A Project Report on

Word Predictor

Submitted in partial fulfilment for the award of the degree of

B.Tech. Computer Science and Engineering

Submitted By:

Keshav Periwal 18BCE0796

Palak Munshi 18BCE0990

Aryaman Jain 18BCE2037

Under the guidance of

Dr. Sweta Bhattacharya

Assistant Professor (Senior)

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

November 2020

Table of Contents

Abstract

In this busy world no one has time now. Technology is being developed every day to increase efficiency. In this front, word predictor is a small step which increases our efficiency multifold times. Word predictor has applications in various areas like texting, search engine etc. To develop our word predictor program, we used the data structure Trie. Our program uses a stored file of words to predict the words which the user may think of thus helping a lot. PROCESS MODEL IDENTIFIER -EVOLUTIONARY MODEL We prefer this model over other models because in this project we want to know the needs and requirements of the user at each and every step and want to make it more efficient and time saving and user-friendly.

Acknowledgement

We would like to thank our professor, **Dr. Sweta Bhattacharya,** for providing us with such a wonderful opportunity to work on an interesting real-world problem. This project will not help a vast number of people but also helped us enrich our skillsets. We are immensely grateful to our faculty for her constant support, guidance and encouragement.

1. Introduction

1.1. Motivation

Autocomplete speeds up human-computer interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. It's impact touches all aspects of our lives- from texting, to searching a result or file on the internet. For some people, this is what let's them do the most basic of tasks such as expressing themselves, communicating with others, searching and just doing their work easily. This project would make the entire process much more convenient for the user and they can save time and energy. Through our project, we truly wish to speed this process up by improving the efficiency and speed of word prediction and solve this particular issue for the people.

1.2. Aim of the proposed Work

To understand the dynamic data structure TRIE based on it develop a program having industrial application to predict words.

1.3. Objective(s) of the proposed work

➢ To understand the dynamic data structure tree used in developing the program.

➢ To understand the data structure ‘trie’ being used in the program.

➢ To construct a strong and efficient algorithm to develop the program which is editable and can be later used as a module for bigger software mechanisms.

➢ To develop a real time program which is efficient and has fast processing and also has an industrial application.

1.4. Report Organization

2. Literature Survey

2.1. Survey of the Existing Models/Work

Sr. No.	Title	Author	Algorithm proposed	Results obtained
1	Word and phrase prediction tool for English and Hindi language	Shashi Pal Singh, Ajai Kumar, Daya Chand Mandad, Yasha Jadwani	The tool is able to predict using multiple algorithms and they tried to build our own bilingual database which contains phrases of variable length along with the frequency of occurrence in corpus. Whenever a word or phrase is selected from the prediction list, its frequency is increased by one. Consequently, the most frequently used word will have highest frequency/probability leading to increase in efficiency of predictions. Those words and phrases that are not present in	Word Prediction predicts list of word choices which the user may be willing to type after a word has been typed. Keystroke savings of various word prediction systems was found to be in

			the database but have been typed by the user are fed into the database for being used in future predictions. Thus, longer the tool is used, better results it will show.	between 37%-47% Conversion of text to speech as well as speech of text of these different languages can also lead to increase in usefulness of this tool. The tool performs predictions using three types of models i.e. frequency model, ngram model and recency model. Over all the application as well as the tool is highly easy to use for users having difficulty in forming words or typing text.
2	N-gram Word Prediction Language Models to Identify the Sequence of Article Blocks in English E-Newspapers	Deepa Nagalavi and M. Hanumanthappa	In this paper the link is established between different blocks of an article with the reading order of a sentence. It is identified with an N-Gram based linguistic processing approach for the retrieval of individual article from newspaper. The model predicts the preceding word knowing the previous content with the probability of a	It was challenging task to get 100% performance as it depends only on the training corpus of large data. However, to improve the

			word sequence. In this work an individual article is identified efficiently with the reading order of the blocks.	word prediction task a linear interpolation model is used which combines trigram, bigram and unigram models. In this work single word prediction is conducted but a set of words or phrases can also be predicted to complete a sentence and merge the blocks of article with the matching sequence.
3	Pretraining Federated Text Models for Next Word Prediction	Joel Stremmel and Arjun Singh	Federated learning is a decentralized approach for training models on distributed devices, by summarizing local changes and sending aggregate parameters from local models to the cloud rather than the data itself. They employ the idea of transfer learning to federated training for next word prediction (NWP) and conduct a number of experiments demonstrating enhancements to current baselines for which federated NWP models have been successful. The study compares federated training baselines from	The research offers effective, yet inexpensive, improvements to federated NWP and paves the way for more rigorous experimentation of transfer learning techniques for federated learning. For central pretraining with federated fine-tuning, we demonstrate

			randomly initialized models to various combinations of pretraining approaches including pretrained word embeddings and whole model pretraining followed by federated fine-tuning for NWP on a dataset of Stack Overflow posts.	a viable procedure but do not achieve performance greater than the federated training baseline with our large network. Simulating federated training conditions to pretrain word embeddings may also yield improved downstream performance by tailoring word representations to reflect different usage across non-IID datasets.
4	A Learning-Classification Based Approach for Word Prediction	Hisham Al-Mubaid	This paper presents a new word prediction approach based on context features and machine learning. The proposed method casts the problem as a learning-classification task by training word predictors with highly discriminating features selected by various feature selection techniques. The method is implemented and evaluated using several datasets and uses very small context (the preceding three words) to predict the following word in that context with high accuracy.	The system achieved impressive results, compared with similar work; the accuracy in some experiments approaches 91% correct predictions. The method was evaluated extensively and compared with the Bayesian algorithm as a

				baseline. The experimental results showed that the approach can achieve impressive accuracy in percentages of correct predictions, which validates its efficiency.
5	A Classification Approach to Word Prediction	Yair Even-Zohar and Dan Roth	They present an approach to word prediction that is based on learning a representation for each word as a function of words and linguistics predicates in its context. To use an expressive representation of the context, they present a way that uses external knowledge to generate expressive context representations, along with a learning method capable of handling the large number of features generated this way that can, potentially, contribute to each prediction. Since the number of words "competing" for each prediction is large, there is a need to "focus the attention" on a smaller subset of these and thus, they exhibit the contribution of a "focus of attention" mechanism to the performance of the word predictor.	The success of this approach hinges on the combination of using a large set of expressive features along with a learning approach that can tolerate it and converges quickly despite the large dimensionality of the data. They believe that this approach would be useful for other disambiguation tasks in NLP. In the experimental study, approach presented is shown to yield

				significant improvements in word prediction tasks.
6	Advances in NLP applied to Word Prediction	Synthema Srl	FastType, an innovative word prediction system that outclassestypical limitations of standard techniques when applied to in-flected languages. FastType is based on combined statistical and rule-base methods relying on robust open-domain language resources, that have been refined to improve Keystroke Saving. Word prediction is particularly useful to minimise keystrokes for users with special needs, and to reduce misspellings for users having limited language proficiency.	FastType has been tried out and eval-uated in some test benchmarks, showing a relevant improve-ment in Keystroke Saving, which now reaches 51%, comparable to what achieved by word prediction methods for non-inflected languages. They additionally enriched the Language Model with morpho-syntactic information and provided the pre-diction method with an on-the-fly Part-of-Speech word tagger and large lexicon dictionaries.
7	User interaction with word prediction:the effects of prediction	Keith Trnka, John McCaw, Debra Yarrington, Kathleen F.	They present a study of two different word prediction methods compared against letter-by-letter entry at simulated AAC communication rates.	They find that word prediction systems can in fact speed communicatio

	quality	McCoy		n rate (an advanced system gave a 58.6%improvement), and that a more accurate word prediction system can raise communication rate higher than is explained by the additional accuracy of the system alone due to better utilization (93.6% utilization for advanced vs. 78.2% for basic)

2.2. Summary/Gaps identified in the Survey

From the survey, it became apparent that there's a need for simple, efficient and easy-to-use algorithm for word prediction. The required algorithm must be fast and should display output in a way which is easily readable. Our proposed algorithm caters to these exact needs.

3. Proposed System Requirements Analysis and Design

3.1. Introduction

In this segment, we discuss the requirements of our project and then its analysis and design in detail.

3.2. Requirement Analysis

3.2.1. Stakeholder Identification

Primary Stakeholders:

● Project leader

● Senior Management

● Project team members(algorithm management)

● Dataset Trainer

● Designer

● Resource Managers (database management)

● Product user group (Users)

● Project testers

Secondary Stakeholders:

● Project customer

● Consultants to the project

3.2.2. Functional Requirements

Our program is the main functional requirement where we will be using a programming language, example C, C++, etc. i. The system shall constantly access the dictionary file. User will give input in the form of a word (partial or full). Words matching the letters typed so far will be displayed on the screen. ii. If there are no matching words, then the system will ask if the user wants to add the word to dictionary iii. After every word prediction cycle completes, the system asks if the user wants to add more words. If yes, another cycle starts, else the system halts.

