Predictive Text Using Auto Complete

Student:

Akshay Hitesh Dhruv

Professor:

Dr. Lijun Yin

Course:

Design and Analysis of Computer Algorithm

CS 575-01

Problem Statement

Here, I have implemented auto complete in a game. The player gets an ordered set of alphabets called HEL. The player needs to use those starting alphabets and make a word. Any word in the English language starting with HEL may or may not exist. If there doesn't exist any such word, then he must use the maximum possible string from HEL starting from its first letter. If he is not able to make a word with HEL but such a word exists in the English language, then he is awarded zero points. If he can make any such word, then he'll be given points according to the letters in the word. Now John wants his friend Sam who is a programmer to make a program that helps him find the possible words so that he can choose from them.

Objectives

- To optimize auto-completion of the word.
- To optimize Depth First Search by improving the traversal of data structure by special algorithm and analysis techniques.
- To Search and Delete Word in the Trie.
- Modify the default structure of Trie and with certain tweaks improve the overall time complexity.
- We substantially increased the number of words in the Trie from 4 to 58000 words and it worked correctly.

Algorithm Techniques

- For printing all possible words in a trie with given root: Backtracking Traversal (like DFS).
- For searching and deleting the prefix in trie: naïve algorithm.
- For insertion of new words in trie: Recursion.

Algorithm Description

Auto Complete is the most important feature and essential tool when it comes to accessing web search engines such as Google, Yahoo, and YouTube etc. Although it is intensively used on internet. It features in many offline activities as well. Finest example can be smart phones. I have implemented auto complete using Trie and analyzed the time complexity of this algorithm.

Trie data structure is a tree with each node consisting of one letter as data and pointer to the next node. It uses Finite Deterministic automation. That means, it uses state to state transition. Words are paths along this tree and the root node has no characters associated with it. The value of each node is the path or the character sequence leading up to it. I have used a file-based database as data store for storing the words into the Trie. When we type few letters of a word, we can do a regular expression match each time through the wordlist text file and print the words that starting the letters. It's also useful to set a flag at every node to indicate whether a word/phrase ends there.

But you may be asking yourself, "Why use trie if set <string> and hash tables can do the same?" There are two main reasons:

First, with Trie, we can insert and find strings in O(M) time where M represent the length of a single word. This is obviously faster than Binary Search Tree. This is also faster than Hashing because of the ways it is implemented. We do not need to compute any hash function. No collision handling is required. Another advantage of Trie is, we can easily print all words in alphabetical order which is not easily possible with hashing. We can efficiently do prefix search (or auto complete) with Trie. The next figure shows a trie with the word's "their", "there", "this", "that", "does", and "did".

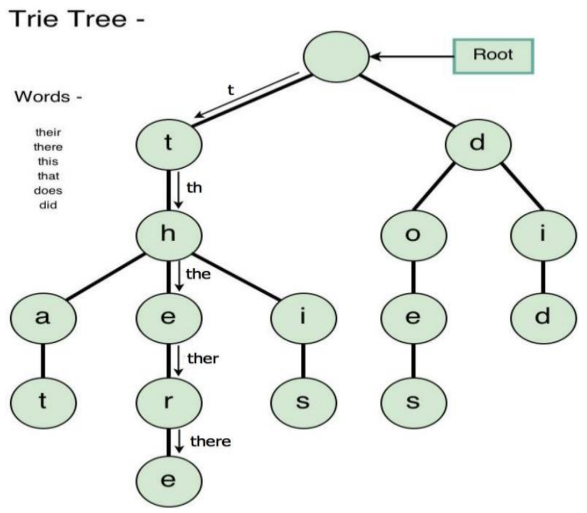
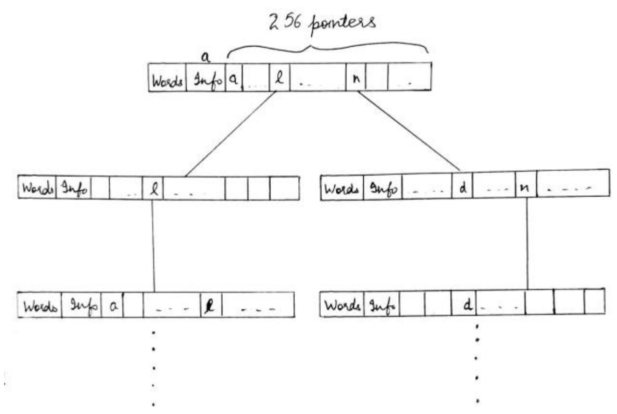


