K L UNIVERSITY

FRESHMAN ENGINEERING DEPARTMENT

A Project Based Lab Report

On

AVL Tree Application

SUBMITTED BY:

2010030071	Jyothin Movva
2010030113	N. Hemanth Srivathsav
2010030493	Vishaladitya Valluru
2010030124	P. Vivek Vardhan Reddy

UNDER THE ESTEEMED GUIDANCE OF

Dr. G. Rekha

HOD

KL UNIVERSITY

Green fields, Vaddeswaram – 522 502

Guntur Dt., AP, India.

DEPARTMENT OF BASIC ENGINEERING SCIENCES

CERTIFICATE

This is to certify that the project based laboratory report entitled "AVL Tree Application " submitted by **Mr.Jyothin Movva(2010030071),Mr.N. Hemanth Srivathsav (2010030113), Mr.Vishaladitya Valluru(2010030493), Mr. P. Vivek Vardhan Reddy(2010030124)** and bearing Regd. No. <REGD.NO> to the Department of Basic Engineering Sciences, KL University in partial fulfillment of the requirements for the completion of a project in "Object Oriented Programming-19CS1203" course in I B Tech II Semester, is a bonafide record of the work carried out by him/her under my supervision during the academic year 2020-21.

PROJECT SUPERVISOR	HEAD OF THE DEPARTMENT
< GUIDE NAME>	Dr. G. REKHA

ACKNOWLEDGEMENTS

The satisfaction that accompanies the successful completion of any task would be incomplete without mention of the people who made it possible and whose encouragement and guidance has been a source of inspiration throughout the course of the project. We express our sincere gratitude to all our Professors of Computer Science & Engineering, K L University, Hyderabad, for their precious suggestions, motivation, and cooperation for the successful completion of this project. It is our privilege.

I express the sincere gratitude to our Principal Dr. L. Koteswara Rao for his administration towards our academic growth.

I express sincere gratitude to our Coordinator and HOD-BES Dr. G. Rekha for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

Name: Regd. No:

2010030071 (Jyothin Movva)

2010030113 (N. Hemanth Srivathsav)

2010030124 (Vishaladitya Valluru)

2010030493 (P. Vivek Vardhan Reddy)

ABSTRACT

The project involves AVL tree implementation which uses Strings as data. This application involves using an AVL tree to arrange the data using single rotations and double rotations. The application has 3 types of transversal functions which we can access the data with. The traversal methods are in-order, pre-order and post-order.

The application is taking advantage of graphical user interface to take the data as input and output the data. It is also using file system to store the inputted data for future use outside the application. The application also the counts the number of occurrences.

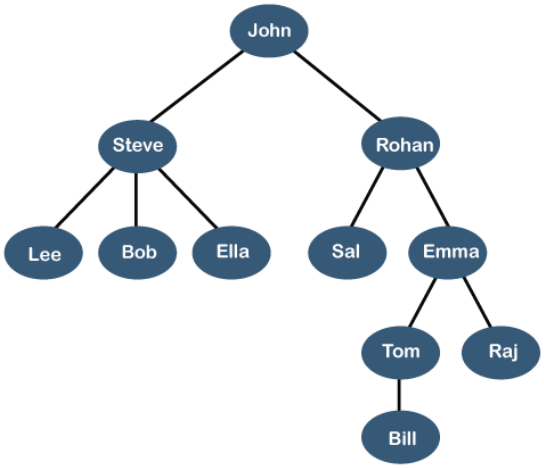
The use of graphical user interface makes the application user-friendly and easy to use. The application is lightweight and can virtually work on any system with java idk 8+.

INDEX

INTRODUCTION

What are Trees in Data Structures?

A tree is a widely used abstract data type, that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes. Simply put, a tree is a non-linear data structure that consists of nodes connected by edges.



Example of A Tree

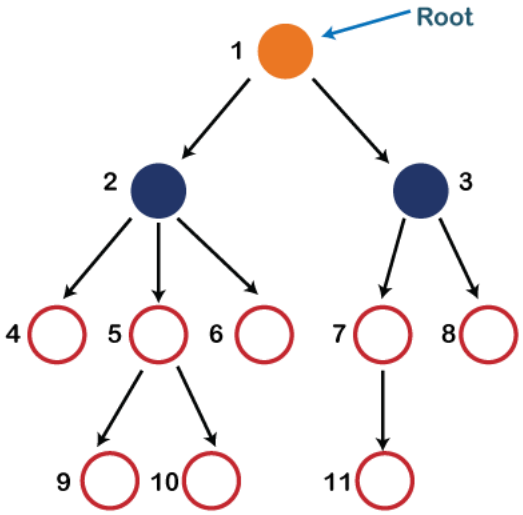
The above tree shows the organization hierarchy of a company. John is the CEO of the company, hence he of the highest hierarchical level in the tree. John has two direct reports Steve and Rohan. Steve has three direct reports Lee, Bob and Ella, where Steve is a manager. Rohan has two direct reports named Sal and Emma. Emma has two direct reports, Tom and Raj. Tom has one direct report, Bill. This structure above is a Tree.