3.2.3. Non-Functional Requirements

a. Performance Requirements

● The system should be interactive to users.

● The interface is simple and easy to use.

● System is user friendly, self-explanatory and also it is provided with a help guide.

● This system can be used by everyone.

● Speed: The system should be made as fast as possible to reduce response time.

● Throughput: The throughput should be as high as possible. We should be able to attain maximum output in minimum time.

● Resource Utilization: Resources are modified according to user requirements.

b. Safety Requirements

● Operation of weekly backups for the database should take place.

● The system should be tough and not prone to breakdowns.

c. Security Requirements

● The administrators maintain the system as per the maintenance contract.

● The system has to be secure from attacks.

● In case of breakdown should be stabilized soon.

d. Quality Attributes

a. Reliability

The solution should provide reliability to the user that the product will run with all the features mentioned in this document are available and executing perfectly. It should be tested and debugged completely. All exceptions should be well handled.

● Try to attain maximum reliability.

● Reliability will also be higher since we try to attain maximum accuracy.

● Maintain proper and updated dictionary files to improve reliability.

b. Accuracy

The product should be able to reach the desired level of accuracy. This prototype version is primarily for proving the concept of the project.

- The information provided in the dictionary files and by the user should be correct.
- Minimize the errors.
- All operations will be done correctly to increase the level of accuracy

3.2.4. System Requirements

3.2.4.1. H/W Requirements(details about Application-Specific Hardware)

The hardware requirements for the application are very low. The most important parts will be a fast-hard drive for the creation of a custom corpus. Microsoft Windows, *Nix, and Mac OS all have browsers that will work with the application. What follows are the minimum specifications for the various computers and other hardware:

I. Client Requirements

❑ Minimum CPU – P3/AMD Athlon 1.0 GHz+

❑ Minimum Disk Space – 512MB

❑ Minimum Memory – 256MB

❑ Minimum Network – 10/100MB Network

❑ Minimum Display – 800x600 Color CRT

❑ Required Software – Web Browser (Google Chrome)

II. Other Hardware Required

❑ Touch Screens/Keyboard

3.2.4.2. S/W Requirements (details about Application-Specific Software)

1. C++ compiler

2. Anaconda Distribution

3. NLTK Toolkit

4. Spacy toolkit

5. Windows Operating System/MacOS/Linux

3.2.5. Software Requirement Specification document

1. Introduction

Purpose

The goal of this document is to provide support information on Word Predictor (v1.0). It will attempt to explain the functionality of the program and the features it provides. It will, however, not be a user manual and provide instructions regarding the full working of the program or the usage of the program.

Document Conventions

- Keywords in this SRS have been highlighted by making the text bold and italicized.
- Priorities for higher-level requirements can be considered to be inherited by the sub-requirements if priorities of sub-requirements are not explicitly mentioned.

Intended Audience and Reading Suggestions

This software specification document is mainly intended for 3 categories of users:

●Developers: Developers can review the project's capabilities and conveniently understand where their efforts should be directed to improve, refine and add features to it.

● Project Testers: They can use this document as the basis for their testing strategy since quite a few bugs are easier to find with the requirements document as reference. In this way, testing becomes more methodically organized.

● End Users: This document can also be read by the end users of this product who wish to read more in detail about the product along with its features and capabilities.

Product Scope

Word Predictors have applications in numerous fields such as messaging applications like WhatsApp, web search engines, word processors, command like interpreters and more. Word prediction, autocomplete, or word completion, is a feature in which an

application predicts the rest of a word a user is typing. Many autocomplete algorithms learn new words after the user has written them a few times, and can suggest alternatives based on the learned habits of the individual user. However, existing programs can be made more efficient using data structures like TRIE.
Word prediction softwares was originally intended to help people with physical disabilities in increasing their typing speed, as well as to help them decrease the number of keystrokes needed in order to complete a word or a sentence. This made typing convenient for the target demographic. Moreover, in today's world, almost all of us are dependent on word prediction and autocomplete for a smooth and fast typing experience.
Thus, in this front we will develop our own program (product) for word prediction using data structure trie which is expected to increase/ efficiency of the user by at least 10%. We believe that this product will cater to the needs of a very wide demographic.

References

- https://www.1q4all.com/pdf/research/WordPredictionSoftware.pdf
- Montgomery, D. J., Karlan, G. R., & Coutinho, M. (2001). The effectiveness of word processor spell checker programs to produce target words for misspellings generated by students with learning disabilities. Journal of Special Education Technology, 16(2), 27-40.
- Handley-More, D. (2003). Facilitating written word using computer word processing and word prediction. American Journal of Occupational Therapy, 57(2), 139-151.
- https://www.geeksforgeeks.org/trie-data-structure-using-smart-pointer-and-oop-in-c/

2. Overall Description

Product Perspective

Word Prediction speeds up human-computer interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. It's impact touches all aspects of our lives- from texting, to searching a result or file on the internet. The original purpose of word prediction software was to help people with physical disabilities increase their typing speed, as well as to help them decrease the number of keystrokes needed in order to complete a word or a sentence. But now, a word predictor has applications in various areas like texting, search engines and thus, can help address this issue for a wide demographic. Our product, Word Predictor, will potentially increase efficiency multifold times.

Product Functions

Our program is the main functional requirement where we will be using a programming language, example C, C++, etc.

- The system shall constantly access the dictionary file. User will give input in the form of a word (partial or full). Words matching the letters typed so far will be displayed on the screen.
- If there are no matching words, then system will ask if user wants to add the word to dictionary
- After every word prediction cycle completes, the system asks if the user wants to add more words. If yes, another cycle starts, else the system halts.

User Classes and Characteristics

- **All types of users-**

Word Predictor is intended to speed up typing for people from all types of backgrounds and experience. Moreover, using it requires very little skill set and can hence, be used by all types of users.

- **Open source software developers and contributors-**

People who have in depth knowledge of programming language used in the project, in order to understand and be able to meaningfully contribute to the source code.

Operating Environment

Since the Word Predictor will be made primarily using cpp, it can be successfully run on all operating systems with some minor changes in the libraries which might be Operating System dependent.

Design and Implementation Constraints

- Code and commands for implementation of above.
- Usage of slang words
- Usage of languages other than english

- Addition of wrong spellings of the word making the identification of words more difficult.

User Documentation

Tutorial :
https://drive.google.com/drive/folders/1y4o9jlO9XhvWjrZh0KxJxm0TQrhKQtbb?usp=sharing

Assumptions and Dependencies

- In this we assume that the user does not search the word that does not exist, if the user does then that word will be added in the corpus and affect the searching for other users.
- In this we assume the api(for compiling the c++ code) documentation does not change in the future.

3. External Interface Requirements

User Interfaces

Users will be using it using any electronic devices such as laptops, mobiles, computers etc. Input device of system will be used to give input to the software such as Codeblocks or any online cpp compiler and when the user presses enter, the software shows words, that matches letters entered so far, on the screen of device being used.

Hardware Interfaces

The hardware interface (keyboard, keypad) will be used to enter letters and then after compilation of the code, the list of words will be fetched from the corpus file.

Software Interfaces

Our program uses an external custom corpus which will be accessed throughout the program. In the file, words are stored in the title case without any space between two words. Every capital letter denotes the beginning of a new word and next one denotes end.

Communications Interfaces

This project supports all types of web browsers. We are using a simple yet interactive word prediction system.

4. System Features

System Feature 1 :

System Feature	**Word Prediction Program**
Priority	**High**
Description	**It will suggest relevant words to the user to minimize time and keystrokes.**

Action	**This module is activated after the user provides a few letters into the interface.**
Result	**The system shows the results of word prediction query and the user can choose one of the displayed words or add a word to the corpus.**

Functional Requirements	TRIE data structure implemented in C++ language and a external custom corpus

System Feature 2:

System Feature	User interaction and feedback
Priority	Medium
Description	The user can use the program in a more convenient and interactive way
Action	One the user enters a few letters in the portal, the website will display the list of words from corpus and other relevant options.
Result	If a new word is searched or added, it will be included into the original dictionary for even better performance in future.
Functional Requirements	A website

5.Other Nonfunctional Requirements

a. Performance Requirements

- The system should be interactive to users
- The interface is simple and easy to use.
- System is user friendly, self-explanatory and also it is provided with a help guide.
- This system can be used by everyone.
- Speed: The system should be made as fast as possible to reduce response time.
- Throughput: The throughput should be as high as possible. We should be able to attain maximum output in minimum time.
- Resource Utilization: Resources are modified according to user requirements

  .

b. Safety Requirements

- Operation of weekly backups for the database should take place.
- The system should be tough and not prone to breakdowns.

c. Security Requirements

- The administrators maintain the system as per the maintenance contract.
- The system has to be secure from attacks.
- In case of breakdown should be stabilized soon.

d. Quality Attributes

a. Reliability

The solution should provide reliability to the user that the product will run with all the features mentioned in this document are available and executing perfectly. It should be tested and debugged completely. All exceptions should be well handled.