Figure. Structure of a Trie Tree

Pseudo Code:

```
Begin
function insert() :
   If key is not present, insert keys from file into trie. If the key is prefix of trie node, just mark leaf node.
End
function print()
  If user specified length matching words present along with the key passed.
      Then travel through the tree using DFS algorithm and print all possible combinations.
End
Begin
Function search()
   If user specified key is present or the possible combinations.
        Then use naïve algorithm to match
   If the key is matched
        Then set the flag to 1 and pass it to print
   Otherwise
        There is no matching key
End
Begin
function deleteNode()
   If tree is empty then return null.
   If last character of the key is being processed,
      Then that node will be no more end of string after deleting it.
      If given key is not prefix of any other string, then delete it and set root = NULL.
   If key is not the last character,
      Then recur for the child which will be obtained by using ASCII value.
   If root does not have any child left and it is not end of another word,
      Then delete it and set root = NULL.
End
Begin
```

Asymptotic Analysis



Space Complexity:

In worst case analysis, the wordlist will consist of words with no common prefix. At every level, 256 combinations will be possible from each node. So, this will result in a lot of memory usage.

Maximum total number of nodes in a trie with m as maximum length of word is:

$256+256^2+256^3+256^4+\ldots\ldots\ldots\ldots\ 256^m$

$= (256(256^{(m)+1} - 1))/ (256-1)$

$= (256/255) (256^{(m+1)} - 1)$

$= O(256^{(m)})$

So trie will have exponential space complexity in worst case complexity.

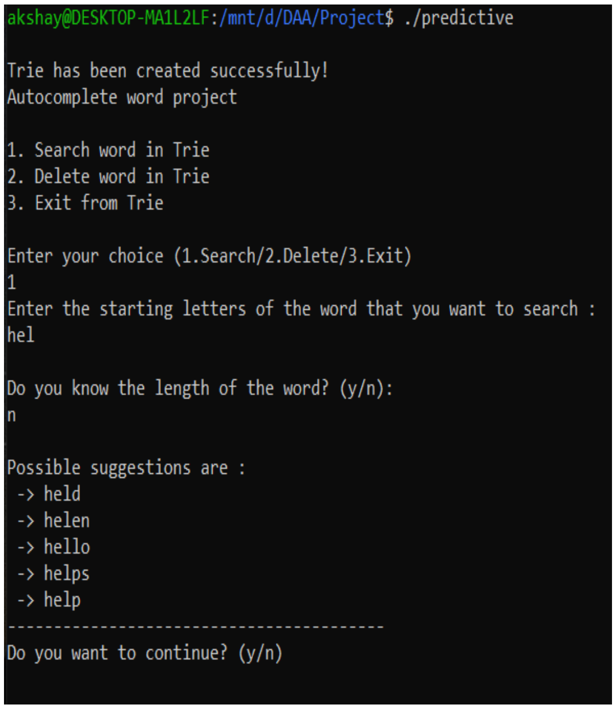
Time Complexity:

The key determines trie depth. Trie will have linear time complexities in all the cases. So, it is the most efficient algorithm for implementing trie data structure despite its large space complexity.

Using trie, we can search / insert / delete the single key in O(M) time. Here, M is maximum string length.

