**131034\_130138\_131048\_DSA\_Report**

1. **Title of the project:**

* Spell Checker

1. **Team Members:**

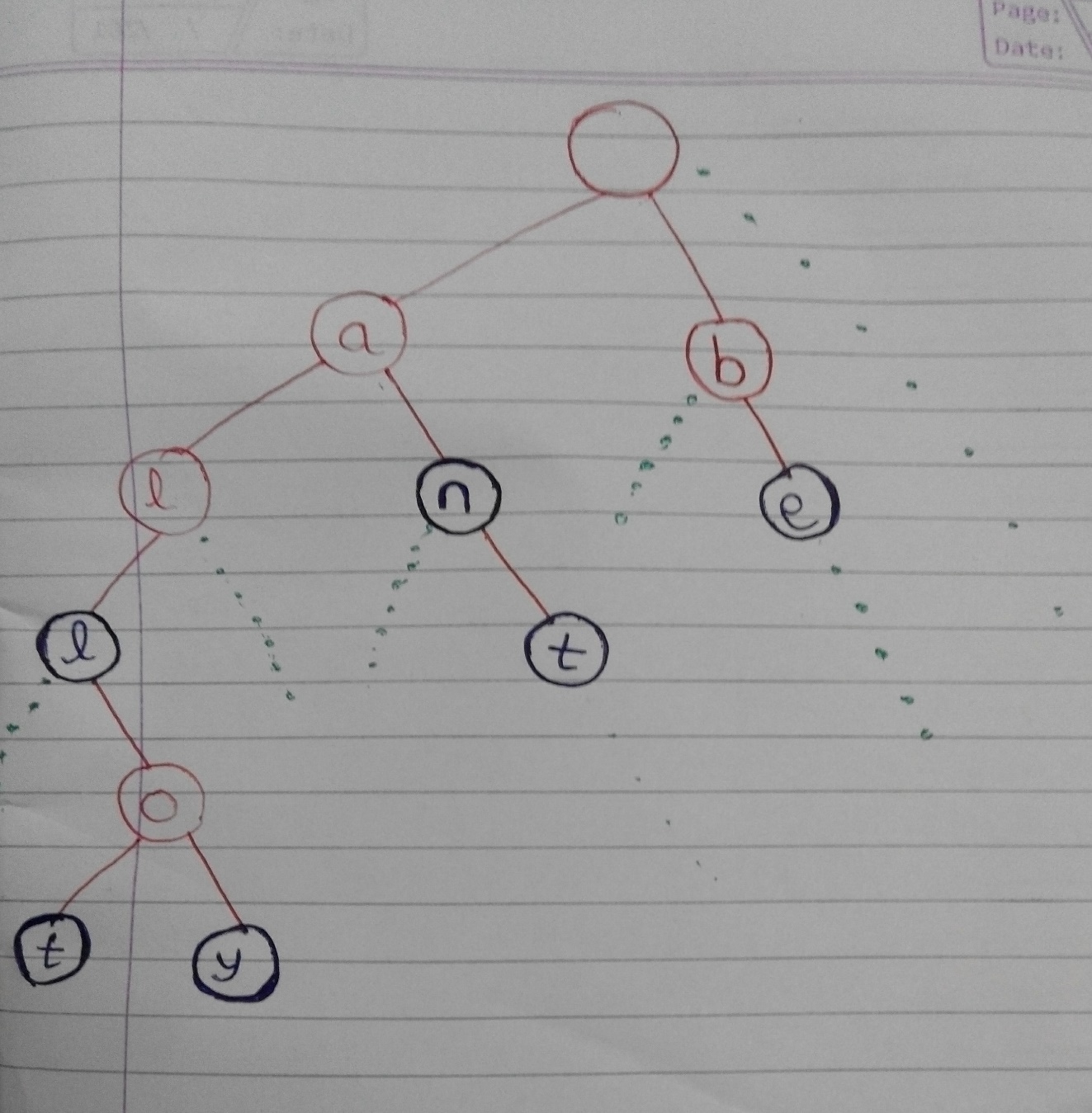
* Pranali Raval\_131034
* Rachana Solanki\_131038
* Shaily Mishra\_131048

