(Node node) {
        Node temp = node.left;
        Node temp2 = temp.right;

        // rotation
        temp.right = node;
        node.left = temp2;

        // height updation
        node.height = Math.max(height(node.left), height(node.right)) + 1;
        temp.height = Math.max(height(temp.left), height(temp.right)) + 1;

        return temp;
    }

    private Node leftRotate(Node node) {
        Node temp = node.right;
        Node temp2 = temp.left;

        // rotation
        temp.left = node;
        node.right = temp2;
```

```
        // height updation
        node.height = Math.max(height(node.left), height(node.right)) + 1;
        temp.height = Math.max(height(temp.left), height(temp.right)) + 1;

        return temp;
    }

    public void display() {
        System.out.println("Your AVL Tree is : ");
        inord(this.root);
    }

    private void inord(Node node) {
        if (node != null) {
            String str = "";

            if (node.left == null) {
                str += ".";
            } else {
                str += node.left.key;
            }

            str += " <- " + node.key + " --> " + node.data + " -> ";

            if (node.right == null) {
                str += ".";
            } else {
                str += node.right.key;
```

```
        }


        System.out.println(str);
        inord(node.left);
        inord(node.right);
    }
}


private int findMin(Node root) {
    while (root.left != null) {
        root = root.left;
    }
    return root.key;
}


private Node delete(Node node, int key) {
    if (node == null) {
        System.out.println("Tree is Empty!");
        return node;
    } else if (key < node.key) {
        node.left = delete(node.left, key);
    } else if (key > node.key) {
        node.right = delete(node.right, key);
    } else {
        if (node.left == null) {
            Node temp = node;
            node = node.right;
        } else if (node.right == null) {
```

```
            Node temp = node;
            node = node.left;
        } else {
            int minn = findMin(root.right);
            node.key = minn;
            node.right = delete(node.right, minn);
        }
    }
    if (node == null) {
        return node;
    }
    node.height = height(node);

    int balance = node.height;

    if ((node.left).height >= 0 && balance > 1) {
        return rightRotate(node);
    } else if ((node.right).height <= 0 && balance < -1) {
        return leftRotate(node);
    } else if ((node.left).height < 0 && balance > 1) {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }

    else if ((node.right).height > 0 && balance < -1) {
        node.right = rightRotate(node.left);
        return leftRotate(node);
    }
```

```
        return node;

    }

}
```

AVLRunner.java

```
package Practical2;


import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.Scanner;


import Practical1.Inputmaker;


public class AvlRunner {


    private class Pair {

        public int v1, v2;


        public Pair(int a, int b) {

            this.v1 = a;

            this.v2 = b;

        }

    }


    private ArrayList<Pair> arr;
```

```
    public AvlRunner() {

        arr = new ArrayList<Pair>();

        fetchInput();

    }



    private void fetchInput() {

        try {

            String line = "";

            BufferedReader br = new BufferedReader(new
FileReader("Practical2\\input.csv"));

            while ((line = br.readLine()) != null) {

                String[] keyval = line.split(",");

                Pair p = new Pair(Integer.parseInt(keyval[0]),
Integer.parseInt(keyval[1]));

                arr.add(p);

            }

            br.close();

            // System.out.println("Input : " + arr);

        } catch (Exception e) {

            System.out.println(e.toString());

        }

    }



    public static void main(String[] args) {

        AVLTree avl1 = new AVLTree();

        AvlRunner avr = new AvlRunner();

        Scanner sc = new Scanner(System.in);

        while (true) {

            System.out.println("1. For fetch Input from file and make AVL tree
```

```
from that input");

            System.out.println("2. For display AVL tree");

            System.out.println("3. For Exit");

            int ch = sc.nextInt();



            switch (ch) {

                case 1:

                    for (Pair p : avr.arr) {

                        System.out.println("inserting : " + p.v1 + " -> " +
p.v2);


                        avl1.insert(p.v1, p.v2);

                    }

                    System.out.println("AVL Tree is Builded Successfully");

                    break;



                case 2:

                    avl1.display();

                    break;



                case 3:

                    System.exit(0);

                default:

                    System.out.println("Please Select from above choice!!");

            }

        }

    }

}
```

input.csv

Practical2 > in

```
1   11,1
2   9,2
3   19,3
4   13,4
5   10,5
6   14,6
7   12,7
8   8,8
9   18,9
10  17,10
11  15,11
12  16,12
```

OUTPUT

```
1. For fetch Input from file and make AVL tree from that input
2. For Delete Node From AVL Tree
3. For display AVL tree
4. For Exit
1
inserting : 11 -> 1
inserting : 9 -> 2
inserting : 19 -> 3
inserting : 13 -> 4
inserting : 10 -> 5
inserting : 14 -> 6
Performing Left Right Rotation
inserting : 12 -> 7
inserting : 8 -> 8
inserting : 18 -> 9
inserting : 17 -> 10
Performing Left Left Rotation
inserting : 15 -> 11
Performing Right Right Rotation
inserting : 16 -> 12
Performing Left Right Rotation
AVL Tree is Builded Successfully
```

```
1. For fetch Input from file and make AVL tree from that input
2. For Delete Node From AVL Tree
3. For display AVL tree
4. For Exit
3
Your AVL Tree is :
11 <- 14 --> 6 -> 18
9 <- 11 --> 1 -> 13
8 <- 9 --> 2 -> 10
. <- 8 --> 8 -> .
. <- 10 --> 5 -> .
12 <- 13 --> 4 -> .
. <- 12 --> 7 -> .
16 <- 18 --> 9 -> 19
15 <- 16 --> 12 -> 17
. <- 15 --> 11 -> .
. <- 17 --> 10 -> .
. <- 19 --> 3 -> .
```

```
1. For fetch Input from file and make AVL tree from that input
2. For Delete Node From AVL Tree
3. For display AVL tree
4. For Exit
2
Enter Key to Delete:
10
10 is deleted Successfully.
1. For fetch Input from file and make AVL tree from that input
2. For Delete Node From AVL Tree
3. For display AVL tree
4. For Exit
3
Your AVL Tree is :
. <- 8 --> 8 -> 14
11 <- 14 --> 6 -> 18
9 <- 11 --> 1 -> 13
. <- 9 --> 2 -> .
12 <- 13 --> 4 -> .
. <- 12 --> 7 -> .
16 <- 18 --> 9 -> 19
15 <- 16 --> 12 -> 17
. <- 15 --> 11 -> .
. <- 17 --> 10 -> .
. <- 19 --> 3 -> .
```

```
1. For fetch Input from file and make AVL tree from that input
Your AVL Tree is :
9 <- 11 --> 1 -> 14
. <- 9 --> 2 -> .
13 <- 14 --> 6 -> 18
12 <- 13 --> 4 -> .
. <- 12 --> 7 -> .
16 <- 18 --> 9 -> 19
15 <- 16 --> 12 -> 17
. <- 15 --> 11 -> .
. <- 17 --> 10 -> .
. <- 19 --> 3 -> .
1. For fetch Input from file and make AVL tree from that input
2. For Delete Node From AVL Tree
3. For display AVL tree
4. For Exit
4
```