**CS250: Data Structure and Algorithm**

**Class: BESE – 5B**

# “Project Report”

# 

# Instructor: Dr. Arsalan Ahmad

# Section: BESE-5B

**Group Member Names:**

**1.** Mohammad Samiullah Bilal. (Reg. # 33076)

**2.** Mohammad Rooshan Naeem. (Reg. # 32659)

**3.** Faizan Zafar. (Reg. # 33125)

**Title of Project:**

Game: **“Spell Checker and Anagram Solver”.**

**Abstract:**

The project is basically a “Spell Checker System” that’ll help user to correctly enter spelling by giving suitable (close match) suggestion i.e. are you looking for this word?..For Example if a user wrongly entered the spelling of Stack as staike or stake, the program will provide suggestions of words along with user desired Word that would be “stack” in above case.

Also in addition to Spell Checker, we’ll introduce word play like “Anagram solver”, in which user will enter some random letters and in return the program will list all the possible words that can be created from the user entered random letters. We’ve put some options like the program will ask user to enter 4,6 or 8 letters words. The program will in return produce the result out of these 4,6 or 8 letter words depending upon user’s choice. Also there is a game in which the program will itself produce some random letters and user has to make all possible words out of the given words. We’ll calculate score and time for each user who would play game and store the result in highest order.

**Introduction:**

There is a usual problem that while writing some document in a hurry ;there is a possible chances that one might enter the spelling of the word wrongly. Taking this as a problem we thought of building something that would take a wrongly spelled word and return as a result some suggested word. Here works our Spell Checker System. Also for increasing Command on words we introduced a little game like Anagram solver.

To overcome the above stated problem we took a whole file of all possible words and loaded it in *TRIE* data structure. And in order to search the word Trie loads all the words in Map and Vector, and using iterator we traverse through words.

**Implementation:**

* **🡪Trie:**
* Tries are used to index and search strings in a text.
* Some examples of tries usage include, finding the longest prefix match in a routing table, auto complete of web addresses in browser etc.
* Trie is a tree where each vertex represents a word or prefix.
* The root represents an empty string.
* Markers are used to indicate end of words.
* A typical node in a trie includes the content (a char), marker for end of word and a collection of children.

Our program uses Trie Data structure to store all the words from a file because time complexity of trie is O (n\*m) and also it is best suited to our desired algorithm. Algorithm is working in a way that whenever a user enters a word our program determines the length of that word and stores all the words from trie into a vector with length less than and greater than that of the entered word by 2. After that our program finds the closest match available to that word available in the vector.

* + **Multimaps and TRIE**

Our Anagram Solver, on the other hand, works on the algorithm such that when a user enter any anagram it is stored in an array. Our program then, determines the length of anagram and uses the trie to store all the words of that length and less than that length in a multimap. They key of every word in the multimap is actually the alphabetical sort of entered word. Our program then, starts matching the character of entered anagram with the keys of multimap. For example, if a word ‘bed’ is stored in a multimap then its alphabetical sort is ‘bde’. Our program compares all the characters of entered anagram with the key of this word and if all the characters of the key are present in that anagram then the word associated with this key will be returned.

**Result:**

* + **TRIE Analysis:**

So in order to store all the possible words we’ve used a Trie Data Structure as Trie data structure is best suited for our problem. Some examples of tries usage include, finding the longest prefix match in a routing table, auto complete of web addresses in browser etc.

The point of a trie is for many queries.

* A trie offers a relatively cheap insertion of each string.
* A trie offers a relatively cheap search for a string.

Suppose you have a dictionary of really long strings, and you'd like to verify if a new very long string is not in it. Then, using hashing, you'd need to go over every letter of the new string. Using a trie, you might be able to rule it out very quickly. The time complexity of trie is

O (n \* m) where n is the number of string and m is the length of highest string.

* + **Vector Analysis**:

A vector is similar to array having a constant running time O (1) to search element with known index. But in our case as we’ve implemented a whole dictionary of n words so making a vector is O (n) all the accessors operation on vector requires O (1).

* + **Multimap Analysis:**

Multimaps are just like Maps except that they can allow multiple keys. The time complexity of multimap is logarithmic in size

**Discussion and conclusions:** To further optimize and refine our algorithm we can use Data structure that has a constant running time i.e. O (1). We can also improve our suggesting algorithm that we can implement in a way that it autocomplete a word while we are writing it like in *Google.* In the end we think the trie is still better not best, of course, structure regarding to our knowledge of data structure and Algorithm so far. And there is always a door open for improvement, so there are lot more option that we could’ve implemented to get more efficient and better result.