- Try to attain maximum reliability.
- Reliability will also be higher since we try to attain maximum accuracy.
- Maintain proper and updated dictionary files to improve reliability.

b. Accuracy

The product should be able to reach the desired level of accuracy. This prototype version is primarily for proving the concept of the project.

- The information provided in the dictionary files and by the user should be correct.
- Minimize the errors
- All operations will be done correctly to increase the level of accuracy

e. Business Rules:

Module Name	**To determine if TRIE is a suitable data structure for word prediction program**

Research Objective	**Compare the performance of some other data structures and then TRIE when used to predict words and choose the data structure which produces the best result.**
Description	**After comparing a few studies on prediction using data structures, it was found that TRIE data structure was much more efficient than other data structures because of its unique advantages such as hashing, self-balancing, no need for collision handling and ease of printing all words in alphabetical order.**
Expected Outcome	**Confirmation that TRIE is an efficient data structure for word prediction.**

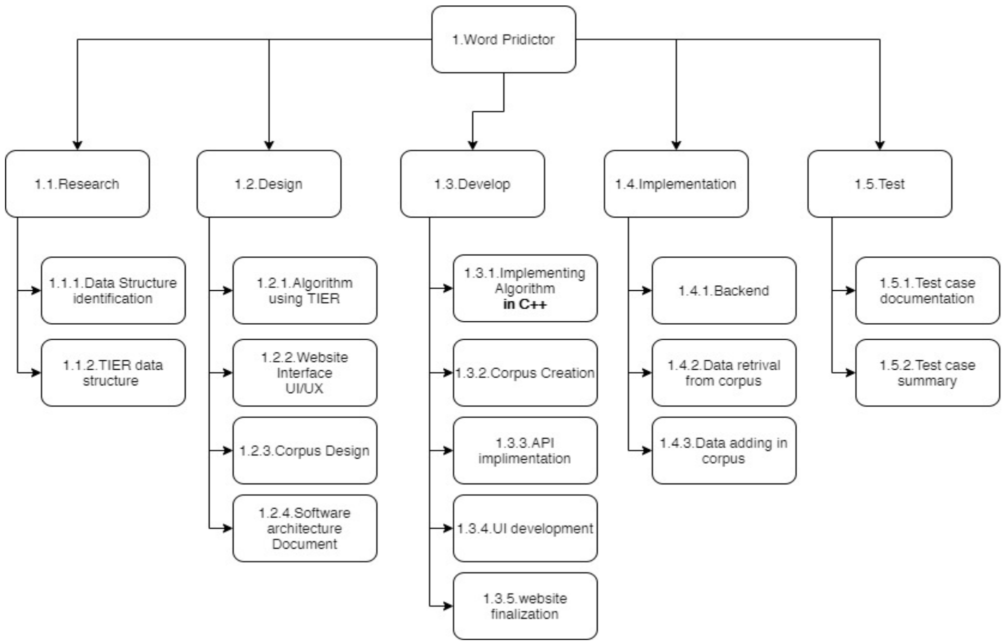
Appendix A: Glossary

Trie: Trie is an efficient information retrieval data structure. Using Trie, search complexities can be brought to optimal limits (key length). If we store keys in binary search tree, a well balanced BST will need time proportional to M * log N, where M is maximum string length and N is number of keys in the tree. Using Trie, we can search the key in O(M) time.

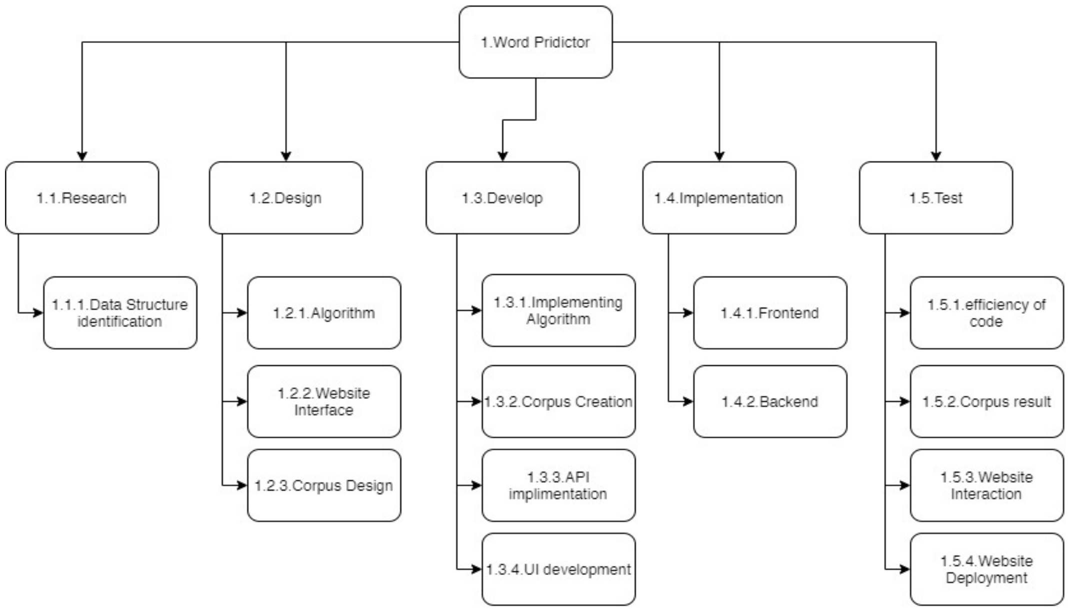
Corpus: It is a collection of written texts, especially a body of writing on a particular subject. Here, it refers to the dictionary of words from which words will be fetched and made available to the user.

3.2.6. Work breakdown structure

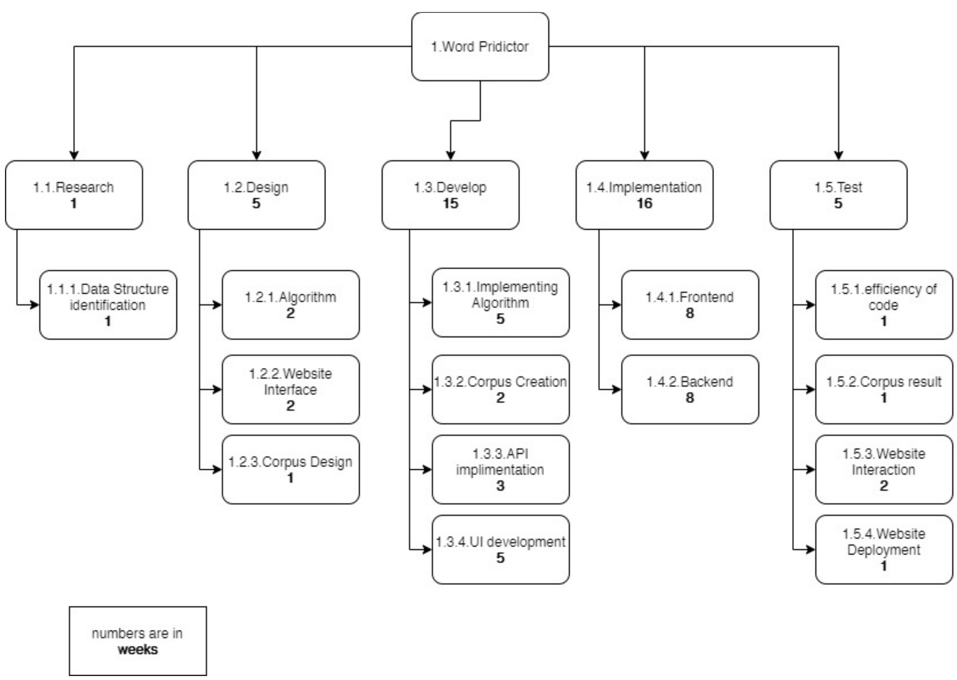
1. Noun-oriented WBS: It is a deliverable-oriented WBS that defines the project work in terms of the components (physical or functional) that make up the deliverable



