English – English Dictionary

GROUP PROJECT

Members: Vinay Kumar (19K-1320)

Om Kumar (19K-0326)

Darpan Kumar (19K-1309)

Abstract:

What we have developed as our project is English to English dictionary with features

- Insert - To add new words
- Search - To search any word
- Delete - To delete present words
- Synonym - Find similar meaning word
- Prefix Search - Search with few letters
- Full Dictionary - Prints full dictionary *ALPHABETICALLY*

For making our dictionary time efficient, in simple words to make it work faster we have used Trie data structure (also known as Prefix tree). Because of Trie data structure efficiency of most used features like search, insert and delete have improved greatly by decreasing complexity to O(m) (where m is just the size of string).

Introduction:

We have developed an English-to-English dictionary. As we have seen in our daily life that we are often searching meaning of different words on daily bases. So, from our daily observation we can simply conclude that the most important and commonly used feature of any dictionary is Searching because of that reason what we really wanted to do, was to make searching efficient.

To program dictionary using Tree was also good option but it gave us time complexity of m*log(n) (where n is number of nodes and m is length of string) which was not good enough. So, we used Trie data Structure which made our dictionary even faster by reducing time complexity to just O(m) (where m is size of string).

Data Structures

After our Implementation:

Searching: The time efficiency of searching increased to just the size of strings.

Deletion: The time efficiency of deletion also increased to just the size of strings.

Insertion: same goes for insertion its time efficiency also increased to just the size strings.

Implementation:

Data Structure Used

- Trie

Utility Data Structure Used

- Vectors
- Strings
- Dynamic Memory

Reason Behind Using Trie:

- Trie is just a tree. But it is more efficient in searching when considering a large dictionary as it's example.
- We will accept only a-z letters as input and insert them in our list.
- Dictionary consists of many inflected forms such as plurals, composite words(such as class -> classroom). Trie helps us to store related words in the same stem.
- Due to the above mentioned property it also helps us in optimizing the efficiency.
- Compared to binary tree where we have maximum depth as ***log2(n)*** **,** Trie has maximum depth equals to the maximum length of word.
- Trie generally uses more memory than trees and lists but it make our program more efficient and optimizes its performance.

Algorithms Used:

- Recursion

Results:

1. Insertion:

   It costs the complexity of O(m)

2. Search:

Searching is in O (m) where m is the size of string/word. As compared to BST implementation of Tree it is much more efficient. BST has complexity of **m* log n**, where m is maximum string/word length and n is number of keys in tree.

3. Deletion:

   Deletion has the complexity of O(m)) where m is size of string/ word.

Conclusion:

Since it is one of the implementation of data structure, we can implement a dictionary using different algorithms and data structures. We have improved time complexity of Searching by using Trie data structure but another concern of ours is memory complexity and when talking about memory, Trie is not best choice. So, what need to be addressed in future is to somehow reduce memory complexity too. However, if storing dictionary words is all that is required (i.e. there is no need to store metadata associated with each word), a minimal deterministic acyclic finite state automaton (DAFSA) or radix tree would use less storage space than a trie. This is because DAFSAs and radix trees can compress identical branches from the trie which correspond to the same suffixes (or parts) of different words being stored.

Real World Applications of TRIE:

- ☞ Predictive Text or Auto Complete, as almost every application of Google uses this features.
- ☞ Spell Checker.
- ☞ Hyphenation.
- ☞ Search Engine results Optimization.
- ☞ Data Analytics.

Work Distribution:

Since this project we used a lot of functions in order to make it much simpler. Following is the distribution of work on based of functions created by each member.

Total Members: 3

1. Vinay Kumar:

Main Functions:

- Prefix Searching
- Print Dictionary Alphabetically

Utility functions:

- Take data from file

Reviewed:

- TRie Header file (for error or corrections)
- UI Header file (for any error or correction)

2. Om Kumar

Main Functions:

- Insertion
- Searching

Utility functions:

- Add data to file

Reviewed:

- Main file (for any error or correction)
- TRie Header file ((for any error or correction)

3. Darpan Kumar

Main Functions:

- Deletion
- Synonym

Utility functions:

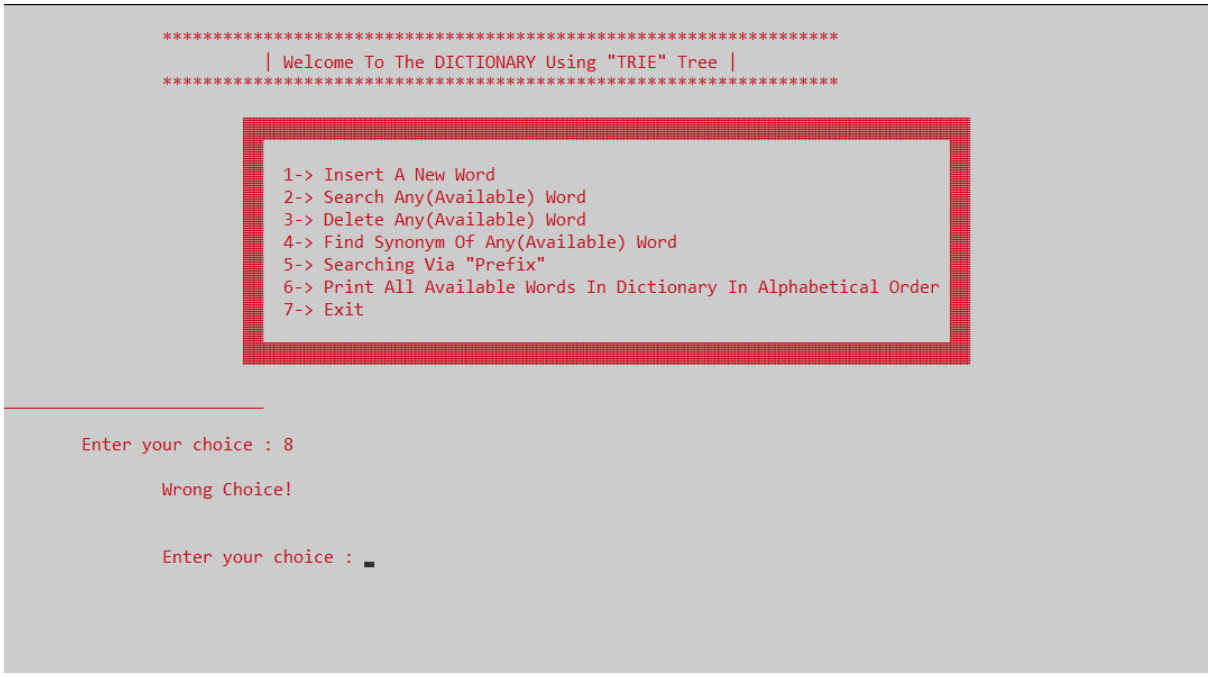
- Take data from file

Reviewed:

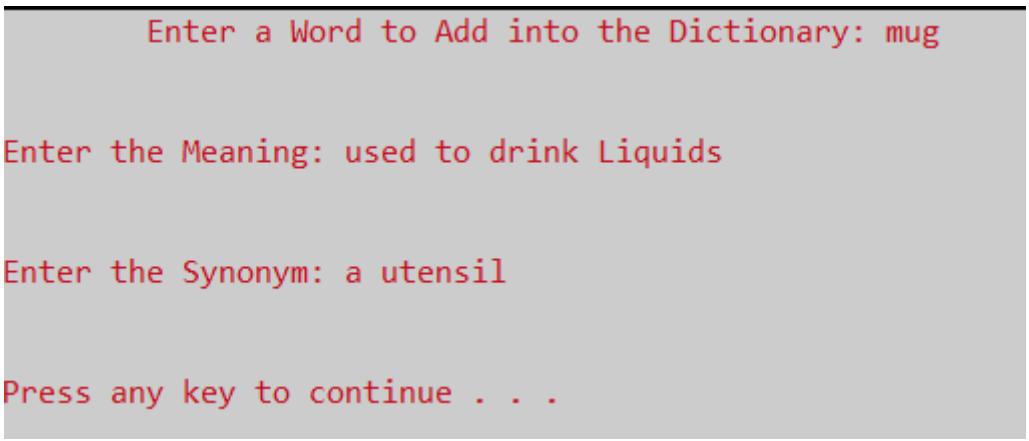
- Main file (for any error or correction)
- TRie Header file (for any error or correction)

Working of Project: (Screen_Shots)

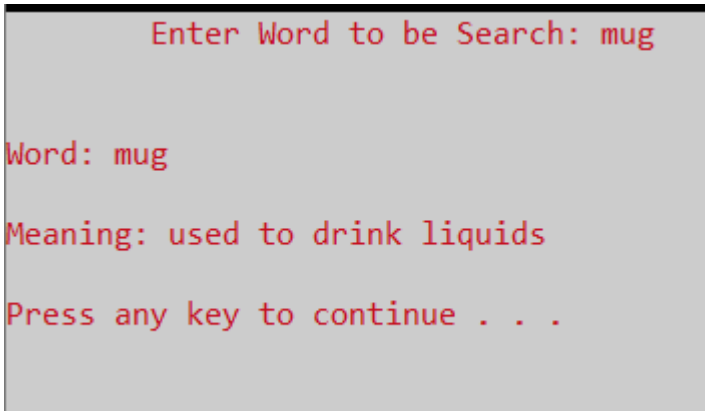
⇨ Opening Screen:



⇨ Insertion:



⇨ Searching:



⇨ **Deletion:**

```
        Enter Word would you like to delete?: mug

SuccessFully Deleted !

Press any key to continue . . .
```

⇨ **Synonym:**

```
        Enter word to Find its Synonym: game

Word: game

Synonym:  tournament, sport

Press any key to continue . . .
```

⇨ **Prefix Searching:**

```
        Enter Word would you like to use as a prefix: dar

dark

darpan

Press any key to continue . . .
```

Data Structures

⇨ Print Dictionary Alphabetically:

```
        All The Words of Dictionary

        ->  a
        ->  abascus
        ->  ability
        ->  able
        ->  about
        ->  above
        ->  accept
        ->  according
        ->  account
        ->  across
        ->  act
        ->  action
        ->  actually
        ->  add
        ->  address
        ->  administration
        ->  admit
        ->  adult
        ->  affect
        ->  after
        ->  again
        ->  age
        ->  agency
        ->  ago
        ->  american
        ->  animal
        ->  b
        ->  baby
```

⇨ Closing Screen:

```
                ****************************************************************
                                        | THANK-YOU |
                ****************************************************************


--------------------------------
Process exited after 67.06 seconds with return value 0
Press any key to continue . . .
```