1. **Brief description:**

* The project implements a simple Spell Checker for English language.
* On execution, first the database of nearly 5000 words is read from the file “DataBase.txt” and a trie tree is created out of that database.
* Then, user is asked to enter the text which he wants to be checked, to the console.
* As the user enters the text, that text is stored in an array and the punctuation marks are removed.
* Elements of that array are the words of input text.
* The words are then converted to the lower case as the words in database are of lower case.
* Then, each word is checked for its occurrence in database.
* If every word of the text is in database, the program asks user if he wants to save the input text in a form of text file.
* If user wants to save the text, he is asked for the file name by which he wants to store it and upon giving it, text is saved and a completion message appears in console.
* After that, if user wants to enter another text, he can enter it or terminate the program by giving required inputs.
* Now, if any of the word of the text is not in database, that word is indicated to user and he is asked if he wants to add that word to database or wants to look for suggestions for that particular word from the database.
* If user selects to add the word to database, word is added to database.
* If user selects to look for suggestions, he is provided with a list of suggestions for that particular word from the database and the program asks if he wants to replace the word with any of the suggestions.
* If user selects to replace the word, he is asked to write the word that the wrong word is to be replaced with.
* On giving that, the word is now replaced with the new word provided.
* At last, he is asked if he wants to save the file or not and after that he wants to continue execution or not.

1. **List of data structures and diagrams:**

* Trie Tree
* Diagram:



* The diagram shows the basic generalized structure of Trie tree.
* The root node is empty and has links to 26 children as 26 alphabets.
* Each node contains an alphabet and link to other 26 nodes.
* When a word gets completed, the node is shown with black color.
* Green dots show links to other children.

1. **List of Operations to be performed on data structure:**

* Insertion of words to the Trie as well as to the database.
* Updation of Trie and databse when user adds a new word to database.
* Print the values when suggestions have to be provided.
* Search for specific elements based on the input words user provides to check if the word is in database or not and on the time of giving suggestions.
* Requirements of processing on data structure:
* Save Trie to the database which is a text file and regenerate Trie from the same file.

1. **Pseudo code for each logical process:**

* **Create Trie:**

Algorithm Create\_Tree()

1. return new empty TrieNode

2. end Create\_Tree

* **Insert words to Trie:**

Algorithm Insert\_Word (Root,Word)

1. TrieNode Current := Root
2. Length := Word.length
3. Letters[] := Word.toCharArray
4. for i := 0 to Length-1 do
5. Position := Letters[i] – 97
6. if ( Current.links[Position] == null )
   * + 1. Current.link[Position] := new TrieNode ( Letters[i] )
7. end if
8. Current := Current.links[Position]
9. end for
10. end Insert\_Word

* **Find words in Trie:**

Algorithm Find\_Word (Root,Word)

1. TrieNode Current := Root
2. Letters[] := Word.toCharArray
3. Length := Letters.length
4. for i:= 0 to Length-1 do
5. if(Current = null)

1. return false

2. Current := Current.links[Letters[i]-97]

5. end for

6. if(i = Length & Current = null)

1. return false

7. end if

8. if(Current != null & Current.FullWord = false)

1. return false

9. end if

10. else

1. return true

11. end Find\_Word

* **Suggest words:**

Algorithm Suggest\_Words (Root,Word)

1. TrieNode Current := Root
2. Letters[] := Word.toCharArray
3. Length := Letters.length
4. for i:= 0 to Length-1 do

1. if(Current != null) then

1. Current := Current.links[Letters[i]-97]

2. end if

5. end for

6. if (Current = null)

1. print(“No suggestion”)

7. end if

8. else

1. print(Word)

2. print subtree with Root as Current

9. end Suggest\_Words

* **Add new word to Trie and database:**

Algorithm Add\_Word(Word)

1. Length := Word.length

2. for i:= 0 to Length-1 do

1. Word[i].WriteToFile

2. print(“Word added to database”)

3. InsertWord(Root, Word[i])

3. end Add\_Word

1. **List of Programs:**

* **TrieNode.java:**

//Class : Trie Node - Node of the trie tree

**public** **class** TrieNode

{

// Data field

**char** letter; //letter to be stored in TrieNode

TrieNode[] next; // links to next 26 nodes

**boolean** fullWord; // fullword field

// Constructor of trie node

TrieNode(**char** letter )

{

**this**.letter = letter;

next = **new** TrieNode[26];

fullWord = **false**; // initial will be false because it will be true only when a word is completed at that node

}

}

* **TrieTree.java:**

// Class : Trie Tree

// It contains method such as creating tree, insertion , searching , suggesting word, printing

**public** **class** TrieTree

{

///Create tree method - creates root node of the tree///////////////////

TrieNode createTree()

{

**return**(**new** TrieNode('\0'));

}

//////Insert Word method - inserts a word in trie tree/////////////////////

**void** insertWord(TrieNode root, String word)

{

**int** l = word.length();

**char**[] letters = word.toCharArray();

TrieNode curNode = root;