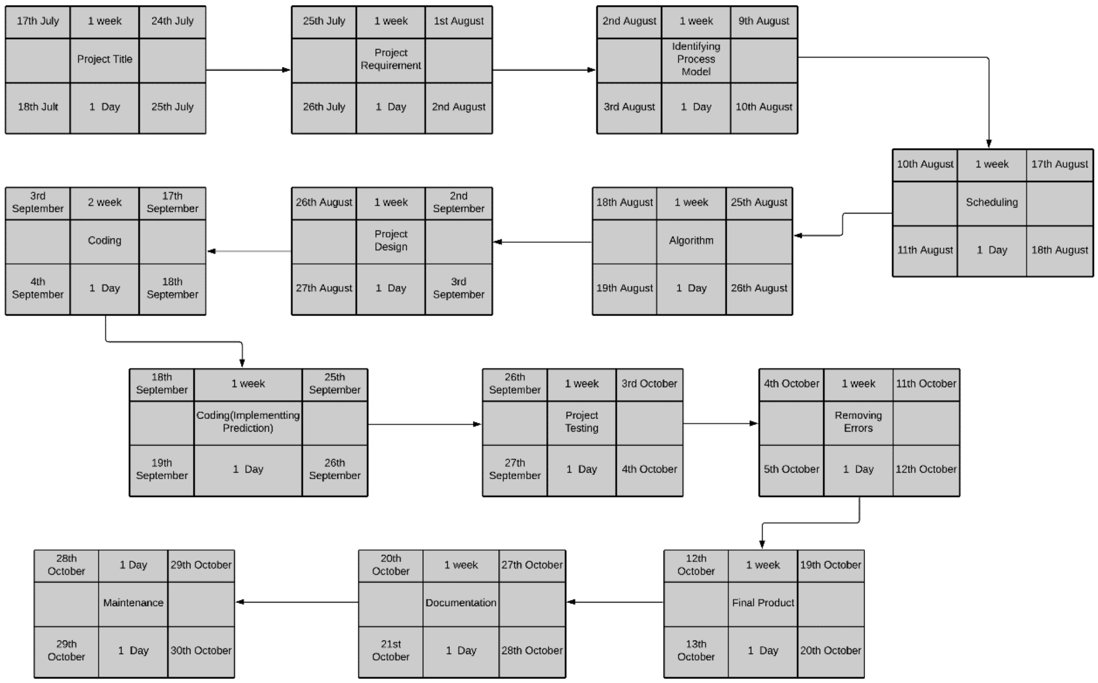
2. Verb-oriented WBS: It is a task-oriented WBS that defines the deliverable of project work in terms of the actions that must be done to produce the deliverable.



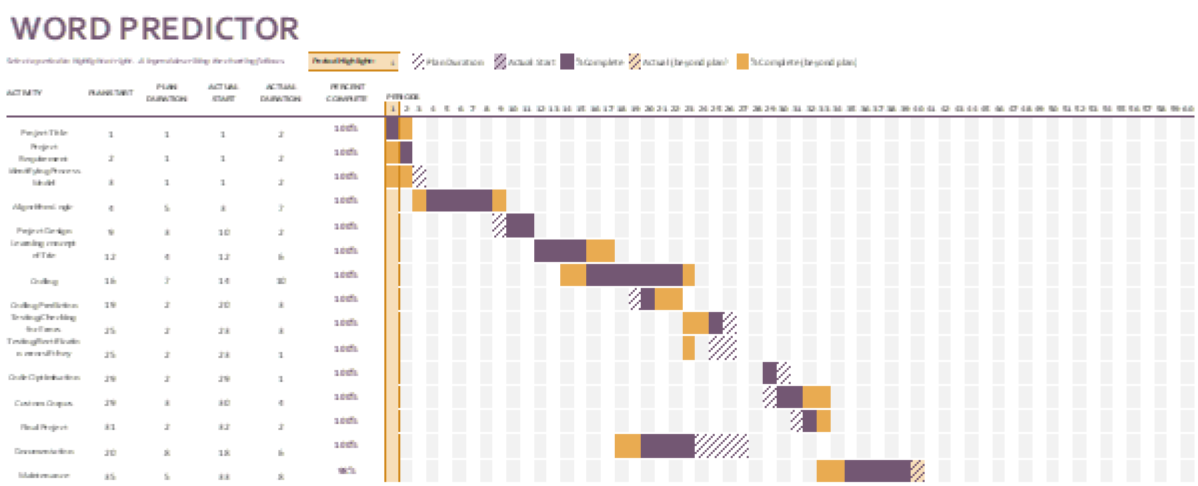
3. Time-Phased WBS: It is a time phased WBS and is typically used for very long projects. The project is broken down into major phases instead of tasks.



3.2.7. Pert chart



3.2.8. Gantt Chart



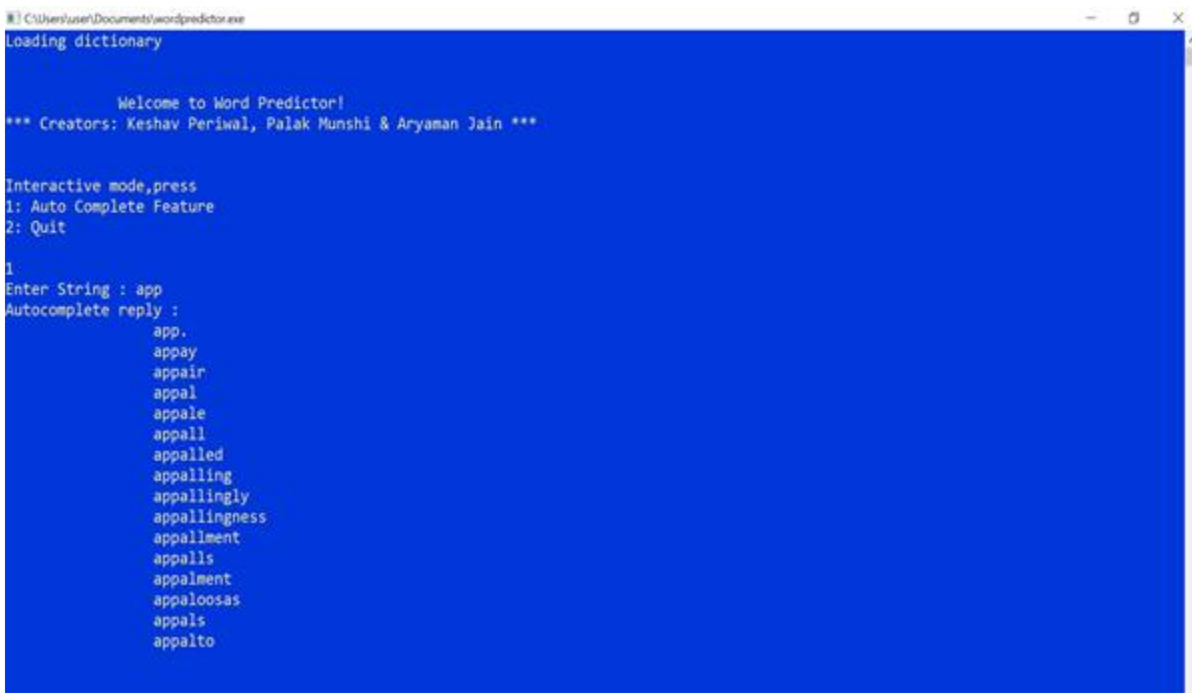
4. Design of the Proposed System

4.1. Introduction

Users will be using it using any electronic devices such as laptops, mobiles, computers etc. Input device of system will be used to give input to the software such as Codeblocks or any online cpp compiler and when the user presses enter, the software shows words, that matches letters entered so far, on the screen of device being used.

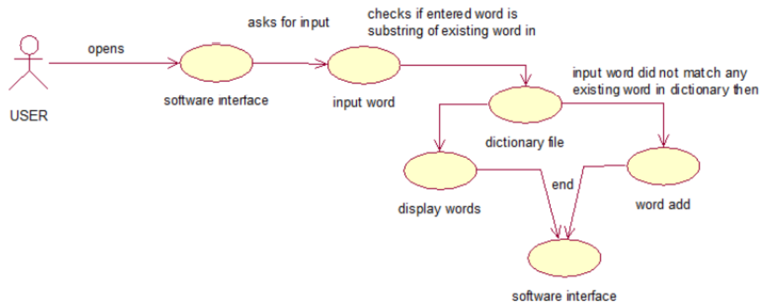
4.2. High level Design

4.2.1. UI design



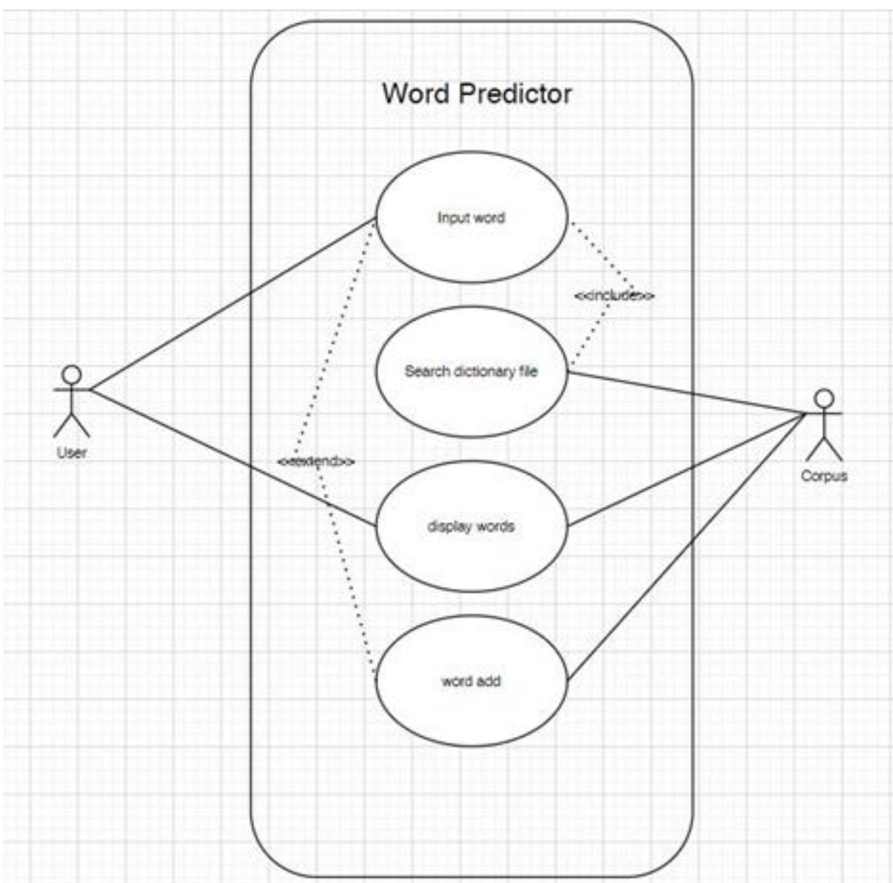
4.3. Detailed Design (ER Diagram/UML Diagram/Mathematical Modeling)

