

WORD AUTOCORRECTION & AUTOCOMPLETION SYSTEM

A report submitted for the course named Project II (CS-300)

By

Anupriya Sinha

Bachelor of Technology, VI Semester

Roll No. 16010108

Under the Supervision and Guidance of

Dr. Prerna Mohit

Assistant Professor



Department of Computer Science and Engineering
Indian Institute of Information Technology Senapati,
Manipur
April, 2019

Abstract

In this project, Attempts have been made to develop a word autocorrection & autocompletion system that takes string consisting of english alphabets as query input and depending upon users' choice it either correct or complete the given query string.

Keywords - Hash Table, Edit Distance, Trie

Declaration

I declare that this submission represents my idea in my own words and where others' idea or words have been included, I have adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/sources in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from proper permission has not been taken when needed.

Date: 16th April, 2019

(Anupriya Sinha)
(16010108)



Department of Computer Science & Engineering
Indian Institute of Information Technology Manipur

Dr. Prerna Mohit
Assistant Professor

Email: prerna@iiitmanipur.ac.in

To Whom It May Concern

This is to certify that the report entitled “**WORD AUTOCORRECTION & AUTOCOMPLETION SYSTEM**” submitted by "Anupriya Sinha", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree, diploma or a course.

Signature of Supervisor

(Dr. Prerna Mohit)



Department of Computer Science & Engineering
Indian Institute of Information Technology Manipur

To Whom It May Concern

This is to certify that the report entitled “**WORD AUTOCORRECTION & AUTOCOMPLETION SYSTEM**” submitted by "Anupriya Sinha", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree, diploma or a course.

Signature of HoD

(Dr. Nongmeikapam Kishorjit Singh)

Signature of Examiner 1: _____

Signature of Examiner 2: _____

Signature of Examiner 3: _____

Acknowledgement

I would like to extend my special thanks to my mentor "Dr. Prerna Mohit", Assistant Professor at IIIT Manipur, Department of CSE, for providing such a golden opportunity to do the project "WORD AUTOCORRECTION & AUTO-COMPLETION SYSTEM" under her guidance. I would also like to thank my seniors and friends who helped me whenever I was stuck and share their ideas which ultimately helped me to make my Application better.

- Anupriya Sinha

Contents

Abstract	ii
Declaration	iii
Certificate	iv
Certificate	v
Acknowledgement	vi
Table of contents	vii
List of figures	x
List of abbreviations	xi
1 Introduction	1
2 Software Analysis and Requirement Specification	2
2.1 Introduction	3
2.1.1 Objective	3
2.1.2 Existing System Study	3
2.1.3 Feasibility Study	4
2.1.4 User Classes	5

2.1.5	SDLC Model	6
2.2	System Functionality	6
2.2.1	Word Autocorrection	7
2.2.2	Word Autocompletion	7
3	System Design	8
3.1	Data Flow Diagram	9
3.2	Word Autocorrection	11
3.2.1	Data Structure	11
3.2.2	Algorithm	11
3.3	Word Autocompletion	15
3.3.1	Data Structure	16
3.3.2	Algorithm	16
4	Implementation	18
4.1	Introduction	19
4.2	Resources Used	19
4.3	Modules	19
4.4	Autocorrection Module	20
4.5	Autocompletion Module	21
5	Result Analysis and Testing	23
5.1	Introduction	24
5.2	Result and testing	24
5.3	Summary	26
6	Conclusion	27
6.1	Limitations	28

6.2	Future Direction	28
Appendix A Screenshot and Description of the Implemented System		29
A.1	Choices to users	29
A.2	Autocorrection	29
A.3	Autocompletion	29
Appendix B User manual		31
B.1	Introduction	31
B.2	Step to install your implemented system	31
B.3	Code Snippet	32
B.3.1	Main Thread class	32
B.4	Classe involved in autocorrection	33
B.5	Classes involved in autocorrection	34
B.5.1	Class to define each node	34
B.5.2	34

List of Figures

2.1	Input and Output	4
2.2	Gantt Chart	5
2.3	Input and Output	6
3.1	DFD level 0	9
3.2	DFD level 1	10
3.3	Separate Chaining: Collision Resolution Techniques	12
3.4	Recursive tree showing edit distance computation	14
3.5	Trie Structure	17
4.1	Hash Table	20
4.2	Computing Edit Distance	21
4.3	Trie Representation	21
5.1	Autocorrection Example1	24
5.2	Autocorrection Example2	25
5.3	Autocompletion Example1	25
5.4	Autocompletion Example2	26
A.1	Choices to users	29
A.2	Autocorrection Example	30
A.3	Autocompletion Example	30

List of abbreviations

		A
ASCII	American Standard Code for Information Interchange	
		D
DFD	Data-flow diagram	
		D
SDLC	Software Development Life Cycle	

Chapter 1

Introduction

Every language in the world has lakhs and lakhs of words. The English language consists of around 1,71,000 words, It is quite difficult for anyone to remember these words with their spellings. Motivated by this fact, and to make the people's life easy, the idea is to make a "Word Autocorrection & Autocompletion system" that takes a single word as input and in output, it either gives the suggestion of correct spelling or complete word for given query string as per users' choice.

The input to the proposed system is any string consisting of English alphabet. This system is composed of broadly two modules one for word autocorrection and another one for word autocompletion.

One of the two modules implements the autocorrection feature. Word autocorrection refers to automatically suggest corrections for the misspelled words. It takes English word, checks if it is correct or not; If it is found to be wrong then it makes the suggestion of possible correct spellings or words. The second module implements the autocompletion feature. It takes English word as input gives all the words with the given word as the prefix as output.

Chapter 2

Software Analysis and Requirement Specification

Outline: This chapter presents the following:

- Introduction: It includes objective, Existing System Study, feasibility study, SDLC Model
- System functionality

The purpose of this document is to create a word Autocompletion and Autocorrection System. Word autocorrection refers to automatically suggest corrections for the misspelled words and word autocompletion refers to the feature in which a system or software predicts the rest of the word a user is typing itself using some particular algorithms. This document includes the Introduction of system, the system's functional.

2.1 Introduction

This section includes the purpose of creating the proposed system, the study of the existing system, feasibility, intended user classes and scope of the system.

2.1.1 Objective

The idea is to make a system that takes a single word as input and give one out of following two output depending upon users choice.

- Check the spelling of word and if it wrong then it will suggest the possible correct spelling.
- Word autocompletion.

The overview of input to the system and output of the system is shown in figure [2.1](#).

2.1.2 Existing System Study

During Information gathering and existing system study, It has been found that the following software have the word autocorrection and autocompletion features.