**for** (**int** i = 0; i < l; i++)

{

// Adds all letters of word

**int** position = letters[i]-97;

**if** (curNode.next[position] == **null**)

curNode.next[position] = **new** TrieNode(letters[i]);

curNode = curNode.next[position];

}

//After all letters are added, node containing last letter is a fullword node

curNode.fullWord=**true**;

}

// Find method - finds if a word if there in a trie tree or not///////////////////

**boolean** find(TrieNode root, String word)

{

**char**[] letters = word.toCharArray();

**int** l = letters.length;

TrieNode curNode = root;

**int** i;

// Checks if the first (l-1) nodes are present or not

**for** (i = 0; i < l; i++)

{

**if** (curNode == **null**) // If not there returns false

**return** **false**;

curNode = curNode.next[letters[i]-97];

}

// Checks if the last node i.e. lth node is there or not

**if** (i == l && curNode == **null**) //If not there returns false

**return** **false**;

//if all the nodes are present but the last node is not a full word field then return false

**if** (curNode != **null** && !curNode.fullWord)

**return** **false**;

**return** **true**;

}

////////////// printTree method - to print all the words in tree//////////////////

**void** printTree(TrieNode root, **int** level, **char**[] branch)

{

// Traverses through all the nodes

// Prints all the words

**if** (root == **null**)

**return**;

**for** (**int** i = 0; i < root.next.length; i++)

{

branch[level] = root.letter;

printTree(root.next[i], level+1, branch);

}

**if** (root.fullWord)

{

**for** (**int** j = 1; j <= level; j++)

System.***out***.print(branch[j]);

System.***out***.println();

}

}

////////////// printSuggest - to print suggestion of word///////////////////////////////

**void** printSuggest(TrieNode root, **int** level, **char**[] branch, String searchword)

{

**if** (root == **null**)

**return**;

**for** (**int** i = 0; i < root.next.length; i++)

{

branch[level] = root.letter;

printSuggest(root.next[i], level+1, branch, searchword);

}

**if** (root.fullWord)

{

System.***out***.print(searchword);

**for** (**int** j = 1; j <= level; j++)

System.***out***.print(branch[j]);

System.***out***.println();

}

}

/////////////// suggest - to give suggestion//////////////////////////////////////

**void** suggest ( TrieNode root , String word)

{

**char**[] letters = word.toCharArray();

**int** l = letters.length;

TrieNode curNode = root;

// if the prefix of word is there in trie tree then only gives suggestion

// words with the same prefix are printed

**int** i=0;

**for** ( i=0 ; i<l ; i++)

{

**if**(curNode!=**null**)

curNode = curNode.next[letters[i]-97];

}

**if** (curNode == **null**)

System.***out***.println( " No suggestion for : " + word);

**else**

{

System.***out***.println(" Suggested words for : " + word );

**char**[] branch = **new** **char**[50];

printSuggest( curNode , 0 , branch, word );

}

}

}

* **Check.java**

import java.io.\*;

import java.util.\*;

// Class : Check - Contains methods that prompt user to enter a text, checks it for spelling and lets user save the text in a file

public class Check

{

//to count no of lines user entered

static int count=0;

static String[] line = new String[500];

static Scanner myinput = new Scanner(System.in);

/////// AddToDataBase method - lets user add a new word to database

public static void AddToDataBase(String word) throws IOException

{

WriteFile file2 = new WriteFile("DataBase.txt",true);

file2.WriteToFile(word);

System.out.println("'" + word + "' added in database");

}

///////Save file method - To save text entered by user in a file/////

public static void SaveFile(String[] word) throws IOException

{

// Takes file name from users

System.out.println(" Enter file name");

String filename = myinput.next();

filename = filename +".txt";

WriteFile file3 = new WriteFile(filename,true);

// Saving text in that file entered by user

for (int m=0 ; m<=count; m++)

file3.WriteToFile(word[m]);

System.out.println("done! file saved");

}

///////CheckText method - To check the input text/////

public void CheckText()

{

TrieTree tree = new TrieTree();

TrieNode rootNode = tree.createTree();

String[] searchword = new String[20];

String[] words;

String choice;

try

{

//Reading database

Read file1 = new Read("DataBase.txt");

words = file1.OpenFile();

// Creating trie tree out of database

for (int i = 0; i < words.length; i++)

{

tree.insertWord(rootNode, words[i]);

}

// User starts entering the text

System.out.println("Enter the text ");

Scanner input = new Scanner(System.in);

line[count] = input.nextLine();

//Searchword contains words of the text line after removing punctuations

searchword = line[count].replace(".", "").replace(",", "").replace(")", "").replace("?", "").replace("!","").replace(":", "").replace("(", "").split(" ");