4.3.1. ER Diagram

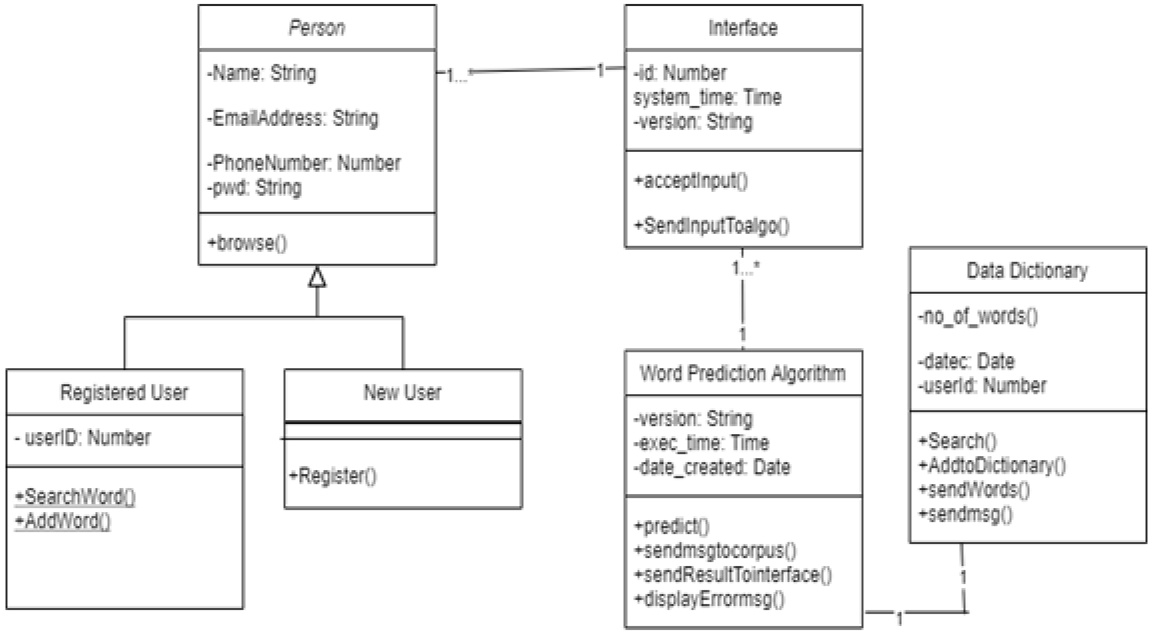


4.3.2. UML diagram (Use case, class, Statechart, Activity and interaction diagrams)

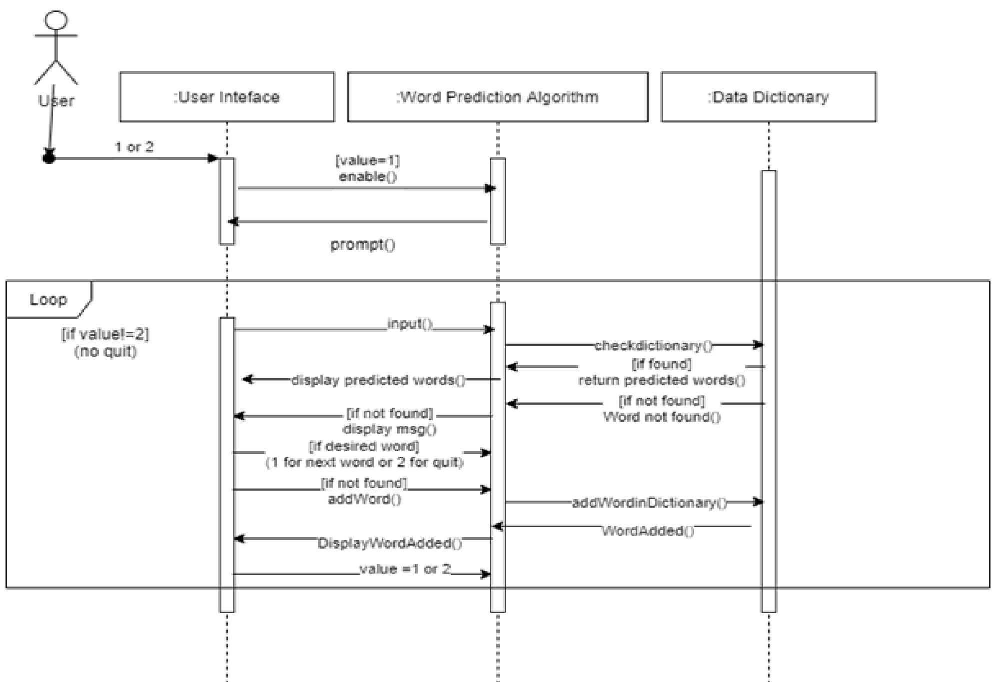
A) Use Case Diagram



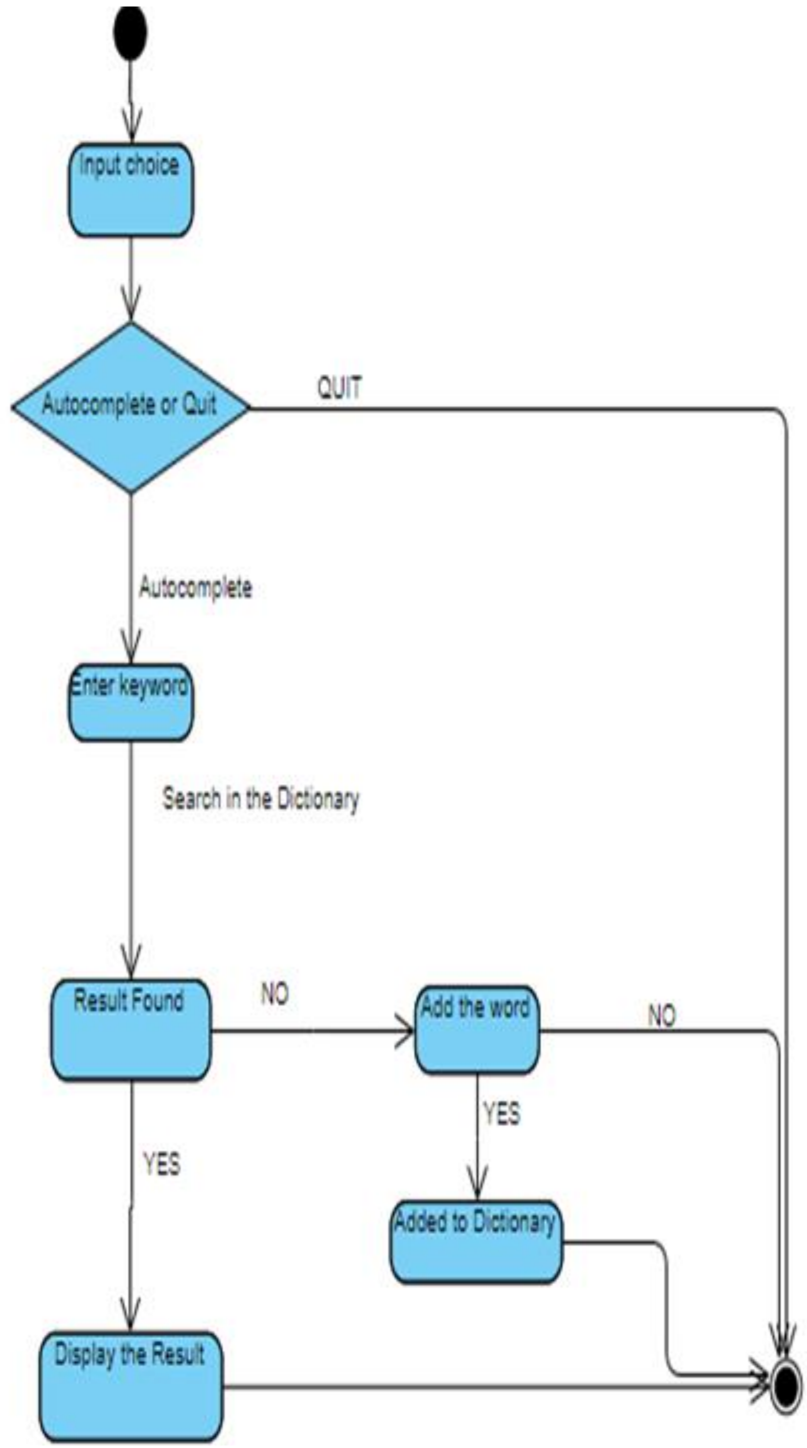
B) Class Diagram



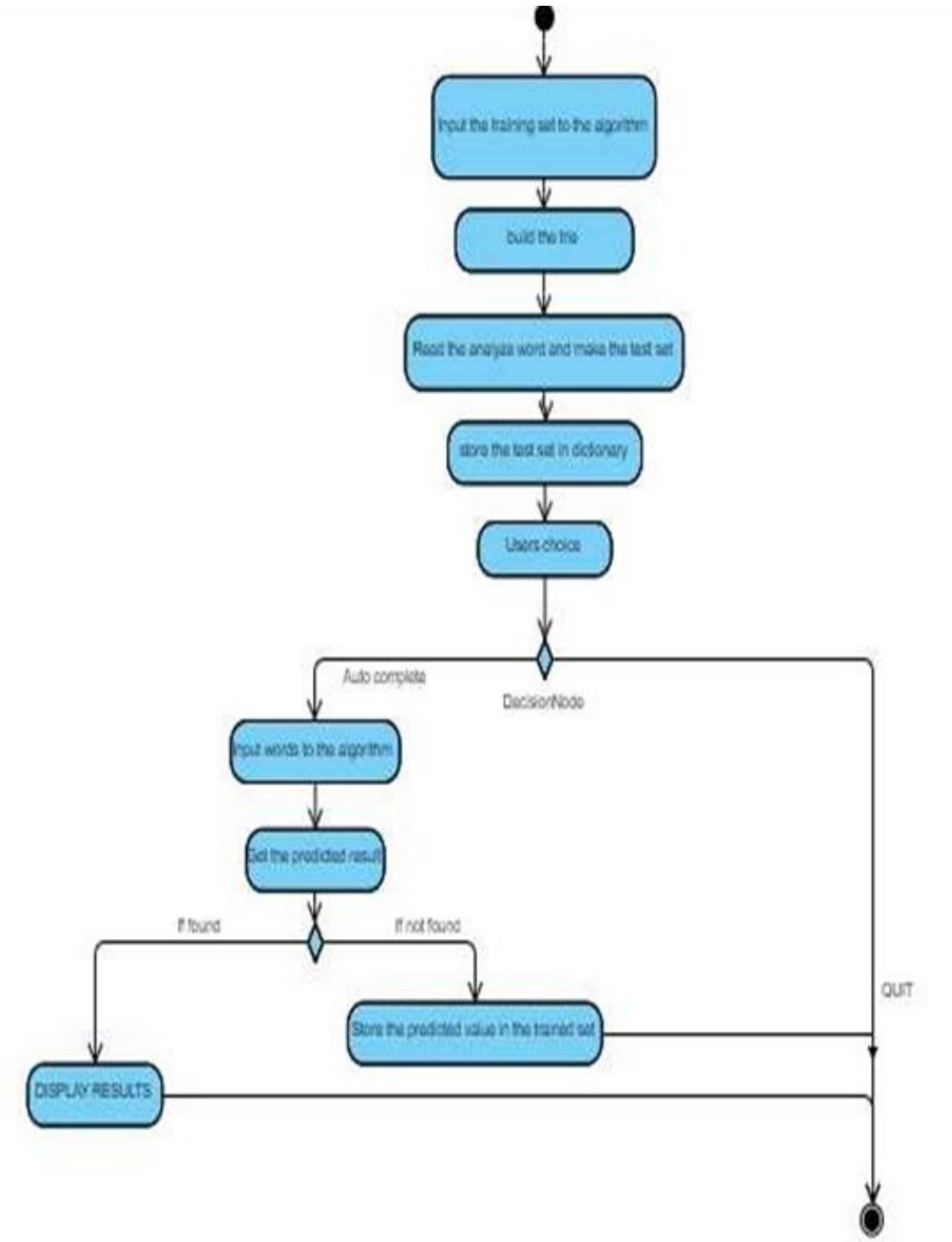
C) Sequence Diagram



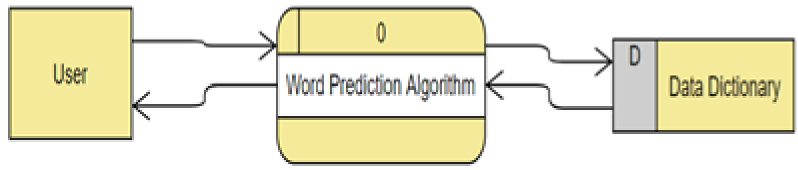
D) State Transition Diagram



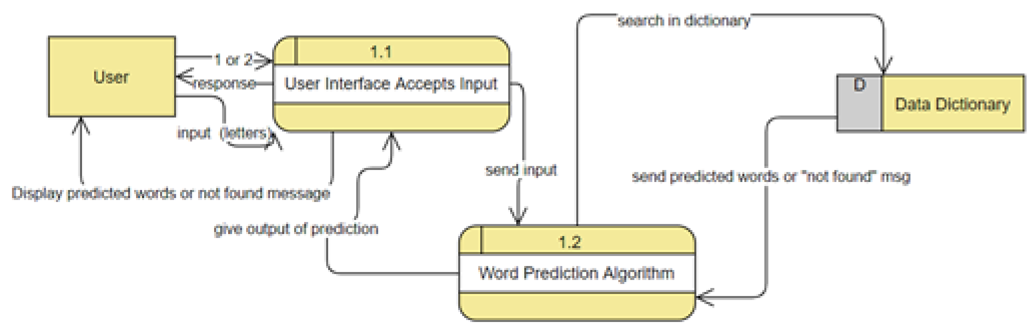
E) Activity Diagram



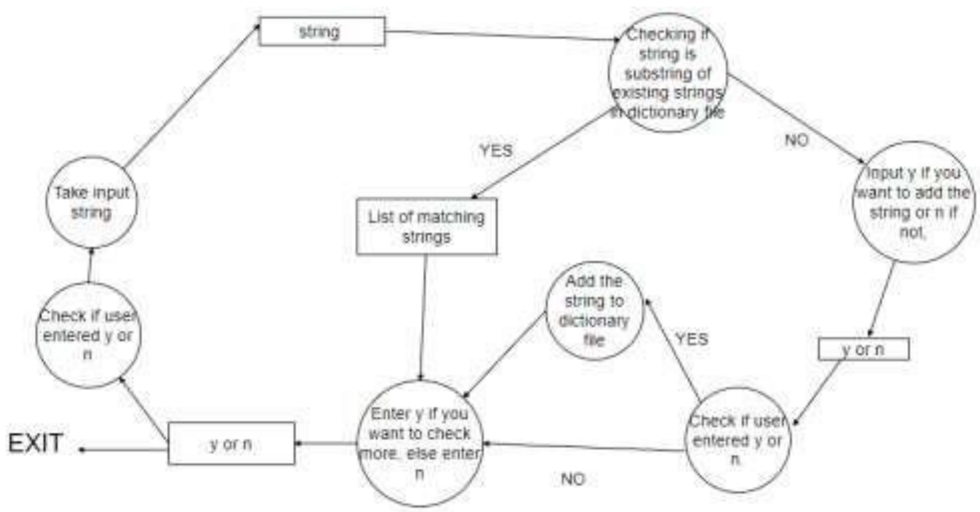
F) Data Flow Diagram Level 0



G) Data Flow Diagram Level 1



H) Data Flow Diagram Level 2



5. Implementation and Testing (Snap shots with description)

5.1. Implementation details (snapshots)

<u>Module 1:</u> Class Trie

<u>Code Snippet:</u>

```
class Trie
{
public:
Trie();
~Trie();
void addWord(string s);
bool searchWord(string s);
bool autoComplete(string s, vector<string> &);
void parseTree(Node *current, char *s, vector<string> &, bool &loop);
private:
Node *root;
};
Trie::Trie()
{
root = new Node();
}
Trie::~Trie()
{
// Free memory
}
```

<u>Description:</u> This is the main structure of our code. In this class Trie we have 4 functions addWord, searchWord, autoComplete, parseTree.