- Google keyboard
- Grammarly
- MS Word

Google keyboard

Google keyboard, commonly known as Gboard, is a virtual keyboard app developed by Google for Android and iOS devices.

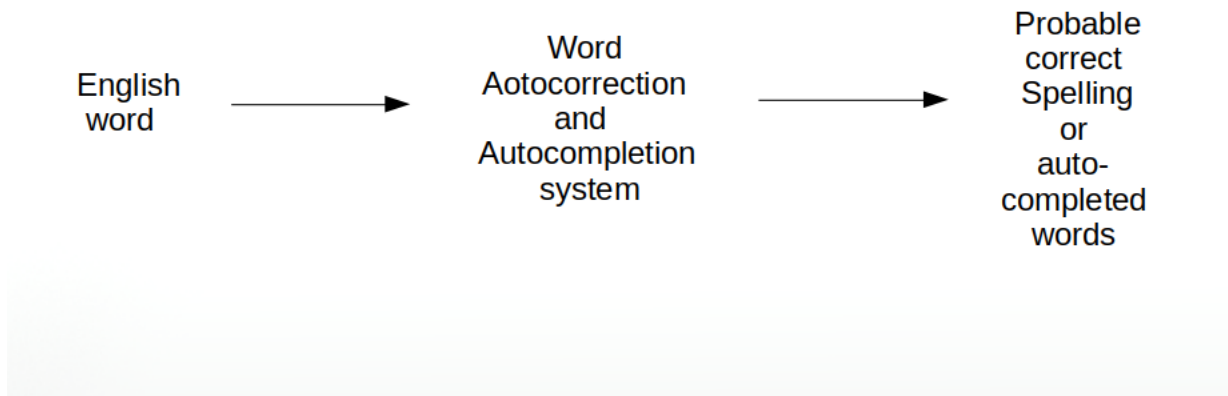


Figure 2.1: Input and Output

Grammarly

Grammarly is an online grammar checking, spell checking, and plagiarism detection platform developed by Grammarly, Inc. Grammarly's algorithms flag potential issues in the text and suggest context-specific corrections for spelling, grammar and have many more features. Grammarly algorithms are based on artificial intelligence and Natural Language Processing. Since Grammarly is not an open source, Its algorithms are not known, but they are known to use artificial intelligence.

2.1.3 Feasibility Study

This section discusses the technical, economic and time feasibility study of proposed system.

Technical Feasibility

Resources used to implement the proposed system are as follows:

- Ubuntu 18.04 OS
- Eclipse IDE: For Implementation of algorithms and GUI design using java EE.
- Dictionary

Ubuntu is a free and open-source Linux distribution based on Debain. Ubuntu aims to be secure by default. User programs run with low privilages and cannot corrupt the

operating system or other users’ file. It is a popular operating system for cloud computing, with support “for OpenStack”.

Eclipse is an integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python. The Eclipse platform can be used to develop rich client applications, integrated development environments and other tools. Eclipse can be used as an IDE for any programming language for which a plug-in is available.

Economic Feasibility

Resorces used to develop the word autocorrection and autocompletion system are open source. Therefore, Implementation of proposed system is economically feasible.

Time Feasibility

The overview of project schedule is shown in figure 2.2, which shows that implementation of proposed system is completed in given time.

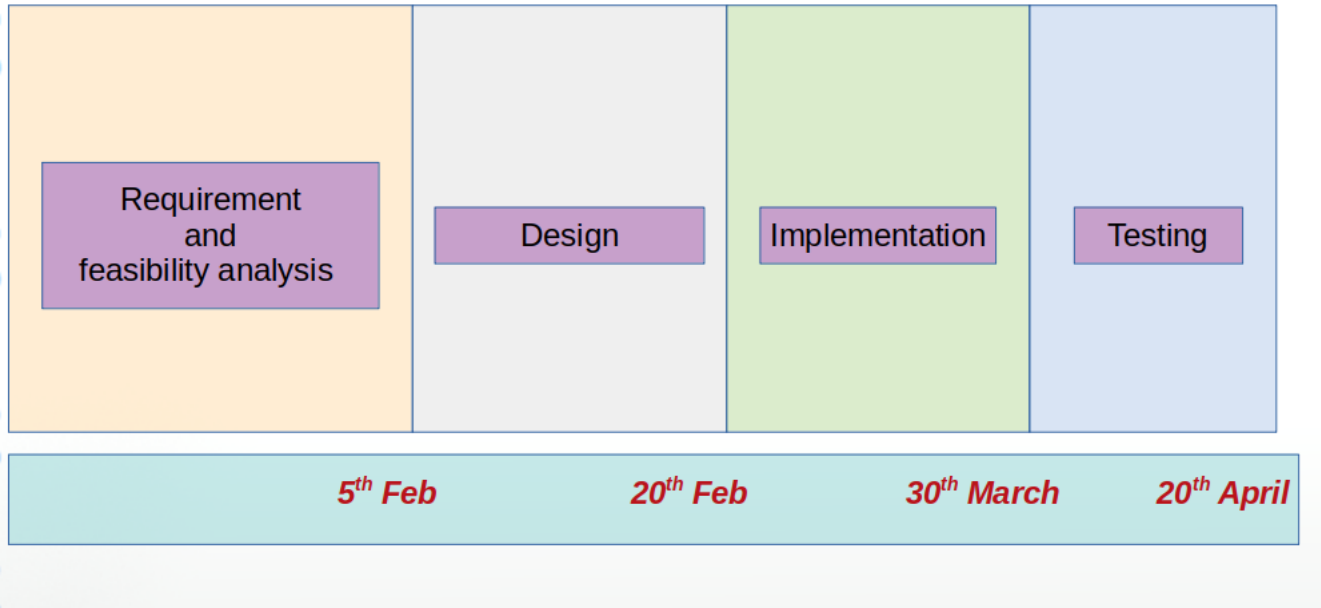


Figure 2.2: Gantt Chart

2.1.4 User Classes

Everyone: Anyone can use this application ranging from children to adult.

2.1.5 SDLC Model

Waterfall SDLC model has been used to implement the proposed system since it is one man project, the algorithm is known and the input and output for are well defined. V model has not been used as it requires lots of time for testing which is not possible in man project with limited time. The prototype model is preferred for GUI based software which is not the case here, that's why it is not used.

2.2 System Functionality

The proposed system broadly has two functionality. It takes English word as input and gives one of the following two output discussed below depending upon users choice.

- If the user chooses to check the spelling of the given word and if the spelling of the given word is wrong then the system will suggest the possible correct spellings.
- If the user choosees to autocomplete the given word then the system will give all the words with the given word as a prefix.

The overview of system with its input and output can be represented as shown in figure below.