Key Points of the Tree data structure.

- A tree data structure is defined as a collection of objects or entities known as nodes, that are linked together to represent or simulate hierarchy.
- A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a tree are arranged in multiple levels.

- In the tree data structure, the topmost node is known as the root node. Each node contains some data, and data given can be of any type. In the above tree structure, the node contains the data of string type.
- Each node contains some data and the link or reference of other nodes that are called children.

Terminology used:



In the above structure, each node is labelled with a number. Each arrow shown in the above figure is known as a link between the two nodes.

- **Root**: The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that does not have any parent. In the above structure, node 1 is the root node of the tree. If a node is directly linked to some other node, it would be called a parent-child relationship.
- **Child node**: If the node is a descendant of any node, then the node is called a child node.
- **Parent**: If the node contains any sub-node, then that node is called the parent of that sub node.
- **Sibling:** The nodes with the same parent are called sibling nodes.

- **Leaf Node**: The node of a tree with no children is called a leaf node. A leaf node is the bottom-most node in a tree.

- **Internal nodes**: A node that has at least one child is known as an internal node.
- **Ancestor node**: An ancestor of a node is any predecessor node on the path from the root to that node. A root node has no ancestors. In the above tree, nodes 1,2 and 5 are the ancestors of node 10.
- **Descendant**: The immediate successor of the given node is known as a descendant of that node. Example, in the above tree, node 10 is the descendant of node 5.

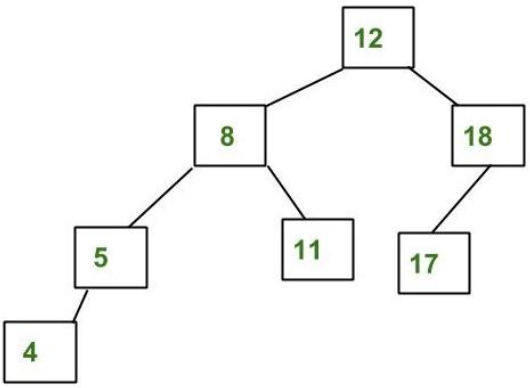
Applications of Trees:

- **Storing naturally hierarchical data:** Trees are used to store the data in the hierarchical structure. For example, the file system. The file system stored on the disc drive, the file and folder are in the form of the naturally hierarchical data and stored in the form of trees.
- **Organize data:** It is used to organize data for efficient insertion, deletion and searching. For example, a binary tree has a log(N) time for searching an element.
- **Heap:** It is also a tree data structure implemented using arrays. It is used to implement priority queues.
- **B- Tree and B+ Tree:** B-Tree and B+ Tree are the tree data structures used to implement indexing in databases.
- **Routing table:** The tree data structure is also used to store the data in routing tables in the routers.

Introduction to AVL trees:

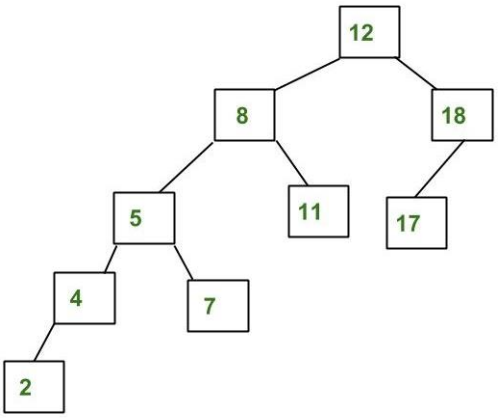
In the subject of computer science and data structures, an AVL tree (named after the inventors Adelson-Velsky and Landis) is a self-balancing binary search tree. It was the first such data structure to be invented. In an AVL tree, the heights of the two child subtrees of any node differ by at most one, if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion and deletion all take O(log n) time in both the average and worst cases, where "n" is the number of nodes in the tree

prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.



An example of an AVL tree.

The above tree is an AVL tree, because the differences between height or left and right subtrees for every node is less than or equal to 1.