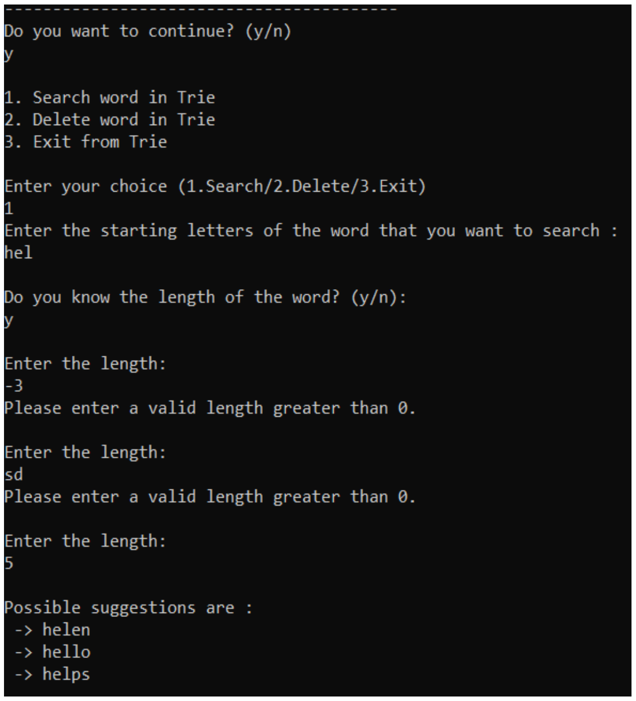
In this project, I got the search result in O(key_length + Σ(L)), where key_length is input given by user and Σ(L) is summation of all the string lengths starting with prefix entered by user.

Results



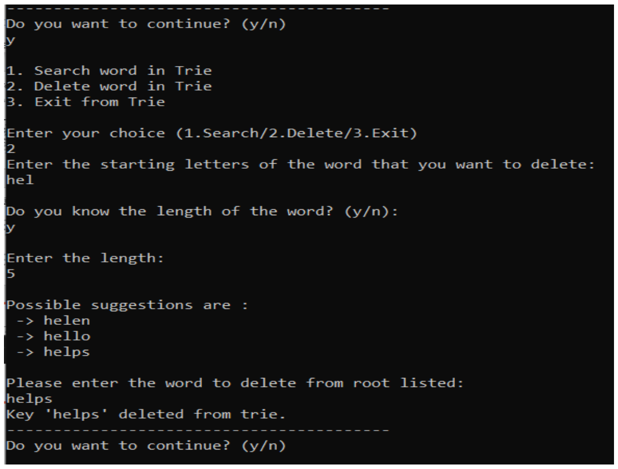
```
akshay@DESKTOP-MA1L2LF:/mnt/d/DAA/Project$ ./predictive

Trie has been created successfully!
Autocomplete word project

1. Search word in Trie
2. Delete word in Trie
3. Exit from Trie

Enter your choice (1.Search/2.Delete/3.Exit)
1
Enter the starting letters of the word that you want to search :
hel

Do you know the length of the word? (y/n):
n

Possible suggestions are :
 -> held
 -> helen
 -> hello
 -> helps
 -> help
----------------------------------------
Do you want to continue? (y/n)
```

Figure 1: Output when user doesn't know length.



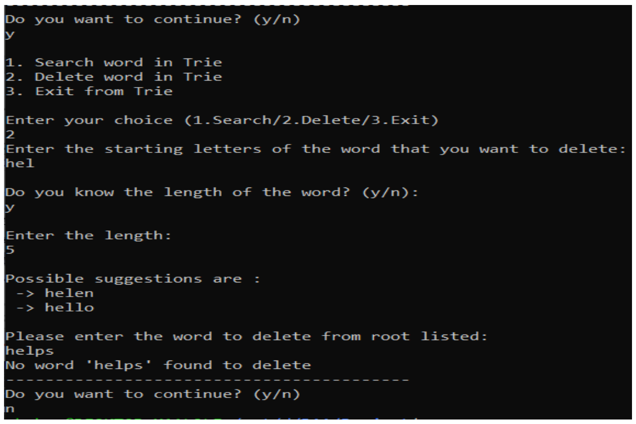
```
----------------------------------------
Do you want to continue? (y/n)
y

1. Search word in Trie
2. Delete word in Trie
3. Exit from Trie

Enter your choice (1.Search/2.Delete/3.Exit)
1
Enter the starting letters of the word that you want to search :
hel

Do you know the length of the word? (y/n):
y

Enter the length:
-3
Please enter a valid length greater than 0.

Enter the length:
sd
Please enter a valid length greater than 0.

Enter the length:
5

Possible suggestions are :
 -> helen
 -> hello
 -> helps
```