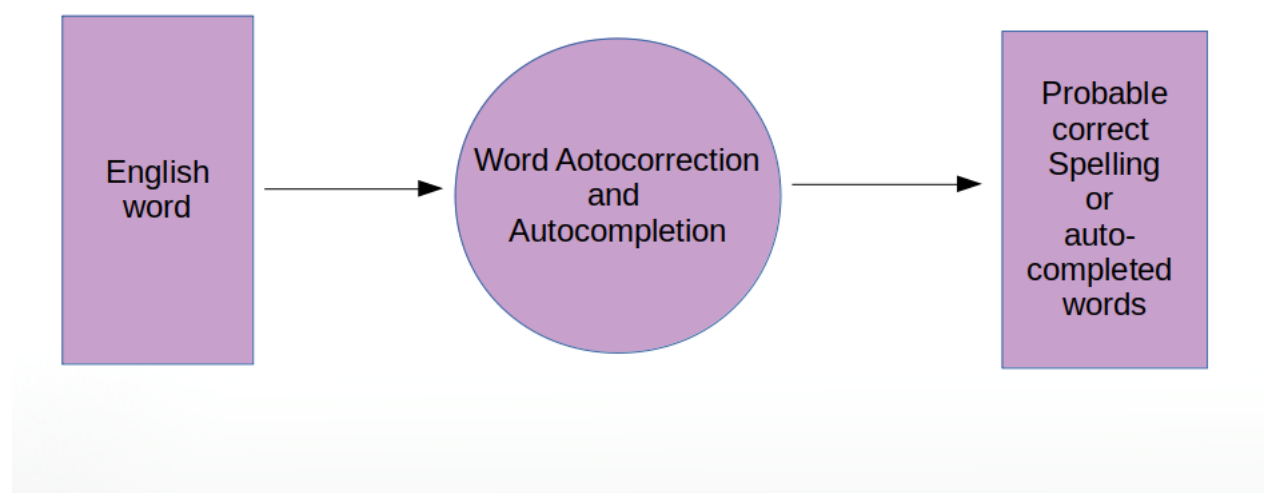


Figure 2.3: Input and Output

2.2.1 Word Autocorrection

Word autocorrection refers to automatically suggest corrections for the misspelled words. The proposed system has the feature of word autocorrection. It takes English word, checks if it is correct or not; If it is found to be wrong then it makes the suggestion of possible correct spellings or words.

2.2.2 Word Autocompletion

Word autocompletion refers to the feature in which a system or software predicts the rest of the word a user is typing itself using some particular algorithms. The proposed system has the feature of word autocorrection. It takes English word as input gives all the words with the given word as the prefix in output.

Chapter 3

System Design

Outline: This chapter presents the following:

- Data Flow Diagram
- A brief explanation of word autocorrection design
- A brief explanation of word autocompletion design

This chapter deals with the design part of the proposed word autocorrection and autocompletion system. The data flow diagram, data structures, and algorithms have been discussed in this chapter in detail.

3.1 Data Flow Diagram

A data-flow diagram is a way of representing a flow of data of a process or a system. The data-flow diagram also provides information about the outputs and inputs of each entity and the process itself. The data-flow diagram does not contain control flow, decision rules, and loop. The flow of a data of the proposed system has been discussed below using two DFD levels, that is DFD level 0 and DFD level 1.

DFD level 0

At DFD level 0, the whole system can be seen as a black box that takes English word as input and in output, it either suggests correct spelling or word for the given misspelled word or predict the word with the given word as a prefix. The DFD level 0 diagram of the proposed system can be depicted as shown in figure:

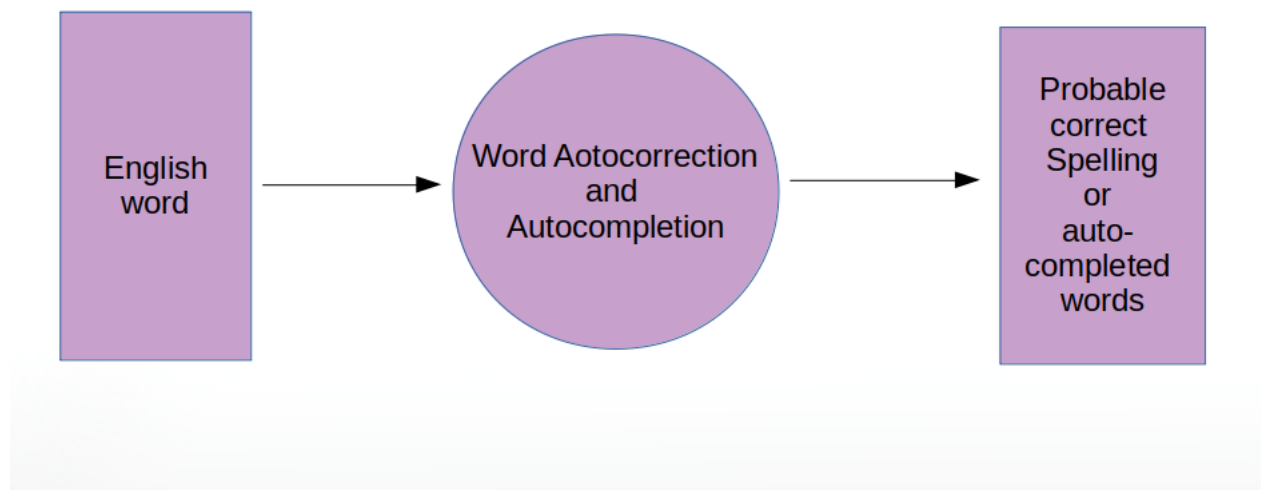


Figure 3.1: DFD level 0

It can be seen in the diagram that input to the system is an English string and output is either probable correct words or spellings or autocompleted word starting with given string.

DFD level 1

At DFD level 1, the whole system is broadly divided into two modules. One module implements the word autocorrection and other module implements the word autocorrection.

It can be seen in the above diagram that word autocorrection module takes a dictionary as input, stores it in the data structure and apply particular algorithm to compute the correct spellings or words for given misspelled word. Similarly, the word autocorrection module also takes a dictionary as input, stores it in the data structure and apply a particular algorithm to compute all the possible complete words.

The details of data structures and algorithms used in two modules are discussed in the next two sections.