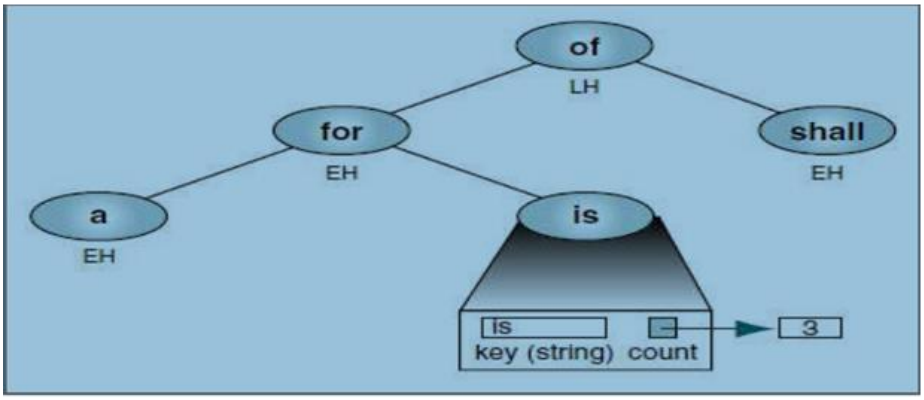


An example of a tree that is NOT AVL.

The above tree is not AVL as the differences between heights of left and right subtrees for 8 and 12 is greater than 1.

The Project:

Create an AVL tree application that uses a tree structure containing all of the words in a document, with a count of the number of times each word is used. Each entry is the AVL tree contains a word from the document and a pointer to an integer that contains a count of the number of times the word appears in the document.



Module 1:

- Program: Counts the words in a file.
- Function: Reads the file and creates an AVL tree containing list of all words used in the file with count of the number of times each word is found in the file.
- Function: Reads one word from file.
- Function: Prints the list with the count for each word.
- Function: Prints one word from the list with its count.

AIM

Advantages:

- Advantages of AVL Tree application is that the height of the avl tree is always balanced.
- It is automated because it is self-balancing the string data which is given as input.
- The application is lightweight and is user friendly.

Disadvantages: -

- AVL trees have high constant factors for some operations. For example, restructuring is an expensive operation, and an AVL tree may have to re-balance itself in the worst case during a removal of a node.
- Deletions are expensive. It is still logarithm to delete a node, but you may have to "rotate" all the way up to the root of the tree. In other words, a bigger constant. Red-Black trees only must do 1 rotate.

SYSTEM REQUIREMENTS

- SOFTWARE REQUIREMENTS:
  The major software requirements of the project are as follows:
  Language : JAVA
  Operating system: Windows XP or later.
  Tools: JDK , Visual Studio Code.

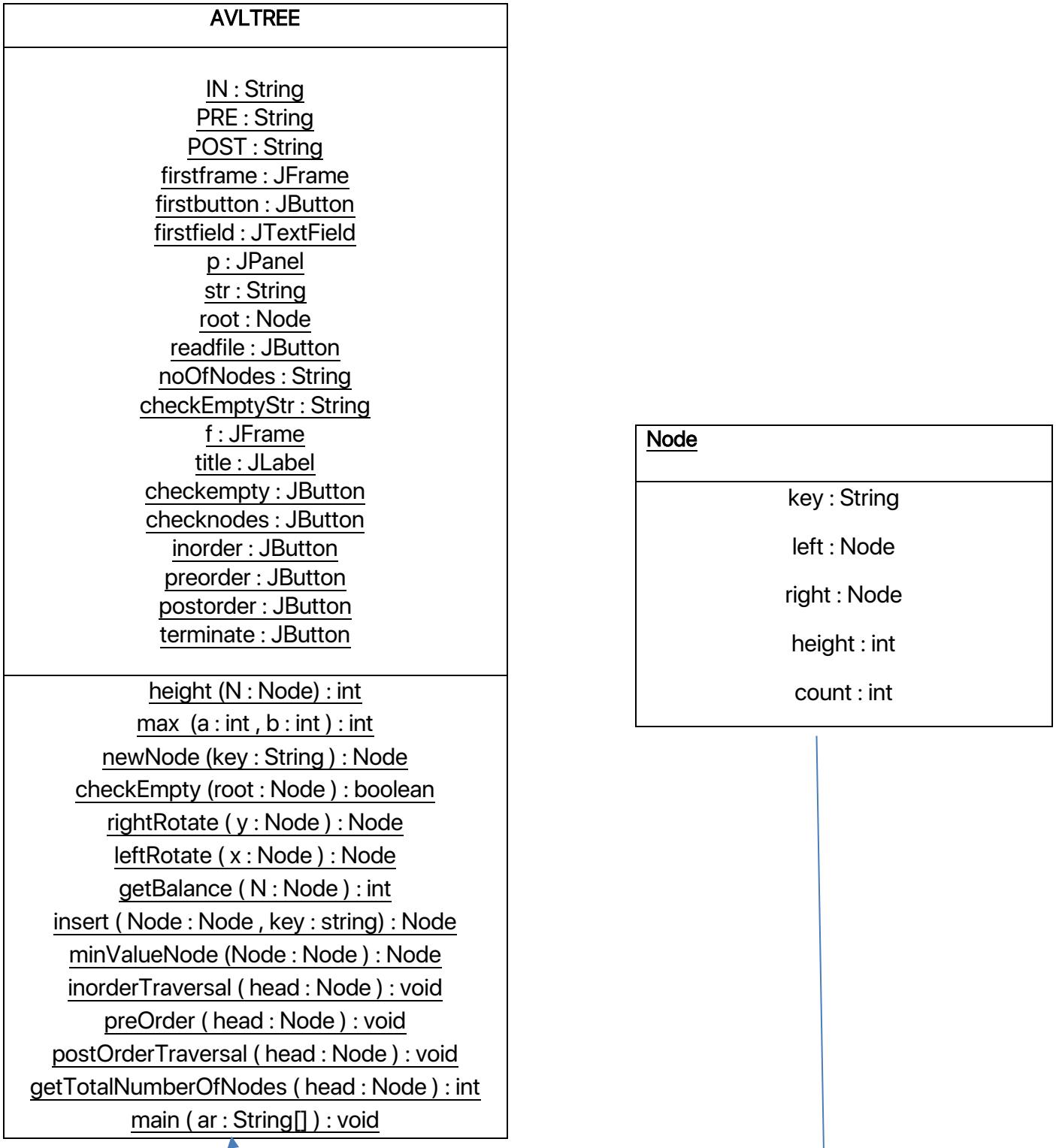
- HARDWARE REQUIREMENTS:

The hardware requirements that map towards the software are as follows:

Ram : 1+

Processor: i3+

CLASS DIAGRAM



IMPLEMENTATION

ALGORITHM

Initialise String IN,PRE,POST

Scanf(String data)

avl.(subStrings(StringData)) [Split String data with respect to " ".]

if(BF of x is 2 and y is left child)

perform LL rotation.

if(BF of x is -2 and y is right child)

perform RR rotation.

if(BF of x is 2 and BF of y is -1 and is right child of x)

perform LR rotation.

if(BF of x is -2 and BF of y is 1 and is left child of x)

perform RL rotation.

Display options – INORDER, PREORDER, POSTORDER, CHECKEMTY, COUNTNODES,TERMINATE

If(INORDER

Display inorder String

If(PREORDER)
Display preorder string

If(POSTORDER)
display postorder string

If(TERMINATE)

Set all Strings to null

And system.exit

PROGRAM

```
package project;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.*;

class avl
{
    static JTextArea g = new JTextArea();
    static String IN = "";
    static String PRE = "";
    static String POST = "";
    static JFrame firstframe = new JFrame("textfield");
    static JButton firstbutton = new JButton("submit");
    static JTextField firstfield = new JTextField(16);
    static JPanel p  = new JPanel();
    static String str;
    static Node root = null;
    static JButton readfile = new JButton("readfile");
    static String noOfNodes;
    static String checkEmptyStr;
    static JFrame f=new JFrame("avl");
    static JLabel title=new JLabel("AVLIMPLIMENTATION");
    static JButton checkempty=new JButton("CHECKEMPTY");
    static JButton checknodes=new JButton("CHECKNODES");
    static JButton inorder=new JButton("INORDER");
    static JButton preorder=new JButton("PREORDER");
    static JButton postorder=new JButton("POSTORDER");
    static JButton terminate=new JButton("TERMINATE");

    static class Node{
        String key;
        Node left;
        Node right;
        int height;
        int count;
    }
    static int height(Node N){
        if (N == null)
            return -1;
        return N.height;
    }
    static int max(int a, int b){
        if(a>b)
        return a;
        else
```

```
        return b;
    }
    static Node newNode(String key){
        Node node = new Node();
        node.key = key;
        node.left = null;
        node.right = null;
        node.height = 0;
        node.count = 1;
        return (node);
    }
    static boolean checkEmpty(Node root){
        if(root == null)
            return true;
        else
            return false;
    }
    static Node rightRotate(Node y){
        Node x = y.left;
        Node T2 = x.right;
        x.right = y;
        y.left = T2;
        y.height = max(height(y.left), height(y.right)) + 1;
        x.height = max(height(x.left), height(x.right)) + 1;
        return x;
    }
    static Node leftRotate(Node x){
        Node y = x.right;
        Node T2 = y.left;
        y.left = x;
        x.right = T2;
        x.height = max(height(x.left), height(x.right)) + 1;
        y.height = max(height(y.left), height(y.right)) + 1;
        return y;
    }
    static int getBalance(Node N)
    {
        if (N == null)
            return 0;
        return height(N.left) - height(N.right);
    }

    static Node insert(Node Node, String key)
    {
        if (Node == null)
            return (newNode(key));

        if (key.compareTo(Node.key) == 0)
        {
            (Node.count)++;
```

