ICS-202 Lab Project

TERM 221

DATE: 11/12/2022

Name: BASIL ALASHQAR & ID: 202045700

Name: MOHAMMED ALHUSSAINI & ID: 202029480

Report

- **In this lab project we have designed and implemented a Dictionary data structure that is based on AVL Tree data structure.**
- **Firstly, we used the AVL Tree data structure which was completed in LAB 07.**
- **Why AVL Tree?**
  1- **AVL trees are balanced binary search trees that guarantee O(logn) worst case complexity for search, insertion and deletion.**
  2- **AVL trees are mostly used for in-memory sorts of sets and dictionaries. AVL trees are also used extensively in database applications in which insertions and deletions are fewer but there are frequent lookups for data required.**
- **Created a new Class for Dictionary data structure that extends the AVL Tree class.**
- **Started by initializing the required Constructors**
  public Dictionary(String s) > Insert the string s to an empty AVL tree.
  public Dictionary( ) > Initialize a new empty AVL Tree "size = 0"
  public Dictionary(File f) > Initialize a new empty AVL Tree then take each word on a given text file and store it in a String variable and insert it on the AVL Tree.

- **Wrote the required methods (addWord, findWord, deleteWord, findSimilar, saveDictionary):**
  1- **public void addWord(String s) throws WordAlreadyExistsException**

```
if (tree.isInTree(s)) { //if the word is already in the tree
        throw new WordAlreadyExistsException("Word already
exists!"); //throw exception
    } else { //if the word is not in the tree
        tree.insert(s); //insert the word into the tree
        System.out.println("Word added successfully!"); //print
success message
    }
```

This method starts by firstly checking if the string is already on the dictionary AVL Tree or not by using (isInTree(s) boolean method) if it gives true the method will throw WordAlreadyExistsException if it's a new word then it will be added to the dictionary avl tree using insert method.

  2- **public boolean findWord(String s)**

```
if (tree.isInTree(s)) { //if the word is in the tree
    System.out.println("Word found!"); //print success message
    return true; //return true
} else { //if the word is not in the tree
    System.out.println("Word not found!");  //print error message
    return false; //return false
}
```

This method will start by checking if string s is in the Dictionary AVL Tree using isInTree(s) method it will return true otherwise it will return false.

  3- **public void deleteWord(String s) throws WordNotFoundException**

```
if (tree.isInTree(s)) { //if the word is in the tree
    tree.deleteAVL(s); //delete the word from the tree
    System.out.println("Word deleted successfully!"); //print
success message
} else { //if the word is not in the tree
    throw new WordNotFoundException("Word not found!"); //throw
exception
}
```

This method firstly will make sure if the string s is in the Dictionary AVL Tree using isInTree(s) method, if it returns true, AVLTree method (deleteAVL(s)) will be used otherwise WordNotFoundException will be thrown.

4- public String[ ] findSimilar (String s)

```
    String similar =""; //creating new array of similar words
    for(int i = 0; i < s.length(); i++){ //for each letter in the
word
        String temp = s.substring(0, i) + "" + s.substring(i +
1); //temp is the word without the letter
        if (tree.isInTree(temp)){ //if the word is in the tree
            // check if duplicate don't add it
            if(!similar.contains(temp)){
                //add the word to the string similar
                similar += temp + " ";}

        }
        for(int j = 0; j < 26; j++){ //for each letter in the
alphabet
            char c = (char) (j + 'a'); //c is the letter
            temp = s.substring(0, i) + c + s.substring(i + 1);
//temp is the word with the letter added
            if (tree.isInTree(temp)){ //if the word is in the
tree
                // check if duplicate don't add it
                if(!similar.contains(temp)){
                    //add the word to the string similar
                    similar += temp + " ";
                }
            }
        }
    }
    String[] similarWords = similar.split(" "); //splitting the
string into an array
    return similarWords; //return the array of similar words
}
```

This method is the most complicated one in the dictionary data structure, Firstly it will initialize a new String variable called “similar” then a for loop will be used to iterate over each character on String s, it will initialize a new string temp that stores the string s without the letter iterated then check if its in the Dictionary AVL Tree, it will add it to the string “similar” with a space then there’s another loop inside the first loop that will iterate over all alphabet letters and each time it will add the new letter in the place of the originialy removed one firstly from the parent loop, then it will test again if the word is in the Dictionary AVL Tree it will add it to the similar string. Finally, the method will convert the string using split method to an array of words then return it.