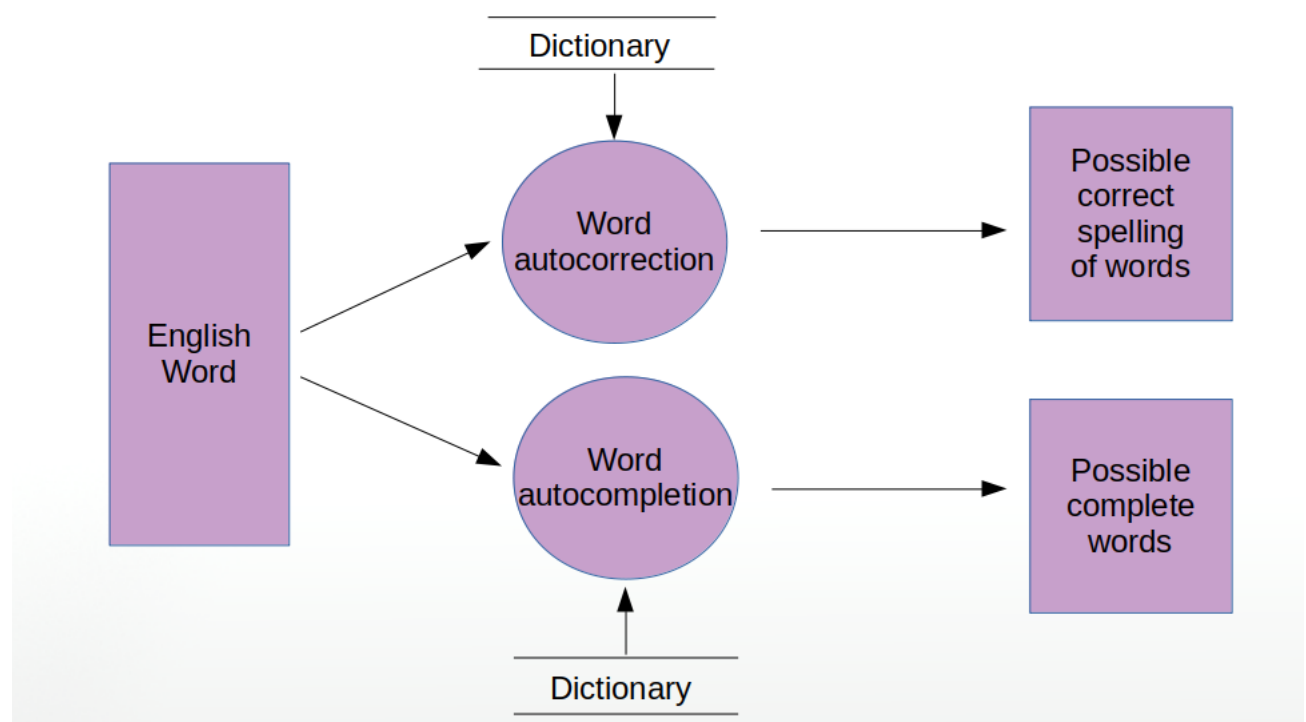


Figure 3.2: DFD level 1

3.2 Word Autocorrection

Word autocorrection refers to automatically suggest corrections for the misspelled words. The word autocorrection module consists of mainly two steps. In the first step, it takes an English dictionary containing around 55,000 words and stores the words in “Hash Table” to decrease the access time. In the second step, it applies the “Edit Distance” algorithm, a dynamic programming technique to give correct spellings or words as output.

3.2.1 Data Structure

Hash table has been used to store the words to decrease the access time of word. Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format and each key maps to the values. A hash table uses a hash function to compute an index for the given key into an array of buckets or slots at which the value can be stored or from which value can be found.

To store words in word autocorrection module, a hash table with 26 buckets or slot has been used. Each slot contains around 2,000 to 2,500 words. The first slot contains all the words starting with “a”, the second slot contains all the words starting with “b” and so on till the twenty-sixth slot. Separate chaining with the linked list, collision resolution technique has been used to overcome collision.

The cost of a table operation is that of scanning the entries of the selected bucket for the desired key. If the distribution of keys is sufficiently uniform, the average cost of a lookup depends only on the average number of keys per bucket—that is, it is roughly proportional to the load factor. For separate-chaining, the worst-case scenario is when all entries are inserted into the same bucket, in which case the hash table is ineffective and the cost is that of searching the bucket data structure. If the latter is a linear list, the lookup procedure may have to scan all its entries, so the worst-case cost is proportional to the number n of entries in the table.

3.2.2 Algorithm

The idea behind suggesting the correct spellings for given misspelled word is to count the minimum number of operations required to convert given misspelled word to other words present in the list of English words and return the words to which given string

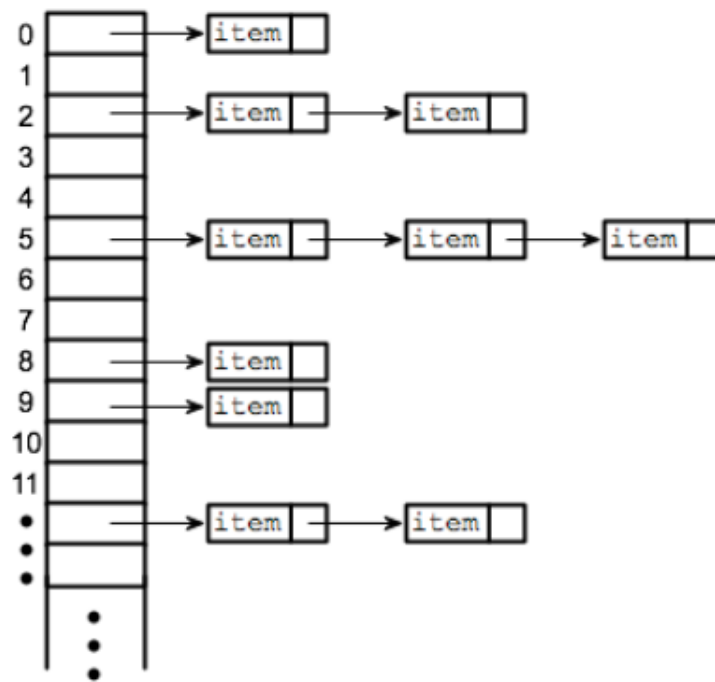


Figure 3.3: Separate Chaining: Collision Resolution Techniques

can be converted in 1 or 2 edit operations. If the number of edit operation is zero for any word than the given word is spelled correctly. The allowed operations are:

- Insertion – Insert a new character.
- Deletion – Delete a character.
- Replace – Replace one character by another.

The minimum number of edit operations required to change one word into another is known as edit distance.

Recursive Approach

To compute the number of edit operations, start comparing one character at a time in both strings from right to left (backwards). Now, for every string there are two options as discussed below:

- If last characters in both the strings are same then just ignore the character and solve the rest of the string recursively.

- Else if last characters in both the strings are not same then, try all the possible operations(insert, replace, delete) and get the solution for rest of the string recursively for each possibility and pick the minimum out of them.