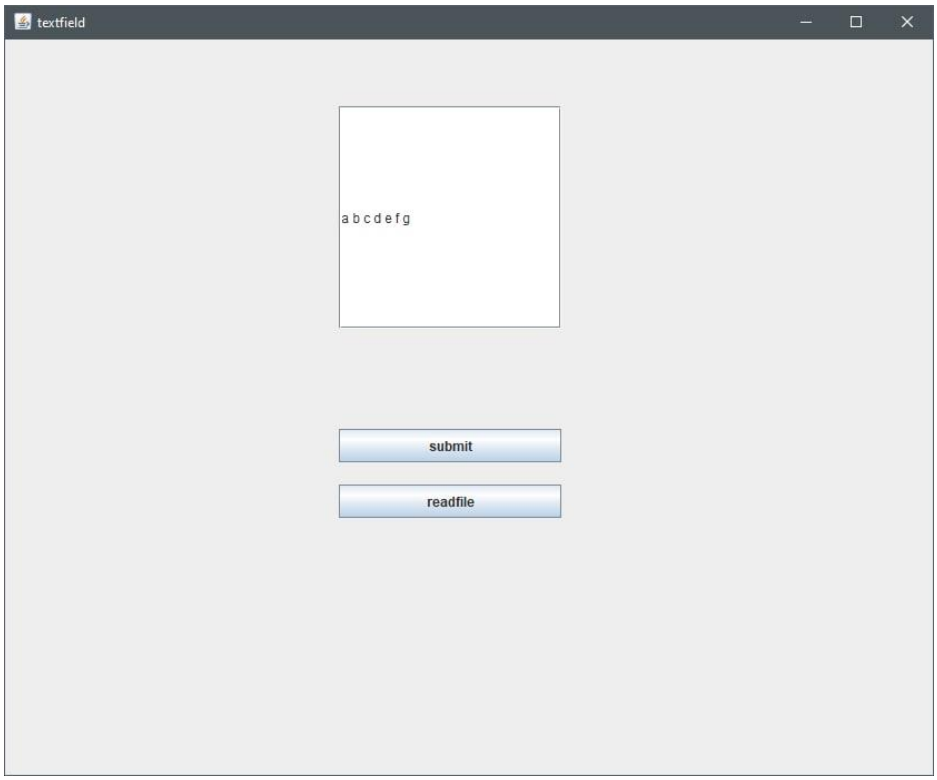
```
            return Node;
        }

        if (key.compareTo(Node.key) < 0)
            Node.left = insert(Node.left, key);
        else
            Node.right = insert(Node.right, key);

        Node.height = max(height(Node.left), height(Node.right)) + 1;
       int balance = getBalance(Node);
        if (balance > 1 && key.compareTo(Node.left.key)<0)
            return rightRotate(Node);
        if (balance < -1 && key.compareTo(Node.right.key) > 0)
            return leftRotate(Node);
        if (balance > 1 && key.compareTo(Node.left.key) > 0) {
            Node.left = leftRotate(Node.left);
            return rightRotate(Node);
        }
        if (balance < -1 && (key.compareTo(Node.right.key) < 0) ) {
            Node.right = rightRotate(Node.right);
            return leftRotate(Node);
        }
        return Node;
    }
    static Node minValueNode(Node Node)
    {
        Node current = Node;
        while (current.left != null)
            current = current.left;

        return current;
    }
    static void inorderTraversal(Node head)
    {
        if (head != null)
        {
            inorderTraversal(head.left);
            IN = IN+("Word = "+head.key+" : Count = "+head.count+"\n");
            inorderTraversal(head.right);
        }
    }
    static void preOrder(Node head)
    {
        if (head != null)
        {
            PRE = PRE+("Word = "+head.key+" : Count = "+head.count+"\n");
            preOrder(head.left);
            preOrder(head.right);
        }
    }
```