// Making spell checker independent of case sensitivity

for(int c=0; c<searchword.length; c++)

{

searchword[c] = searchword[c].toLowerCase();

}

// Spell Checking

for(int i=0 ; i<searchword.length ; i++)

{

// Checking if each word of the text line is there in the database

if (!tree.find(rootNode, searchword[i])) // IF NOT THERE

{

System.out.println("'" + searchword[i] + "' is not in database");

// 3 Options given - Add that word to database , Suggest words for that word , and Ignore and write next line

System.out.println("Enter 'add' to add '" + searchword[i] + "' to database and enter 'suggest' to look for suggestions for '" + searchword[i] + "' or enter 'ignore' ");

choice = myinput.next();

// If the User selects add - adds word to database and trie tree

if(choice.equalsIgnoreCase("add"))

{

AddToDataBase(searchword[i]);

tree.insertWord(rootNode,searchword[i]);

}

//If the User selects suggest - suggest words

// Then 2 Options given - replace or not

if (choice.equalsIgnoreCase("suggest"))

{

// Checks suggestion for that word

tree.suggest (rootNode , searchword[i]);

//To replace that word

System.out.println("Enter 'replace' to replace '" + searchword[i] + "' or enter 'not' if you dont want to replace");

choice = myinput.next();

if(choice.equalsIgnoreCase("replace"))

{

//Enter word to replace this word

System.out.println("Enter word to replace '" + searchword[i] + "' with:");

String replace = myinput.next();

//Replacing it line[count] = line[count].replace(searchword[i], replace);

System.out.println("'" + searchword[i] + "' is replaced ");

}

}

}

}

//2 Options to write further or to exit and save file

System.out.println("Enter 1 to write another text and 0 to save file and exit");

int l = myinput.nextInt();

if(l==1)

{

count++;

//Writing another line

CheckText();

}

else

{

SaveFile(line);

System.exit(0);

}

}

catch(IOException e){

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

}

}

}

* **WriteFile.java**

**import** java.io.\*;

// Class to write text in a file

**public** **class** WriteFile

{

//////////////// data fields //////////////////////

**private** String path;

**private** **boolean** append = **false**;

//////////// Constructor /////////////////////////////////

**public** WriteFile (String Path)

{

path = Path;

}

**public** WriteFile (String file\_path , **boolean** append\_value)

{

path = file\_path;

append = append\_value;

}

///////////// Method to write text in file////////////////

**public** **void** WriteToFile (String text) **throws** IOException

{

FileWriter write = **new** FileWriter (path, append);

PrintWriter print\_line = **new** PrintWriter (write);

print\_line.printf("%s"+ "%n" , text);

print\_line.close();

}

}

* **Read.java**

**import** java.io.\*;

// Class : Read - to read words from a file

**public** **class** Read

{

//////////////// data fields //////////////////////

**private** String path;

//////////// Constructor /////////////////////////////////

**public** Read (String Path)

{

path = Path;

}

////// Method - to count number of lines in a file//////////////

**int** Lines() **throws** IOException

{

FileReader fr = **new** FileReader (path);

BufferedReader bf = **new** BufferedReader(fr);

String Line;

**int** Lines=0;

// counts until the end of file

**while** ((Line = bf.readLine() ) != **null** ) Lines++;

bf.close();

**return** Lines;

}

////////////////////////////////////////////////////////////////////////////

/////////// Method - to read from a file and return the data////////////

**public** String[] OpenFile() **throws** IOException

{

FileReader f = **new** FileReader (path);

BufferedReader b = **new** BufferedReader(f);

**int** Lines = Lines();

String[] Data = **new** String[Lines];

**int** i;

**for**( i=0; i<Lines; i++ )

{

// Reads line from file and stores in an array

Data[i] = b.readLine();

}

b.close();

**return** Data;

}

}

* **Main.java**

**import** java.io.\*;

//Class : Main - contains main method

**public** **class** Main

{

**public** **static** **void** main(String[] args) **throws** IOException

{

//////////////// Creating objects of Check class //////////////////////

Check c = **new** Check();