The cost of a table operation is that of scanning the entries of the selected bucket for the desired key. If the distribution of keys is sufficiently uniform, the average cost of a lookup depends only on the average number of keys per bucket—that is, it is roughly proportional to the load factor. For separate-chaining, the worst-case scenario is when all entries are inserted into the same bucket, in which case the hash table is ineffective and the cost is that of searching the bucket data structure. If the latter is a linear list, the lookup procedure may have to scan all its entries, so the worst-case cost is proportional to the number n of entries in the table.

Say there are two strings s_1 and s_2 with length m and n respectively. The cases are as follows:

Case 1: If the last characters are same, ignore the last character and solve recursively solve for $m-1$, $n-1$.

Case 2: If the last characters are not same then try all the possible operations recursively.

- Insert a character into s_1 (same as the last character in string s_2 so that last character in both the strings are same): now s_1 length will be $m+1$, s_2 length: n , ignore the last character and recursively solve for m , $n-1$.
- Remove the last character from string s_1 . Now, s_1 length will be $m-1$, s_2 length will be n ; recursively solve for $m-1$, n .
- Replace the last character into s_1 (same as the last character in string s_2 so that last character in both the strings are same): s_1 length will be m , s_2 length : n , ignore the last character and recursively solve for $m-1$, $n-1$.

The recursive tree for the above approach is shown in figure 3.4¹:

It can be noticed that worst-case time complexity will be $O(3^n)$.

It can also be noticed that many subproblems are solved repeatedly so there are overlapping subproblems here. So, it can be solved using the dynamic programming technique in a bottom-up manner in $O(mn)$ time complexity.

¹<https://algorithms.tutorialhorizon.com/dynamic-programming-edit-distance-problem>

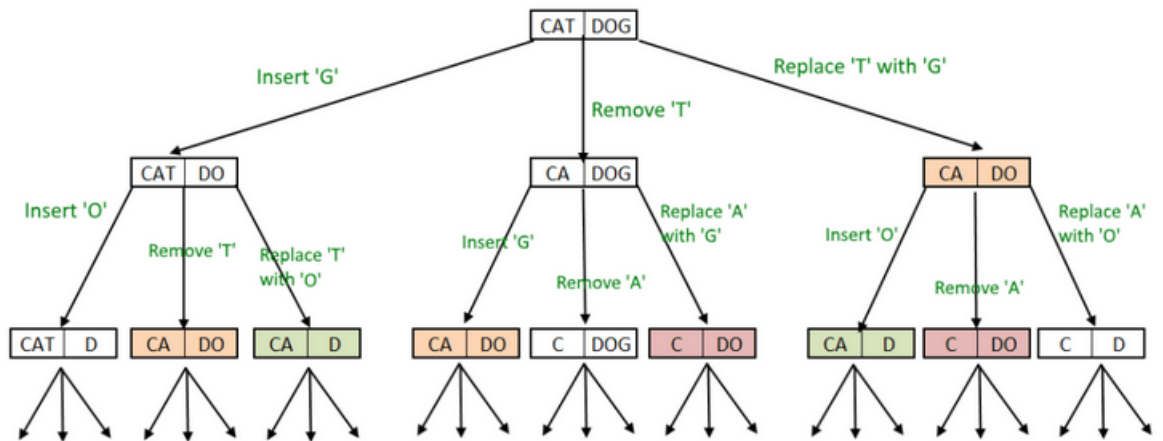


Figure 3.4: Recursive tree showing edit distance computation

Dynamic programming Approach

Since the recursive approach includes many overlapping subproblems that are solved repeatedly. Edit distance can also be computed using the dynamic programming technique in a bottom-up manner.

Levenshtein Distance

The Levenshtein distance has been used in the proposed system to compute the minimum number of single-character edits required to convert one word to other words into words. Each of these operations has a unit cost. For example:

- kitten \rightarrow sitten (substitution of s for k costs 1)
- sitten \rightarrow sittin (substitution of i for e costs 1)
- sitten \rightarrow sitting (insertion of g at end costs 1)

The edit distance problem has an optimal structure. The problem can be broken down into smaller, simple “subproblems”, which can be further broken down into simpler subproblems, and so on, until, the solution becomes trivial.

Say there are two strings $X[1...m]$ and $Y[1...n]$; substrings $X[1...i]$ and $Y[1...j]$, and the aim is to convert string X to string Y . The cases are as follows:

Case 1: One of the two substrings is empty.

If substring X is empty, then all the characters in Y need to be inserted in X, in order to convert X to Y, and the number of operations required will be equal to the number of characters in substring Y. Therefore cost that edit distance will be equal to the length of substring Y.

Example: (“”, “PQR”) \rightarrow (“PQR”, “PQR”) [cost = 3]

If substring Y is empty, then all the characters in X need to be deleted from X, in order to convert X to Y. and the number of operations required will be equal to the number of characters in substring X. Therefore cost that edit distance will be equal to the length of substring X.

Example: (“PQR”, “”) \rightarrow (“”, “”) [cost = 3]

Case 2: Last characters of substring X and substring Y are same.

If the last characters of substring X and Y matches, nothing needs to be done.

Example: (“QDR”, “PQR”) \rightarrow (“QD”, “PQ”) [cost = 0]

Case 3: Last character of substring X and substring Y are different.

If the last characters of substring X and Y matches, then return the minimum of below three operations:

- Insert the last character of Y to X. The size of Y reduces by 1 and size of X remains the same. This accounts for X[1...i], Y[1...j-1] by 1.

Example: (“PQP”, “PQR”) \rightarrow (“PQPR”, “PQR”) == (“PQP”, “PQ”) (using case 2) [cost = 1]

- Delete the last character of X. The size of X reduces by 1 and the size of Y remains the same. This accounts for X[1...i-1], Y[1...j] by 1.

Example: (“PQP”, “PQR”) \rightarrow (“PQ”, “PQR”) [cost = 1]

- Substitute or replace the current character of x by current character of Y. The size of both X and Y reduces by 1.

Example: (“PQP”, “PQR”) \rightarrow (“PQR”, “PQR”) == (“PQ”, “PQ”)

3.3 Word Autocompletion

Word autocompletion refers to the feature in which a system or software predicts the rest of the word a user is typing itself using some particular algorithms. The word

autocorrection module consists of two steps. In the first step, it takes an English dictionary containing around 55,000 words as an input file and stores the words in “Trie” data structure. In the second step, it recursively finds all the words with a given input string as the prefix.

3.3.1 Data Structure

In words autocompletion module, the trie data structure has been used to store the complete list of given English words. Trie is an efficient information retrieval data structure. Trie brings down the search complexities to an optimal limit of key length $[O(M)]$. If keys are stored in the binary search tree, a well balanced BST will need time proportional to $M \cdot \log N$, where M is maximum string length and N is the number of keys in the tree.