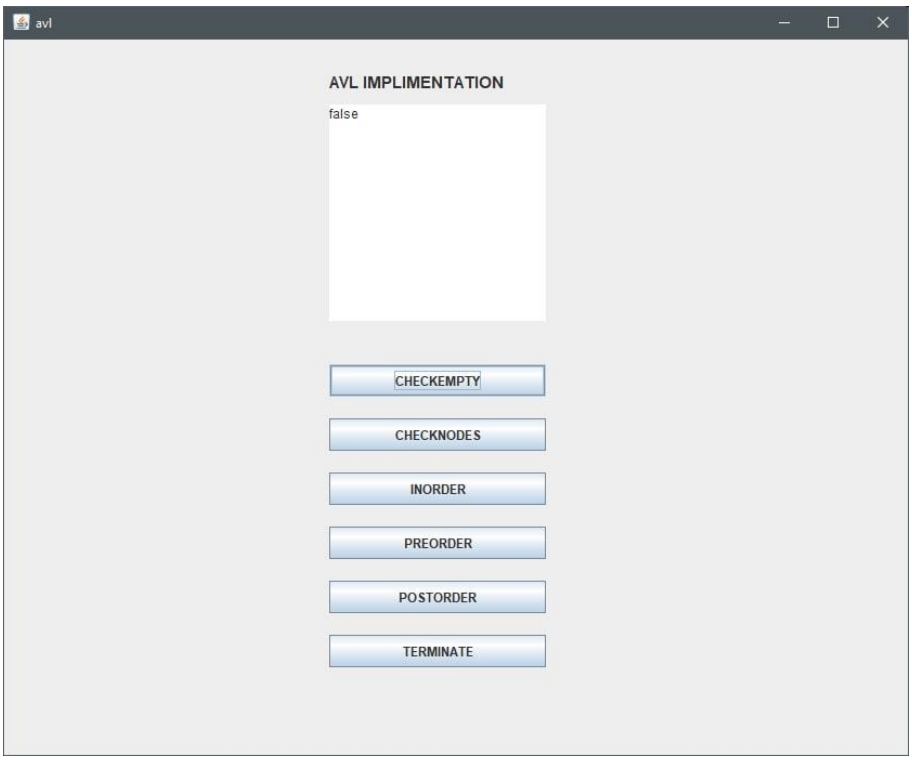
```
    static void postorderTraversal(Node head)
    {
        if (head != null)
        {
            postorderTraversal(head.left);
            postorderTraversal(head.right);
            POST = POST+("Word = "+head.key+" : Count = "+head.count+"\n");
        }
    }
    static int getTotalNumberOfNodes(Node head)
    {
        if (head == null)
            return 0;
        else{
            int length = 1;
            length = length + getTotalNumberOfNodes(head.left);
            length = length + getTotalNumberOfNodes(head.right);
            return length;
        }
    }
    public static void main(String args[])throws IOException{
        title = new JLabel("AVL IMPLIMENTATION");
        title.setFont(new Font("Arial", Font.BOLD, 15));
        title.setSize(300, 20);
        title.setLocation(300, 30);
        f.add(title);
        g.setBounds(300,60,200,200);
        checkempty.setBounds(300,300,200,30);
        checknodes.setBounds(300,350,200,30);
        inorder.setBounds(300,400,200,30);
        preorder.setBounds(300,450,200,30);
        postorder.setBounds(300,500,200,30);
        terminate.setBounds(300,550,200,30);
        firstfield.setBounds(300,60,200,200);;
        firstbutton.setBounds(300,350,200,30);
        readfile.setBounds(300,400,200,30);
        f.add(g);
        f.add(checkempty);
        f.add(checknodes);
        f.add(inorder);
        f.add(preorder);
        f.add(postorder);
        f.add(terminate);
        f.setSize(850,700);
        p.setLayout(null);
        f.setLayout(null);
        f.setVisible(false);
        p.add(readfile);
        p.add(firstfield);
        p.add(firstbutton);
```