5- public void saveDictionary(String fileName) throws IOException

```
//create file
File file = new File(fileName); //creating new file
//create file writer
FileWriter writer = new FileWriter(file); //creating new file
writer
String[] list = tree.inorderList(); //creating list of words in
dictionary
for (String s : list) { //for each word in the list
    writer.write(s + " "); //write the word to the file
    writer.write(System.lineSeparator()); //write a new line
}
writer.close(); //close the writer
```

This is the last method in our project. It's used to save the dictionary to a new file after you done the operations. In this method, we are using (File and FileWriter). Firstly, we initialize a file with a name of the given String in method header. Then, we create a String[] array by converting the AVL Tree to an array. Using the method inorderList() from AVL Tree class. Then, the a for loop with iterate over all words in the array and write it on the file. Finally there will be writer.close() to close the output file.

- **Created the required Exceptions (WordAlreadyExistsException and WordNotFoundException)**

```
public static class WordNotFoundException extends Exception {
  //exception for word not found
      public WordNotFoundException(String errorMessage) {
  //constructor
          super("Word not found."); //print error message
      }
  }

  public class WordAlreadyExistsException extends Exception {
  //exception for word already existing
      public WordAlreadyExistsException(String errorMessage) {
  //constructor
          super("Word Already Exists."); //print error message
      }
```

- **Finally created the Driver(main) class to test our dictionary.**

```
public class Driver {
    public static void main(String[] args) {
        try {
            // ask to enter file name
            System.out.print("Enter filename> "); //prompt user for file name
            Scanner scanner1 = new Scanner(System.in); //scanner for user
input
            String filename = "src\\" + scanner1.nextLine(); //filename is
the user input
            Dictionary dictionary = new Dictionary(new File(filename));
//creating new dictionary
            Scanner scanner = new Scanner(System.in); //creating new scanner
            System.out.println("Welcome to the dictionary!"); //welcome
message
            System.out.println("");
            System.out.print("check word> "); //prompting user for input
            String s = scanner.next(); //getting user input
            dictionary.findWord(s); //finding word in dictionary
            System.out.println("");
            System.out.print("add new word>"); //prompting user for input
            s = scanner.next(); //getting user input
            try {
            dictionary.addWord(s); //adding word to dictionary
            } catch (Dictionary.WordAlreadyExistsException e) {
                System.out.println("Word already exists."); //print error
message
            }
            System.out.println("");
            System.out.print("remove word> "); //prompting user for input
            s = scanner.next(); //getting user input
            try {
            dictionary.deleteWord(s); //deleting word from dictionary
            } catch (Dictionary.WordNotFoundException e) {
                System.out.println("Word not found.");; //print error message
            }
            System.out.println("");
            System.out.print("Search for similar words> "); //prompting user
for input
            s = scanner.next(); //getting user input
            String[] similar = dictionary.findSimilar(s); //finding similar
words
            if(similar.length==0){ //if there are no similar words
                System.out.println("No similar words found!"); //print
message
            } else { //if there are similar words
                System.out.println("Similar words: " +
Arrays.toString(similar)); //print similar words
            }
            System.out.println("");
            System.out.print("Save Updated Dictionary (Y/N)> "); //prompting
user for input
            s = scanner.next(); //getting user input
```

```
            if (s.equals("Y")) { //if the user wants to save the dictionary
                //ask to enter file name
                System.out.print("Enter filename> ");
                String filename2 = "src\\" + scanner1.nextLine();
                dictionary.saveDictionary(filename2); //save the dictionary
                System.out.println("Dictionary saved successfully!"); //print
success message
            } else { //if the user does not want to save the dictionary
                System.out.println("Dictionary not saved!"); //print message
            }
            System.out.println("Thank you for using our Dictionary");
//goodbye message
            scanner.close(); //closing scanner
        } catch (FileNotFoundException e) { //catching file not found
exception
            System.out.println("File not found!"); //print error message
        } catch (IOException e) { //catching io exception
            e.printStackTrace(); //print stack trace
        }

    }
}
//END OF LAB PROJECT
```