c.CheckText(); //Calling CheckText method

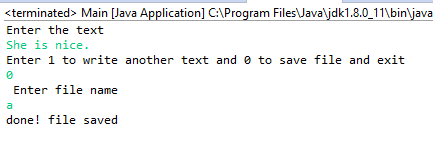
}

}

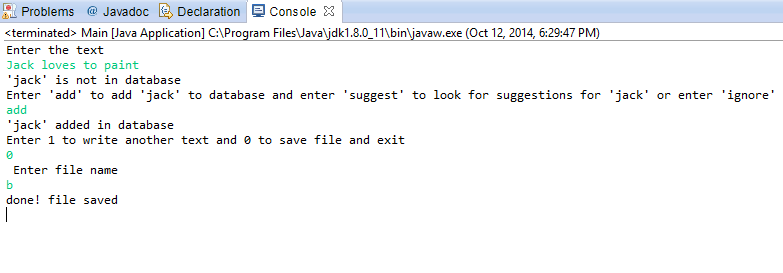
* **Instructions to compile and execute the program:**
* Execute the file “Main.java”.
* **Input Data**
* Any text
* ‘add’ if user wants to add a word to database and trie after checking the input text and some word doesn’t match exactly with any word in database.
* ‘suggest’ if user wants to see for suggestions list after checking the input text and some word doesn’t completely match with any word in database.
* ‘replace’ if user wants to replace the word of input text by one of the suggestions.
* 1 to write another text after a single cycle
* 0 to save the input text in a file
* Filename by which the file containing input text is to be saved.
* **Outputs generated:**
* Prompts to add, suggest or replace words based on user input text and command to save the user input text in a file.

1. **Test Results:**

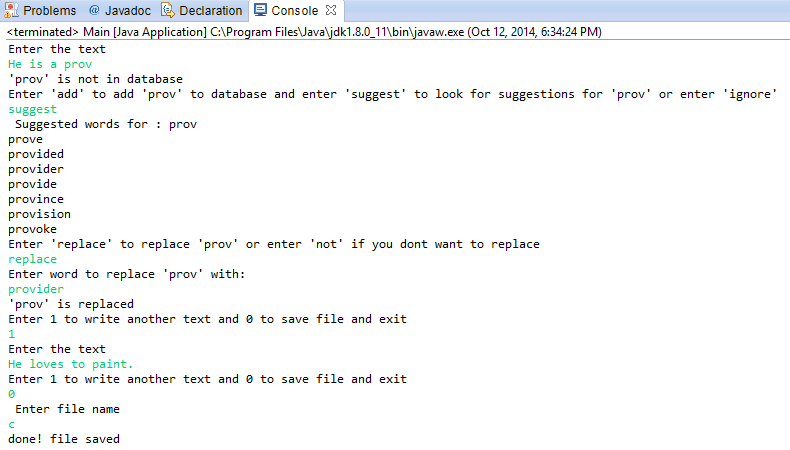
* **Snapshots:**
* Scenario 1: All the words from the user input text are present in database.



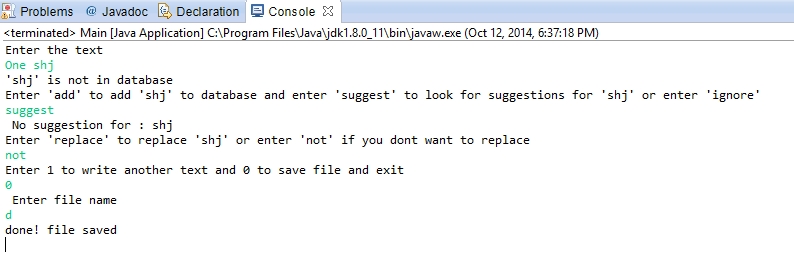
* Scenario 2: Some words from the user input text are not present in database and user wants to add those words to DataBase.



* Scenario 3: Some words from user input text are not present in database and user wants to look for suggestions for those words and replace them with one of the suggestions.



* Scenario 4: User wants to look for suggestions but there is no suggestion for that word.



1. **Additional assignments completed:**

* Save File:
* Source code:

///////Save file method - To save text entered by user in a file/////

public static void SaveFile(String[] word) throws IOException

{

// Takes file name from users

System.out.println(" Enter file name");

String filename = myinput.next();

filename = filename +".txt";

WriteFile file3 = new WriteFile(filename,true);

// Saving text in that file entered by user

for (int m=0 ; m<=count; m++)

file3.WriteToFile(word[m]);

System.out.println("done! file saved");

}

* Brief:

The program checks the spelling of the text entered by user. But along with that, user can also save the entered text in the form of a text file. The above method takes the file name by which he wants to save the file, creates a text file by that name and writes the user input text in that text file.

1. **References:**

* TrieTree.java - <http://exceptional-code.blogspot.in/2011/07/coding-up-trie-prefix-tree.html>
* Read.java and WriteFile.java – <http://www.homeandlearn.co.uk/java>
* DataBase (word list) – <http://www.englishclub.com/vocabulary/common-words-5000.htm>