Trie data structure is a tree with each node consisting of one letter as data and a list of its children. The root node of the trie is empty. It uses Finite Deterministic automaton. That means it uses the state to state transition. Property that all the descendants of a node have a common prefix i.e sequence of character from root to that node made trie fit data structure to be used for word autocompletion. The figure² given below represents how the words or keys are stored in the trie data structure.

3.3.2 Algorithm

First of all, all the available list of English words are stored in the trie data structure and the property that all the descendants of a node have a common prefix i.e sequence of character from root to the node has been used for word completion. Now for the given query following steps are performed:

- Search for the given string whether it is present or not using trie search algorithm.
- If the given string is not present return null.
- If the given string is present and isEnd is set to true then, return query string itself
- Else if the given string is present and isEnd is set to false then recursively store all the words with query string as the prefix in an ArrayList and return the ArrayList.

²<https://www.hackerearth.com/practice/data-structures/advanced-data-structures/trie-keyword-tree/tutorial/>

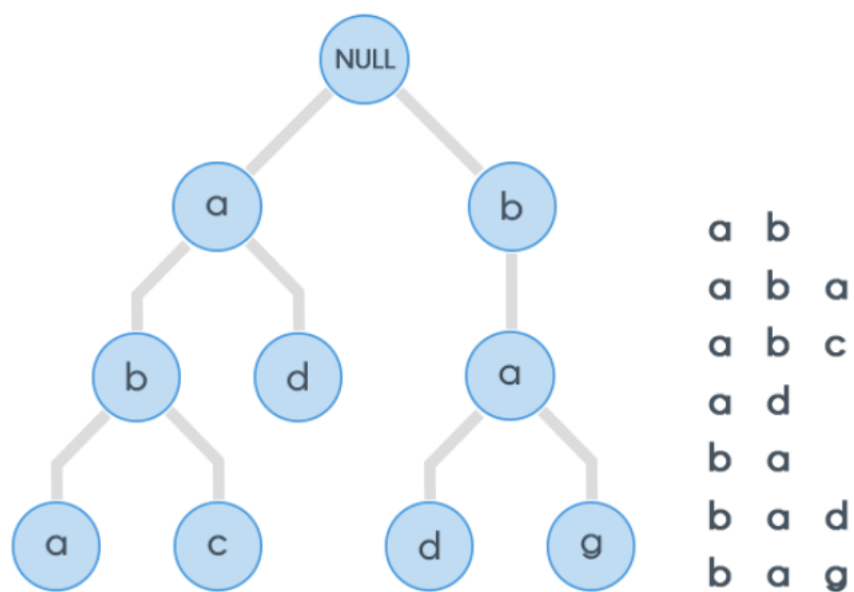


Figure 3.5: Trie Structure

Chapter 4

Implementation

Outline: This chapter presents the following:

- Resources used to implement proposed system
- Modules of system
- Autocorrection module
- Autocompletion module

4.1 Introduction

This section discusses how the proposed word autocorrection and word autocompletion system has been implemented.

4.2 Resources Used

To implement the proposed system following platforms have been used:

- Ubuntu 18.04 operating system
- Eclipse IDE

Eclipse is a commonly used integrated development environment (IDE) for developing applications using the Java programming language and other programming languages such as C/C++, Python.

4.3 Modules

The proposed system takes string consisting of English alphabets and gives one of the two output: either suggest correct spelling or give the complete words for given query string as per users' choice. It broadly has two independent modules for each of its feature.

Module 1: Word autocorrection module first takes file that contain English word list as input, store it in hash table and then use “Edit Distance” problem algorithm, a dynamic programming technique to find the word close to given misspelled string that is to find the words to which given misspelled word can be converted in 1 or 2 edit operations(Insertion, Deletion, Substitution).

Module 2: Word auto-completion module first takes the file that contain English word list as input, store it in the trie data structure and then uses a recursive function to find the words with an input query string as the prefix.

The above two modules are discussed below in detail. The implementation of the whole software consists of four classes. One class is for the main thread, one class is for word autocorrection module and rest two classes are for word auto-completion module.

4.4 Autocorrection Module

The words are stored in a “Hash Table”, they are stored in an array format and each key(words) maps to the values. Hash table with 26 buckets or slots have been used to store words and the hash function used to compute the index of bucket for the given key(word) at which the value can be stored or from which value can be found is given by:

Hash Function : $h(x) = [(\text{ASCII value of first character of word}) - 97]$

Separate chaining with the linked list, a collision resolution technique has been used to overcome collision. The below figure represents how the words are stored in the hash table.

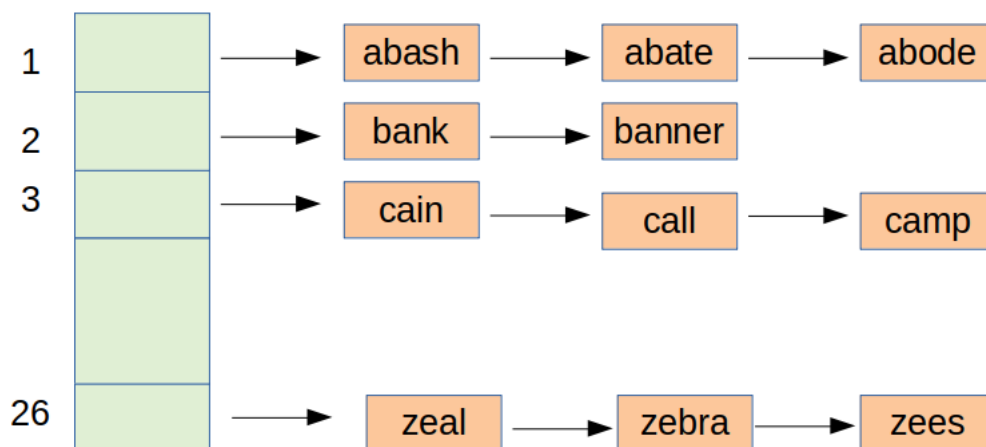


Figure 4.1: Hash Table

The dynamic programming technique, edit distance has been used to implement word autocorrection.