Figure 2: Output when user inputs non-positive length.



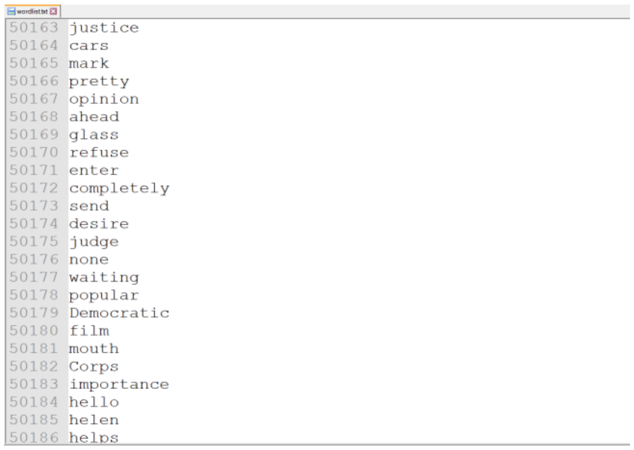
```
----------------------------------------
Do you want to continue? (y/n)
y

1. Search word in Trie
2. Delete word in Trie
3. Exit from Trie

Enter your choice (1.Search/2.Delete/3.Exit)
2
Enter the starting letters of the word that you want to delete:
hel

Do you know the length of the word? (y/n):
y

Enter the length:
5

Possible suggestions are :
 -> helen
 -> hello
 -> helps

Please enter the word to delete from root listed:
helps
Key 'helps' deleted from trie.
----------------------------------------
Do you want to continue? (y/n)
```

Figure 3: Output when user wants to delete.



```
Do you want to continue? (y/n)
y

1. Search word in Trie
2. Delete word in Trie
3. Exit from Trie

Enter your choice (1.Search/2.Delete/3.Exit)
2
Enter the starting letters of the word that you want to delete:
hel

Do you know the length of the word? (y/n):
y

Enter the length:
5

Possible suggestions are :
 -> helen
 -> hello

Please enter the word to delete from root listed:
helps
No word 'helps' found to delete
----------------------------------------
Do you want to continue? (y/n)
n
```

Figure 4: Output after user has deleted.



```
justice
cars
mark
pretty
opinion
ahead
glass
refuse
enter
completely
send
desire
judge
none
waiting
popular
Democratic
film
mouth
Corps
importance
hello
helen
helps
```

Figure 5: The dictionary

Conclusion

- John's problem of finding the desired solution was solved.
- Auto complete was successfully implemented using Trie.
- Predict words if words of desired length are not available.
- DFS and Trie's implementation was understood.

Contribution to Society

Auto Complete speeds up human-computer interactions when it correctly predicts words being typed. It works best in domains with a limited number of possible words (such as in command line interpreters), when some words are much more common (such as when addressing an e-mail) or writing structured and predictable text (as in source code editors).

References

- https://www.topcoder.com/community/data-science/data-science-tutorials/using-tries/
- http://igoro.com/archive/efficient-auto-complete-with-a-ternary-search-tree/
- https://www.geeksforgeeks.org/trie-delete/
- https://en.wikipedia.org/wiki/Autocomplete
- https://en.wikipedia.org/wiki/Trie
- http://www.geeksforgeeks.org/trie-insert-and-search/
- https://medium.com/basecs/trying-to-understand-tries-3ec6bede0014