<u>Module 2:</u> addWord()

<u>Code Snippet:</u>

```
void Trie::addWord(string s)
{
Node *current = root;
if (s.length() == 0)
{
current->setWordMarker();
return;
}
for (int i = 0; i < s.length(); i++)
{
    Node *child = current->findChild(s[i]);
    if (child != NULL)
    {
        current = child;
    }
    Else
    {
        Node *tmp = new Node();
        tmp->setContent(s[i]);
        current->appendChild(tmp);
        current = tmp;
    }
    if (i == s.length() - 1)
    current->setWordMarker();
}
}
```

<u>Description:</u> This function helps us to add the word in the corpus if the entered word from the user is not available in the corpus. In for loop the we are finding whether the word is there or not, if there then it will return current as a word and if not then else statement will append the word in the corpus.

<u>Module 3:</u> Function searchWord

<u>Code Snippet:</u>

```
bool Trie::searchWord(string s)
{
Node *current = root;
while (current != NULL)
{
for (int i = 0; i < s.length(); i++)
    {
        Node *tmp = current->findChild(s[i]);
        if (tmp == NULL)
        return false;
        current = tmp;
    }
if (current->wordMarker())
return true;
Else
return false;
}
return false;
}
```

<u>Description:</u> This function is used to search the word entered by the user. In for statement findChild is used to find the word in the corpus, if the word is found then it will return it as current or else it will return false.

<u>Module 4:</u> Function autoComplete

<u>Code Snippet:</u>

```
bool Trie::autoComplete(std::string s, std::vector<string> &res)
{
Node *current = root;
for (int i = 0; i < s.length(); i++)
    {
        Node *tmp = current->findChild(s[i]);
        if (tmp == NULL)
        return false;
        current = tmp;
    }
char c[100];
strcpy(c, s.c_str());
bool loop = true;
parseTree(current, c, res, loop);
return true;
}
```

Description: This function is used to concatenate the words and form the words. In for statement first it will try to find the words with the string entered by the user and after that it concatenates the words and shows the output.

Module 5: Function parseTree

Code Snippet:

```
void Trie::parseTree(Node *current, char *s, std::vector<string> &res, bool &loop)
{
char k[100] = {0};
char a[2] = {0};
if (loop)
{
    if (current != NULL)
    {
        if (current->wordMarker() == true)
        {
            res.push_back(s);
            if (res.size() > 15)
                loop = false;
        }
    vector<Node *> child = current->children();
    for (int i = 0; i < child.size() && loop; i++)
    {
        strcpy(k, s);
        a[0] = child[i]->content();
        a[1] = '\0';
        strcat(k, a);
        if (loop)
            parseTree(child[i], k, res, loop);
    }
    }
}
}
```

Description: This function is used to parse the tree and concatenate the input string with different letters and try to form a word and then show them as an output.

Module 6: Function loadDictonary

Code Snippet:

```
bool loadDictionary(Trie *trie, string filename)
{
ifstream words;
ifstream input;
words.open(filename.c_str());
if (!words.is_open())
{
    cout << "Dictionary file Not Open" << endl;
    return false;
}
while (!words.eof())
{
    char s[100];
    words >> s;
    trie->addWord(s);
}
return true;
}
```

Description: This function loads the corpus and then tries to run all the functions explained above.

Code Optimization

Various Code optimization techniques are- Constant Folding, Constant Propagation, Common Subexpression elimination, Code movement, Dead code elimination, Strength reduction. The ones demonstrated below incorporate Dead Code Elimination and Constant Propagation.

1) Before Optimsation:

```
Node *Node::findChild(char c)
{
 for (int i = 0; i < mChildren.size(); i++)
 {
 Node *tmp = mChildren.at(i);

 if(tmp->content() == c)
 {
     continue;
 }
 else if(tmp->content() == c)
    {
        return tmp;
 }
 }
 return NULL;
}
```

After Optimization:

```
Node *Node::findChild(char c)
{
 for (int i = 0; i < mChildren.size(); i++)
 {
 Node *tmp = mChildren.at(i);

 if (tmp->content() == c)
 {
 return tmp;
 }
 }
 return NULL;
}
```

The optimization: The if-else conditions are optimized to just an if condition without any loss in logic and thus time and space complexities are reduced.

2) Before Optimization:

```
void Trie::addWord(string s)
{
 Node *current = root;
 if (s.length() == 0)
 {
 current->setWordMarker();
 return;
 }

 for (int i = 0; i < s.length(); i++)
 {
 Node *child = current->findChild(s[i]);

 if (child != NULL)
 {
 current = child;
 }
```

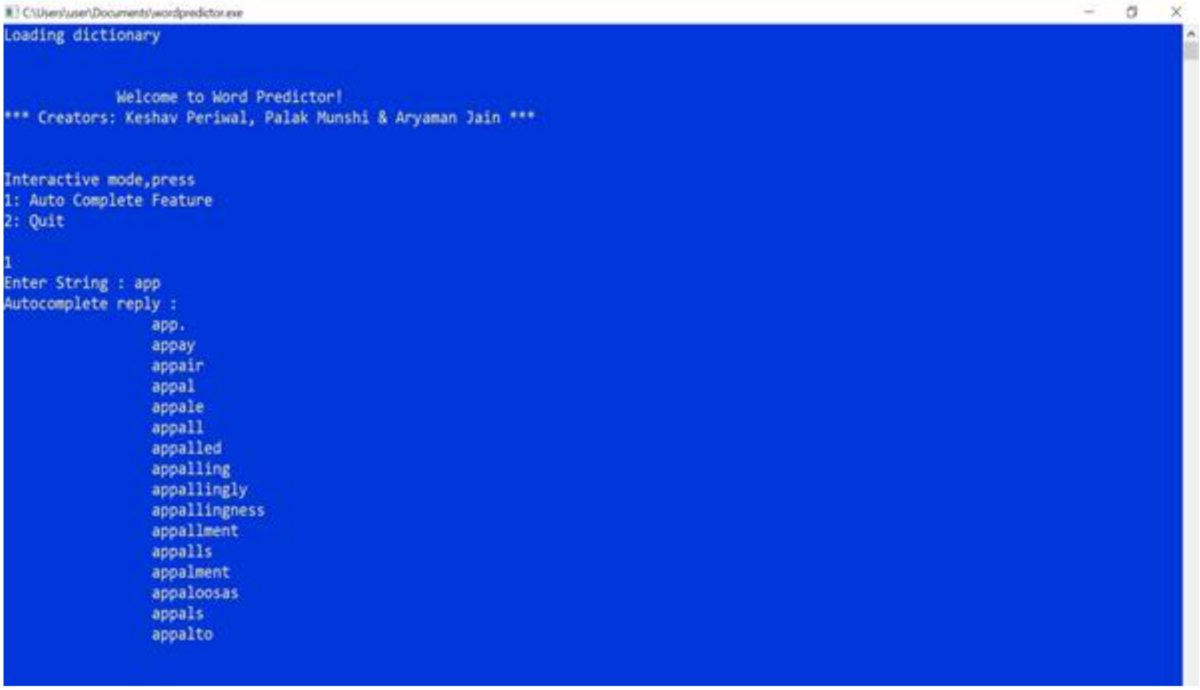
After Optimization:

```
void Trie::addWord(string s)
{
Node *current = root;
int m=s.length();
if (m == 0)
{
current->setWordMarker();
return;
}

for (int i = 0; i <m; i++)
{
Node *child = current->findChild(s[i]);
if (child != NULL)
{
current = child;
}

else
{
Node *tmp = new Node();
tmp->setContent(s[i]);
current->appendChild(tmp);
current = tmp;
}
if (i == m - 1)
current->setWordMarker();
}
}
```