- **The Output of the Driver(main) Class:**

```
Enter filename> mydictionary.txt
Loading dictionary...
Welcome to the dictionary!

check word> puinter
Word not found.

add new word>punter
Word added successfully!

remove word> puinter
Word not found.

Search for similar words> puinter
Similar words: [painter, pointer, printer, punter]

Save Updated Dictionary (Y/N)> Y
Enter filename> mydictionary2.txt
Dictionary saved successfully!
Thank you for using our Dictionary

Process finished with exit code 0
```

- **Dictionary Full Code:**

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

public class Dictionary extends AVLTree {
    private String s;
    private int size;
    private AVLTree<String> tree;

    public Dictionary(String s) { //constructor
        this.s = s;
        tree.insert(s);
        size = 0;
    }

    public Dictionary(File f) throws FileNotFoundException {
//constructor
        System.out.println("Loading dictionary..."); //loading
dictionary
        tree = new AVLTree<>(); //creating new tree
        Scanner scanner = new Scanner(f); //scanning file
        size = 0; //size of dictionary
        while (scanner.hasNextLine()) { //while there is a next line
            String s = scanner.nextLine(); //s is the next line
            tree.insertAVL(s); //inserting the word into the tree
            size++; //incrementing size
        }
        scanner.close(); //closing scanner
    }

    public Dictionary() { //constructor
        tree = new AVLTree<>(); //creating new tree
        size = 0; //size of dictionary
    }

    public void addWord(String s) throws WordAlreadyExistsException
{ //adding word to dictionary
        if (tree.isInTree(s)) { //if the word is already in the tree
            throw new WordAlreadyExistsException("Word already
exists!"); //throw exception
        } else { //if the word is not in the tree
            tree.insert(s); //insert the word into the tree
            System.out.println("Word added successfully!"); //print
success message
        }
    }

    public boolean findWord(String s) { //finding word in dictionary
        if (tree.isInTree(s)) { //if the word is in the tree
            System.out.println("Word found!"); //print success
message
```

```
            return true; //return true
        } else { //if the word is not in the tree
            System.out.println("Word not found!");  //print error
message
            return false; //return false
        }
    }

    public void deleteWord(String s) throws WordNotFoundException {
//deleting word from dictionary
        if (tree.isInTree(s)) { //if the word is in the tree
            tree.deleteAVL(s); //delete the word from the tree
            System.out.println("Word deleted successfully!");
//print success message
        } else { //if the word is not in the tree
            throw new WordNotFoundException("Word not found!");
//throw exception
        }
    }

    public String[] findSimilar(String s){ //finding similar words
        String similar =""; //creating new array of similar words
        for(int i = 0; i < s.length(); i++){ //for each letter in
the word
            String temp = s.substring(0, i) + "" + s.substring(i +
1); //temp is the word without the letter
            if (tree.isInTree(temp)){ //if the word is in the tree
                // check if duplicate don't add it
                if(!similar.contains(temp)){
                    //add the word to the string similar
                    similar += temp + " ";}

            }
            for(int j = 0; j < 26; j++){ //for each letter in the
alphabet
                char c = (char) (j + 'a'); //c is the letter
                temp = s.substring(0, i) + c + s.substring(i + 1);
//temp is the word with the letter added
                if (tree.isInTree(temp)){ //if the word is in the
tree
                    // check if duplicate don't add it
                    if(!similar.contains(temp)){
                        //add the word to the string similar
                        similar += temp + " ";
                    }
                }
            }
        }
        String[] similarWords = similar.split(" "); //splitting the
string into an array
        return similarWords; //return the array of similar words
    }
```

```
    public void saveDictionary(String fileName) throws IOException {
//saving dictionary to file
        //create file
        File file = new File(fileName); //creating new file
        //create file writer
        FileWriter writer = new FileWriter(file); //creating new
file writer
        String[] list = tree.inorderList(); //creating list of words
in dictionary
        for (String s : list) { //for each word in the list
            writer.write(s + " "); //write the word to the file
            writer.write(System.lineSeparator()); //write a new line
        }
        writer.close(); //close the writer
    }

    public static class WordNotFoundException extends Exception {
//exception for word not found
        public WordNotFoundException(String errorMessage) {
//constructor
            super("Word not found."); //print error message
        }
    }

    public class WordAlreadyExistsException extends Exception {
//exception for word already existing
        public WordAlreadyExistsException(String errorMessage) {
//constructor
            super("Word Already Exists."); //print error message
        }
    }
}
```