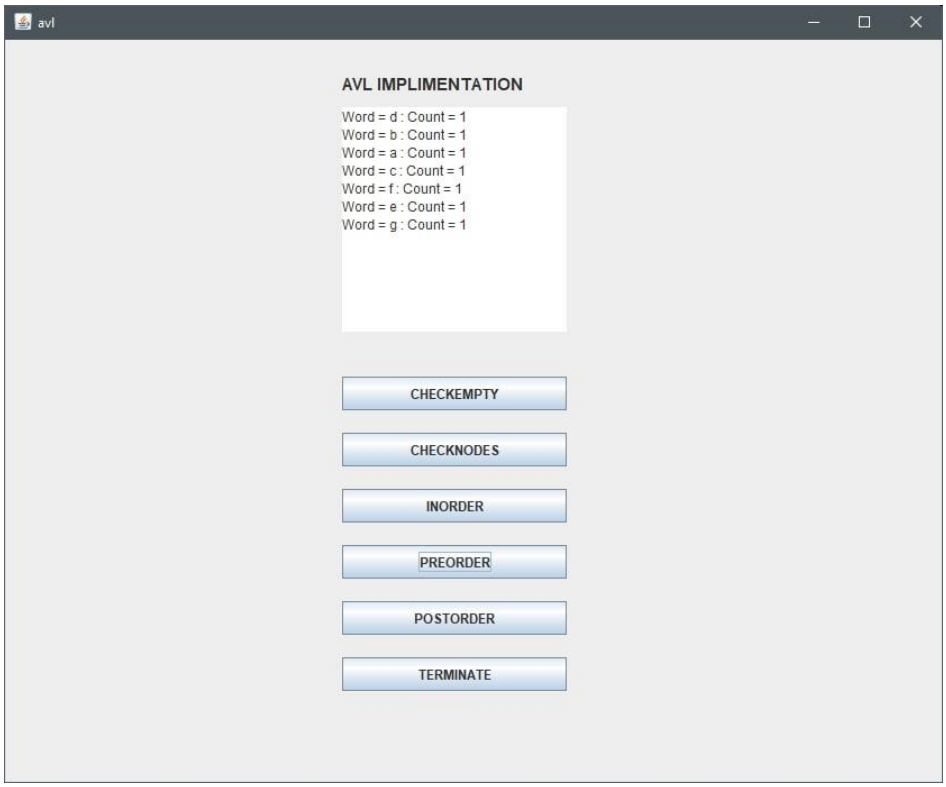
```
        firstframe.add(p);
        firstframe.setSize(850, 700);
        firstframe.setVisible(true);
        firstbutton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    str = firstfield.getText();
                    System.out.println("read ->"+str);
                    try{
                        BufferedWriter out = new BufferedWriter(new FileWriter("text.txt")
);
                                        out = new BufferedWriter(new FileWriter("text.txt"
,true));
                                        out.append(str);
                                        out.close();

                                    }
                    catch (IOException g){
                        System.out.println(g);
                    }
                    }
                    });
        readfile.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                try{
                    File f1=new File("text.txt");
                    ArrayList<String> words = new ArrayList<String>();
                    Scanner fr = new  Scanner(f1);
                    String s[]  = null;
                    while(fr.hasNext())
                    {
                    String z = fr.nextLine() ;
                    s= z.split(" ");
                    for(String i : s )
                    words.add(i);
                    }
                    System.out.println("file ->"+words);
                    for(String i : words )
                    root = insert(root,i);
                    }
                    catch(IOException h){System.out.print(g);}
                    inorderTraversal(root);
                    postorderTraversal(root);
                    preOrder(root);
                    noOfNodes = "Number of Nodes -
> "+String.valueOf(getTotalNumberOfNodes(root));
                    checkEmptyStr = String.valueOf(checkEmpty(root));
                    firstframe.setVisible(false);
                        f.setVisible(true);
                    }
                    });
```

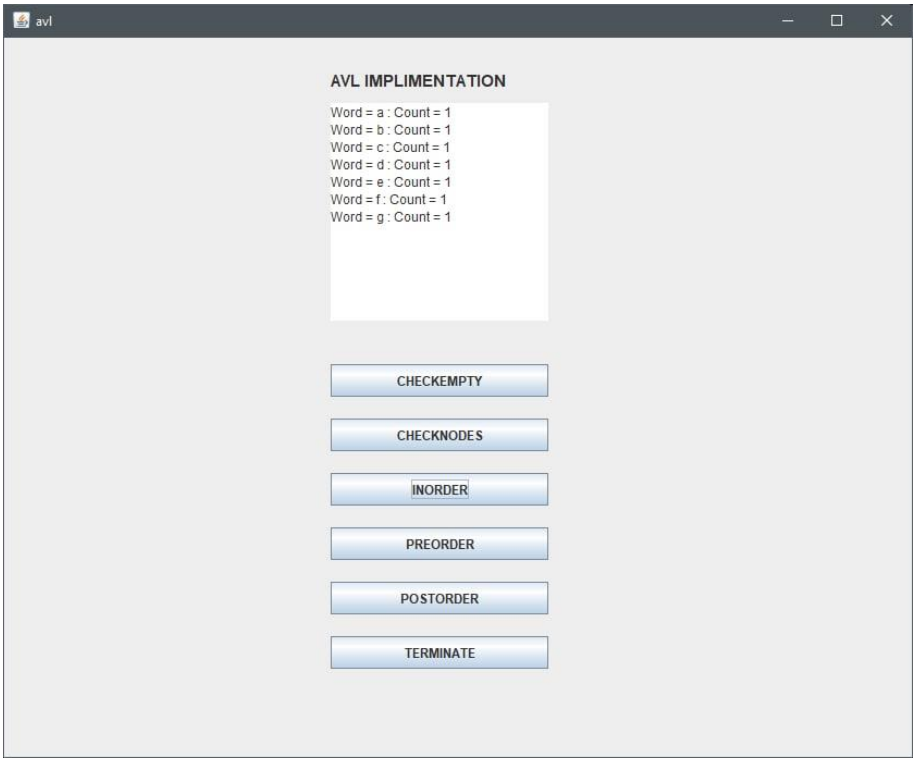
```
        checkempty.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    g.setText(checkEmptyStr);
                    }
                    });
        checknodes.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    g.setText(noOfNodes);
                    }
                    });
        inorder.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    g.setText(IN);
                    }
                    });
        preorder.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent e){
                    g.setText(PRE);
                    }
                    });
        postorder.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    g.setText(POST);
                    }
                    });
        terminate.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                    IN = null;
                    POST = null;
                    PRE = null;
                    noOfNodes = null;
                    checkEmptyStr = null;
                    System.out.print("strings reset\nTERMINATED\n");
                    System.exit(0);
                    }
                    });
        firstframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
}
```