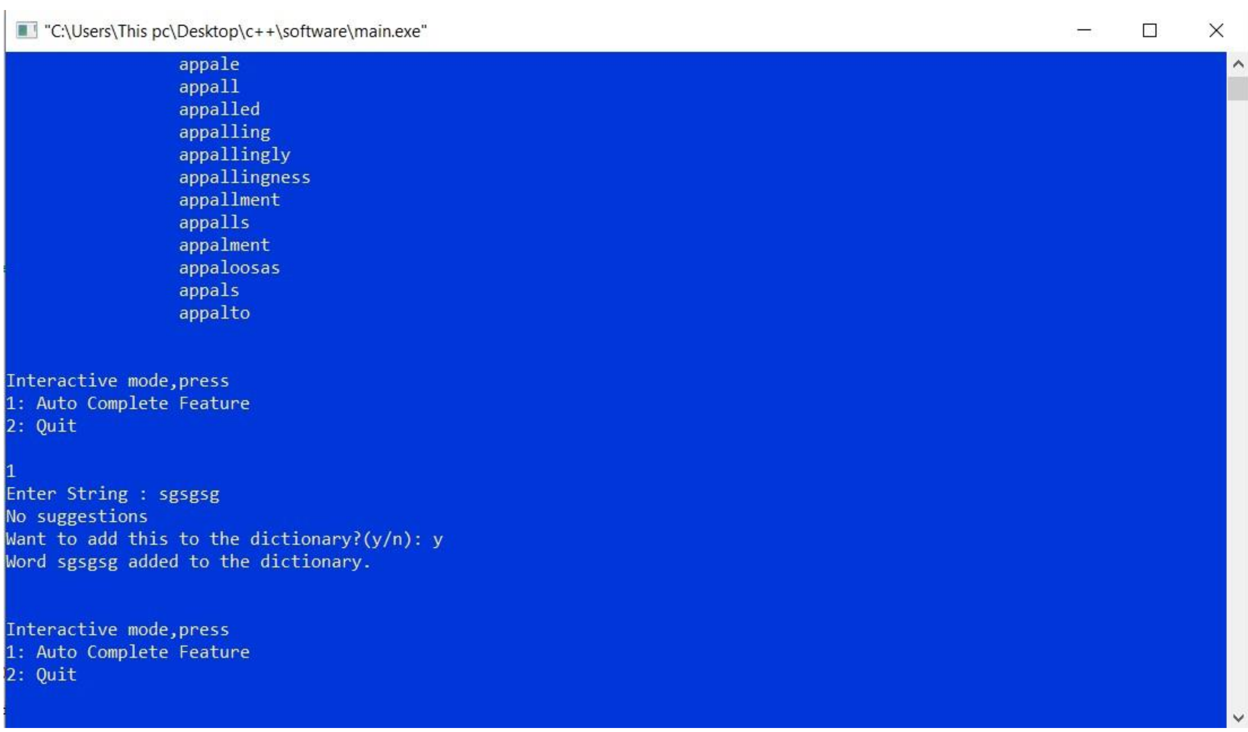
The optimization: The value of s.length() is stored in a variable m and the need to calculate s.length again in the future is eliminated because every time the variable ‘m’ is referred instead.

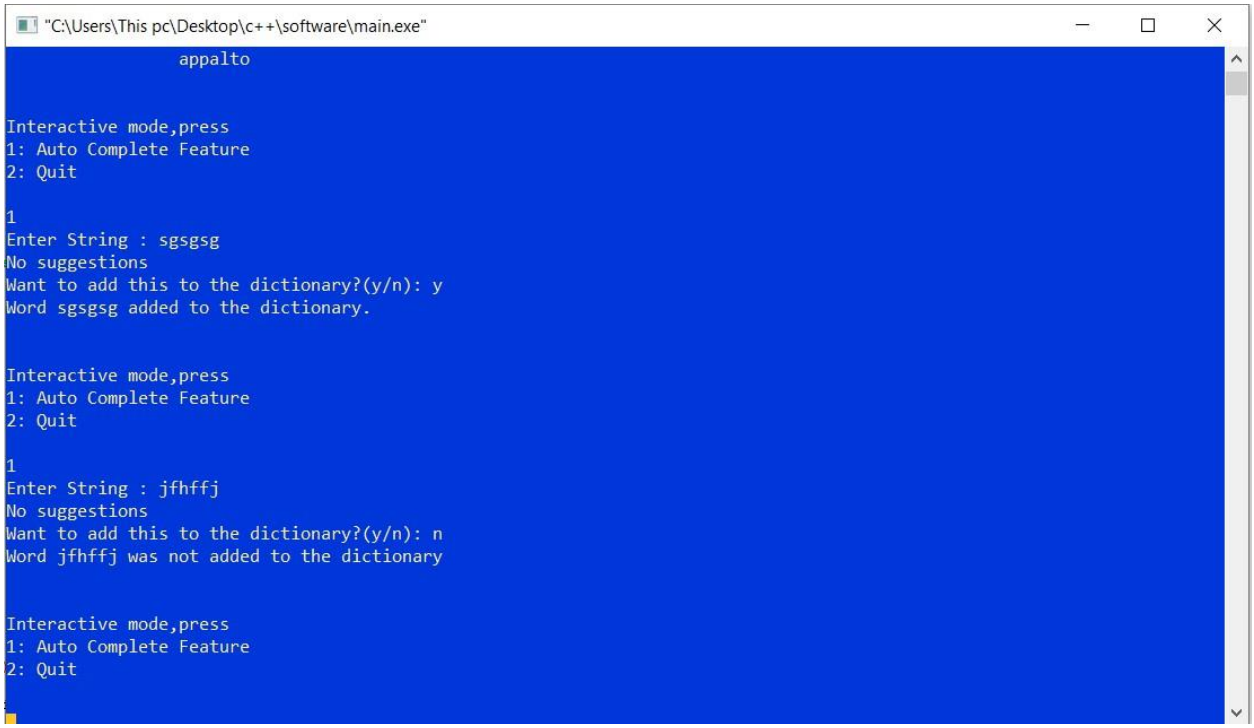
OUTPUT:

First the program asks if user wants to go for ‘1. Autocomplete’ or wants to ‘2. Quit’ the program. If user chooses ‘1’, he’ll get this output:



The program asks for the string. And displays matching words. If the input string does not have any matches in the dictionary file, it shows this:





The user is asked if they want to add the string into the dictionary file. If users will press ‘y’, it asks for a full word that’s to be added. If the user presses ‘n’, the program again asks if they want to ‘1. Autocomplete’ or ‘2. Quit’.

5.2. Testing

5.2.1. Types of Testing

For our algorithm, we performed the blackbox and the whitebox testing. Blackbox Testing is defined as is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths.

Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. The test cases are summarized later.

White Box Testing is a software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. All the code lines, branches and conditions are reachable.

5.2.2. Test cases

Testing:

Test Case	Expected Output	Actual Output	PASS/FAIL
1.On selecting auto complete feature	It should ask for “Enter String”	“Enter String”, appears	PASS
2.On selecting quit	It should end the program	The program ends	PASS
3.Entering a few letters of the word	Should display the list of words		
3.a.”app”	List of all the words starting with “app”	List of all words	PASS
3.b.”hel”	List of all the words starting with “hel”	List of all words	PASS
4.Enter the word which is not in the corpus	Should show a massage ”No suggestion”		PASS
4.a.”serp”	List of all the words starting with “serp”	List of all words	PASS
4.b.”sdrdedf”(jibrish)		“No suggestions”	PASS
5.Ask to add the word to the corpus	Adds word to corpus[y/n]	Add word to corpus[y/n]	PASS
5.a..on typing [y]	The word should be added to the corpus and a message should be presented with “the “word” is added	The “word” is added	PASS
5.b. On typing [n	The word should not be added to the corpus and a message should be presented with “the “word” is not added	The “word” is not added	PASS

6. Conclusion:

Word Predictor has applications in messaging applications like whatsapp, web search engines, word processors, command like interpreters etc. The original purpose of word prediction software was to help people with physical disabilities increase their typing speed,[1] as well as to help them decrease the number of keystrokes needed in order to complete a word or a sentence. Thus, in this front we developed our own program for word prediction using data structure trie which definitely increases efficiency of the user by at least 10%.

Limitations and Scope for future Work

Advantages of our Algorithm:

- A word predictor has applications in various areas like texting, search engines and thus, can help address this issue for a wide demographic.
- Existing programs can be made more efficient using data structures like TRIE.
- Using data structure Trie, word prediction can be made more efficient by at least 10%.
- With data structure Trie, the time complexity for retrieval is reduced and it is much easier to print words in alphabetical order. Both of these are useful for building an efficient and fast word-prediction algorithm.

Limitations of our Algorithm:

- Instead of displaying suggestions when prompted, the algorithm could be tried to make dynamic which displays new suggestions as each new letter is typed.
- The corpus can be made much more extensive.

Future Scope:

- Algorithm can be refined to be more dynamic
- Algorithm can be integrated with software interfaces such as a website
- Algorithm can be worked upon to include prediction for more languages

References

- S. P. Singh, A. Kumar, D. C. Mandad and Y. Jadwani, "Word and phrase prediction tool for English and Hindi language," 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, 2016, pp. 1485-1488, doi: 10.1109/ICEEOT.2016.7754930.
- Trnka, Keith & McCaw, John & Yarrington, Debra & Mccoy, Kathleen & Pennington, Christopher. (2009). User Interaction with Word Prediction: The Effects of Prediction Quality. TACCESS. 1. 10.1145/1497302.1497307.
- Aliprandi, Carlo & Carmignani, Nicola & Deha, Nedjma & Mancarella, Paolo & Rubino, Michele & Srl, Synthema & Pisa, & Italy,. (2008). Advances in NLP applied to Word Prediction.
- Even-Zohar, Yair & Roth, Dan. (2002). A Classification Approach to Word Prediction.
- Singh, Arjun & Stremmel, Joel. (2020). Pretraining Federated Text Models for Next Word Prediction.
- D. Nagalavi and M. Hanumanthappa, "N-gram Word prediction language models to identify the sequence of article blocks in English e-newspapers," 2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), Bangalore, 2016, pp. 307-311, doi: 10.1109/CSITSS.2016.7779376.
- Al-Mubaid, Hisham. (2007). A Learning-Classification Based Approach for Word Prediction. Int. Arab J. Inf. Technol.. 4. 264-271.