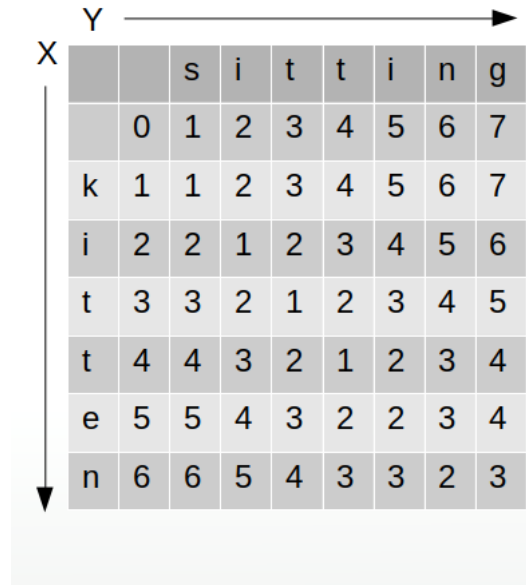


Figure 4.2: Computing Edit Distance

4.5 Autocompletion Module

The words are stored in a “Hash Table”. Each node of trie stores an alphabet as data and list of its children. Each node is defined by a class named “TrieNode”. This class contains a character type member variable named “content” to store the data(alphabet), a boolean type member variable “isEnd” to track of the end of the word and an ArrayList named “childNodesList” of defined class “TrieNode” type to store a list of children. The below figure represents how the words are stored in the trie data structure.

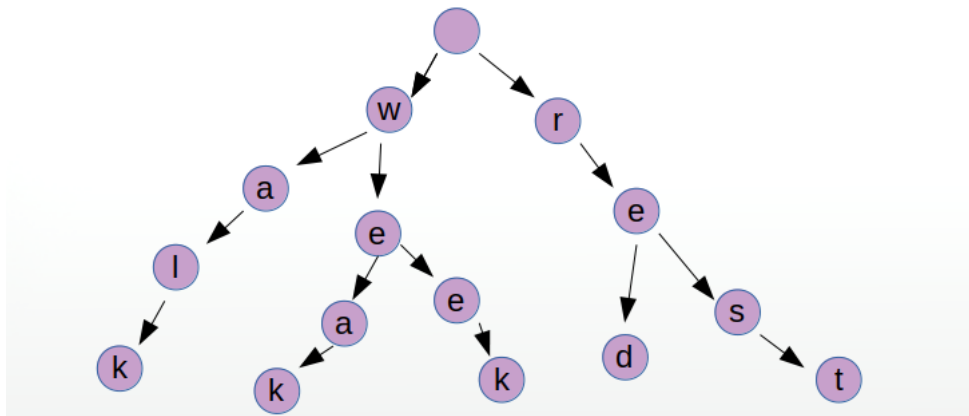


Figure 4.3: Trie Representation

The function “autoComplete” is defined inside another class named “Trie” that defines trie’s standard function. This “autoComplete” function calls another recursive function “getAllWords” that return all the words with given query string provided query string is valid.

Chapter 5

Result Analysis and Testing

Outline: This chapter presents the following:

- Result and testing
- Summary

5.1 Introduction

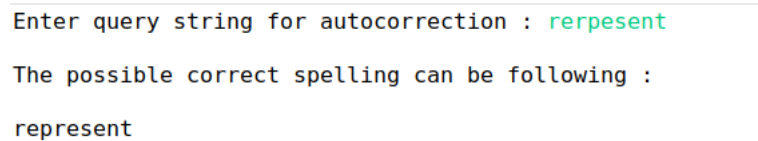
This section discusses the different test cases and the corresponding results that were tested using unit testing and integration testing technique.

5.2 Result and testing

The unit testing technique has been used to test the working of each module independently. After getting satisfied with the result of unit testing of each module, both the modules were combined together and the entire system was tested.

Autocorrection Module Following are the test cases with the screenshot of the result that were tested for word autocorrection module during unit testing.

- **Test case 1:** If by mistake “represent” was misspelled as “rerpesent”, then the output will be the correct spelling i.e “represent”.



```
Enter query string for autocorrection : rerpesent
The possible correct spelling can be following :
represent
```

Figure 5.1: Autocorrection Example1

- **Test case 2:** Test case 1: If someone types the misspelled word “extres”, then the output will be the correct word to which given misspelled can be converted in atmost 2 edit operations i.e “entires”, “entree”, “entries” etc.

```
Enter query string for autocorrection : extres

The possible correct spelling can be following :

entires
entree
entries
exerts
exes
exiles
expires
express
externs
extols
extorts
extra
extras
extrema
extreme
extremes
exudes
```

Figure 5.2: Autocorrection Example2

Autocompletion Module Following are the test cases with the screenshot of result that were tested for word autocompletion module during unit testing.

- **Test case 1:** If user enters some incomplete word, say “repres” then the output will be all the words with given query string as prefix as shown in below figure.

```
Enter query string for autocompletion : repres

Following are the complete words with prefix repres :
representable
representational
representations
representativeness
representatives
represented
representing
represents
repressed
represses
repressing
repressions
repressively
```

Figure 5.3: Autocompletion Example1

- **Test case 2:** If a user enters some incomplete word, say “repres” then the output will be all the words with a given query string as the prefix as shown in below figure.

```
Enter query string for autocompletion : excep
Following are the complete words with prefix excep :
excepted
excepting
exceptionable
exceptionally
exceptions
excepts
```

Figure 5.4: Autocompletion Example2

5.3 Summary

After going through the unit testing and integration testing, it has been found both the modules of autocorrection and autocompletion are working properly, hence the proposed system is working fine.

Chapter 6

Conclusion

Outline: This chapter presents the following:

- Conclusion
- Limitations
- Future direction

The software has been developed that autocorrects and autocompletes the query string as per users' choice. It is a command line software that on executing, prompts the user to choose between two options: autocorrect or autocomplete. Two features have been implemented using two separate independent modules. If the user chooses to autocorrect, then one of two modules that implement autocorrect feature start executing and if the user goes for autocomplete then the second module that implements autocomplete feature executes.

6.1 Limitations

- The first letter should be correct for the desired output of word autocorrect feature.
- Query string should be correct for the desired output of word autocomplete feature.

6.2 Future Direction

There are many more features that can be included in this word autocorrection and autocompletion system. Some of the features that can be added to this application are as follow :

- Extended for complete sentence suggestion.
- Ordering suggestion words according to the search frequency.
- Real-time word autocorrection and autocompletion.

Appendix A

Screenshot and Description of the Implemented System

A.1 Choices to users

Below is the very first thing appear after on executing “Main.class” file, providing users choice to either go for word autocorrection or autocompletion.

```
Enter the digit(0,1,2):  
1 for word autocorrection.  
2 for word autocompletion.  
0 to exit
```

Figure A.1: Choices to users

A.2 Autocorrection

Below figure [A.2](#) shows the output of word autocorrection module depending upon users’ query after integrating modules.