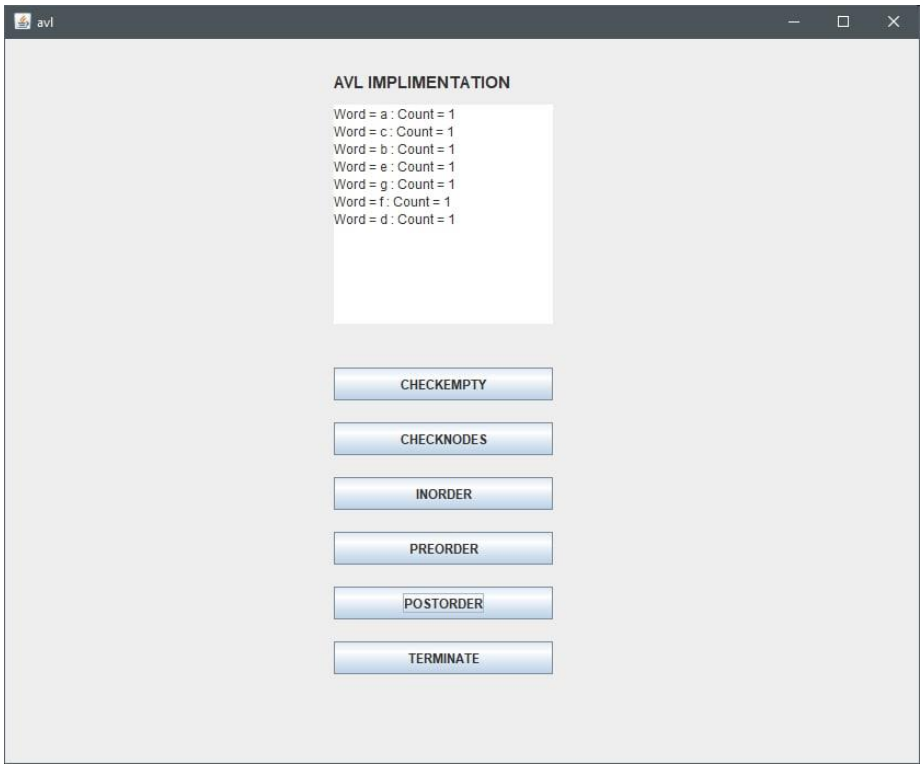
OUTPUTS


textfield
a b c d e f g
submit
readfile


avl
AVL IMPLIMENTATION
false
CHECKEMPTY
CHECKNODES
INORDER
PREORDER
POSTORDER
TERMINATE


avl
AVL IMPLIMENTATION
Word = d : Count = 1
Word = b : Count = 1
Word = a : Count = 1
Word = c : Count = 1
Word = f : Count = 1
Word = e : Count = 1
Word = g : Count = 1
CHECKEMPTY
CHECKNODES
INORDER
PREORDER
POSTORDER
TERMINATE


avl
AVL IMPLIMENTATION
Word = a : Count = 1
Word = b : Count = 1
Word = c : Count = 1
Word = d : Count = 1
Word = e : Count = 1
Word = f : Count = 1
Word = g : Count = 1
CHECKEMPTY
CHECKNODES
INORDER
PREORDER
POSTORDER
TERMINATE




```
PS D:\work\klstuff\sem 2\ds\project> java .\final.java -d .
read ->a b c d e f g
file ->[a, b, c, d, e, f, g]
strings reset
TERMINATED
PS D:\work\klstuff\sem 2\ds\project>
```

CONCLUSION

This application is the implementation of AVL tree. This application can be implemented in employer/student data management systems. It is a user friendly code which is also light weight so it can function on any kind of system without much hassle.