A.3 Autocompletion

Below figure [A.3](#) shows the output of word autocompletion module depending upon users’ query after integrating modules.

```
Enter the digit(0,1,2):
1 for word autocorrection.
2 for word autocompletion.
0 to exit

1
Enter query string for autocorrection : sleepng

The possible correct spelling can be following :
```

- seeing
- seeping
- sleep
- sleeper
- sleeping
- sleeps
- sleepy
- slewing
- steeping
- sweeping

Figure A.2: Autocorrection Example

```
Enter the digit(0,1,2):
1 for word autocorrection.
2 for word autocompletion.
0 to exit

2
Enter query string for autocompletion : gues

Following are the complete words with prefix gues :
```

- guessable
- guessed
- guesses
- guessing
- guesswork
- guesting
- guests
- Exiting

Figure A.3: Autocompletion Example

Appendix B

User manual

B.1 Introduction

The developed software has two features: word autocorrection and autocompletion.

B.2 Step to install your implemented system

Download all the .class files from the github and then execute “Main.class” file to run the software(Make sure all .class files are in same folder).

B.3 Code Snippet

B.3.1 Main Thread class

```
import java.io.File; import java.io.FileNotFoundException; import java.util.ArrayList;
import java.util.List; import java.util.Scanner;

    public class Main

    public static void main(String args[])

        System.out.println("Enter the digit(0,1,2:"); System.out.println("1 for word au-
tocorrection.2 for word autocompletion.0 to exit");

        String query = null;

        Scanner ip = new Scanner(System.in); int op = ip.nextInt();

        switch(op) case 1: System.out.print("Enter query string for autocorrection : ");
query = ip.next();

        LevenshteinEditDistance ed = new LevenshteinEditDistance(); ed.editDistance1(query);
break;

        case 2: File file = new File("/home/anupriya/Documents/Acads/project 6th
sem/english dictionary words"); Trie t = new Trie(); try

            Scanner sc = new Scanner(file); while (sc.hasNextLine()) String y = sc.nextLine();
t.insert(y);

            catch (FileNotFoundException e) e.printStackTrace();

            System.out.print("Enter query string for autocompletion : ");

            List<String> m = new ArrayList<>();

            query = ip.next();

            m = t.autoComplete(query);

            if(m==null || m.size() == 0) System.out.printf("Sorry, No word found with
prefix else System.out.printf("are the complete words with prefix

            for(String str : m) System.out.println(str);

            default: System.out.println("Exiting"); break;
```

B.4 Classe involved in autocorrection

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

```
    public class LevenshteinEditDistance

    static ArrayList<String> dict[] = new ArrayList[26];

    LevenshteinEditDistance () for (int i = 0; i < 26; i++) dict[i] = new Ar-
rayList<String>();

    public static void editDistance1(String x)

    File file = new File("/home/anupriya/Documents/Acads/project 6th sem/english
dictionary words"); try

    Scanner sc = new Scanner(file); while (sc.hasNextLine())

    String y = sc.nextLine(); System.out.println(y); System.out.println((int)y.charAt(0)-
97); dict[(int)y.charAt(0)-97].add(y);

    catch (FileNotFoundException e) e.printStackTrace();

    float start = System.nanoTime(), end; int xlen = x.length(), ylen; int l=0;

    ArrayList<String> sim = new ArrayList<String>();

    for(int d=0;d<dict[(int)x.charAt(0)-97].size();d++)

    String y = dict[(int)x.charAt(0)-97].get(d); ylen = y.length(); System.out.println(d);
l++; int edt[][] = new int[xlen+1][ylen+1];

    for(int i=0;i<=xlen; i++) edt[i][0]=i;

    for(int j=0;j<=ylen; j++) edt[0][j]=j;

    for(int i=1;i<=xlen;i++) for(int j=1;j<=ylen;j++) if(x.charAt(i-1)!=y.charAt(j-
1)) edt[i][j] = Math.min(Math.min(edt[i-1][j]+1, edt[i][j-1]+1), edt[i-1][j-1]+1); else edt[i][j]
= edt[i-1][j-1]; if(edt[xlen][ylen]<=2) sim.add(y);

    System.out.println("possible correct spelling can be following : "); for(int s=0;s<sim.size();s++)
System.out.println(sim.get(s));

    end = System.nanoTime();
```

B.5 Classes involved in autocorrection

B.5.1 Class to define each node

```
class TrieNode

    char content;
    boolean isEnd;
    LinkedList<TrieNode> childNodesList;

    public TrieNode(char c)

        childNodesList = new LinkedList<TrieNode>();
        isEnd = false;
        content = c;

    public TrieNode subNode(char c)

        if (childNodesList != null)
            for (TrieNode eachChild : childNodesList)
                if (eachChild.content == c)
                    return eachChild;
            return null;
```

B.5.2

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

    public class Main

        public static void main(String args[])

            System.out.println("Enter the digit(0,1,2):");
```

```
System.out.println("1 for word autocorrection.2 for word autocompletion.0 to exit");
```

```
String query = null;
```

```
Scanner ip = new Scanner(System.in);  
int op = ip.nextInt();
```

```
switch(1)  
case 1: System.out.print("Enter query string for autocorrection : ");  
query = ip.next();
```

```
LevenshteinEditDistance ed = new LevenshteinEditDistance();  
ed.editDistance1(query);  
break;
```

```
case 2: File file = new File("/home/anupriya/Documents/Acads/project 6th  
sem/english dictionary words");  
Trie t = new Trie();  
try
```

```
Scanner sc = new Scanner(file);  
while (sc.hasNextLine())  
String y = sc.nextLine();  
t.insert(y);
```

```
catch (FileNotFoundException e)  
e.printStackTrace();
```

```
System.out.print("Enter query string for autocompletion : ");
```

```
List<String> m = new ArrayList<>();
```

```
query = ip.next();
```

```
m = t.autoComplete(query);
```

```
        if(m==null || m.size() == 0)

System.out.printf("Sorry, No word found with prefix
else

System.out.printf("are the complete words with prefix

        for(String str : m)

System.out.println(str);


        default: System.out.println("Exiting");
break;
```


References

- <http://techieme.in/building-an-autocomplete-system-using-trie/> [20th Jan, 2019 - 5th April, 2019]
- <https://www.techiedelight.com/levenshtein-distance-edit-distance-problem/> [20th Jan, 2019 - 5th April, 2019]
- <https://algorithms.tutorialhorizon.com/dynamic-programming-edit-distance-problem/> [20th Jan, 2019 - 5th April, 2019]
- <https://www.geeksforgeeks.org/auto-complete-feature-using-trie/> [20th Jan, 2019 - 5th April, 2019]
- <http://www.sarathlakshman.com/2011/03/03/implementing-autocomplete-with-trie-data-structure> [20th Jan, 2019 - 5th April, 2019]
- <https://medium.com/@dookpham/predictive-text-autocomplete-using-a-trie-prefix-tree-data-structure-in-javascript-part-1-6ff7fa83c74b> [20th Jan, 2019 - 5th April, 2019]