PES UNIVERSITY, BENGALURU

FINAL SEMESTER ASSESSMENT (FSA) – B.TECH. (CSE) – IV SEM

SESSION: JANUARY –MAY, 2019

UE17CS251 – DESIGN AND ANALYSIS OF ALGORITHMS

Project Report

on

"Spell checker using trie trees in C, with an auto-suggest feature"

Submitted by

Siddarth Karki	PES1201700120
Anshuman Pandey	PES1201700255

Name of the Examiners	Signature with Date
1._____	_____
2._____	_____
3._____	_____
4._____	_____

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PES UNIVERSITY

(ESTABLISHED UNDER KARNATAKA ACT NO. 16 OF 2013)

100 FEET RING ROAD, BENGALURU – 560 085, KARNATAKA, INDIA

About

This project is a spell checker implemented in C using the trie data structure.

It also has an additional auto suggest feature, whose implementation has been explained in detail in the upcoming sections.

The Trie Data Structure

Trie is an efficient information re*trie*val data structure. Using Trie, search complexities can be brought to optimal limit (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 tree. Using Trie, we can search the key in O(M) time. However the penalty is on Trie storage requirements.

Every node of Trie consists of multiple branches. Each branch represents a possible character of keys. We need to mark the last node of every key as end of word node. A Trie node field *isEndOfWord* is used to distinguish the node as end of word node. The data structure used in the program is:

```
#define ALPHABET_SIZE 26
//the trie data strcuture
struct TrieNode
{
   struct TrieNode *children[ALPHABET_SIZE];
   int isEndOfWord;
};
```

Input

The program takes in two input files, one is the dictionary(dict.txt) using which it checks the legality of the words in the other input file (input.txt), the later is given by the user.

Output

The output(output.txt) produced by the program is the line-by-line errors in the input file.

Algorithms

//to insert a word in a trie tree

//input : a string key[0....n-1] and the root of the trie tree

//output: The trie with the key inserted in it

```
Insert_trie(key[0..n-1], root):
        string_length <- strlen(key)
        temp_root <- root
        for i<-0 to i<-(n-1):
                if temp_root->children[char_to_index(key[i]]==Null:
                        temp_root->children(char_to_index[key[i]]= get_node
                        temp_root <- temp_root->children[char_to_index(key[i]]
                else
                        temp_root <- temp_root->children[char_to_index(key[i]]
        temp_root-> IsEndOfWord -> true
```

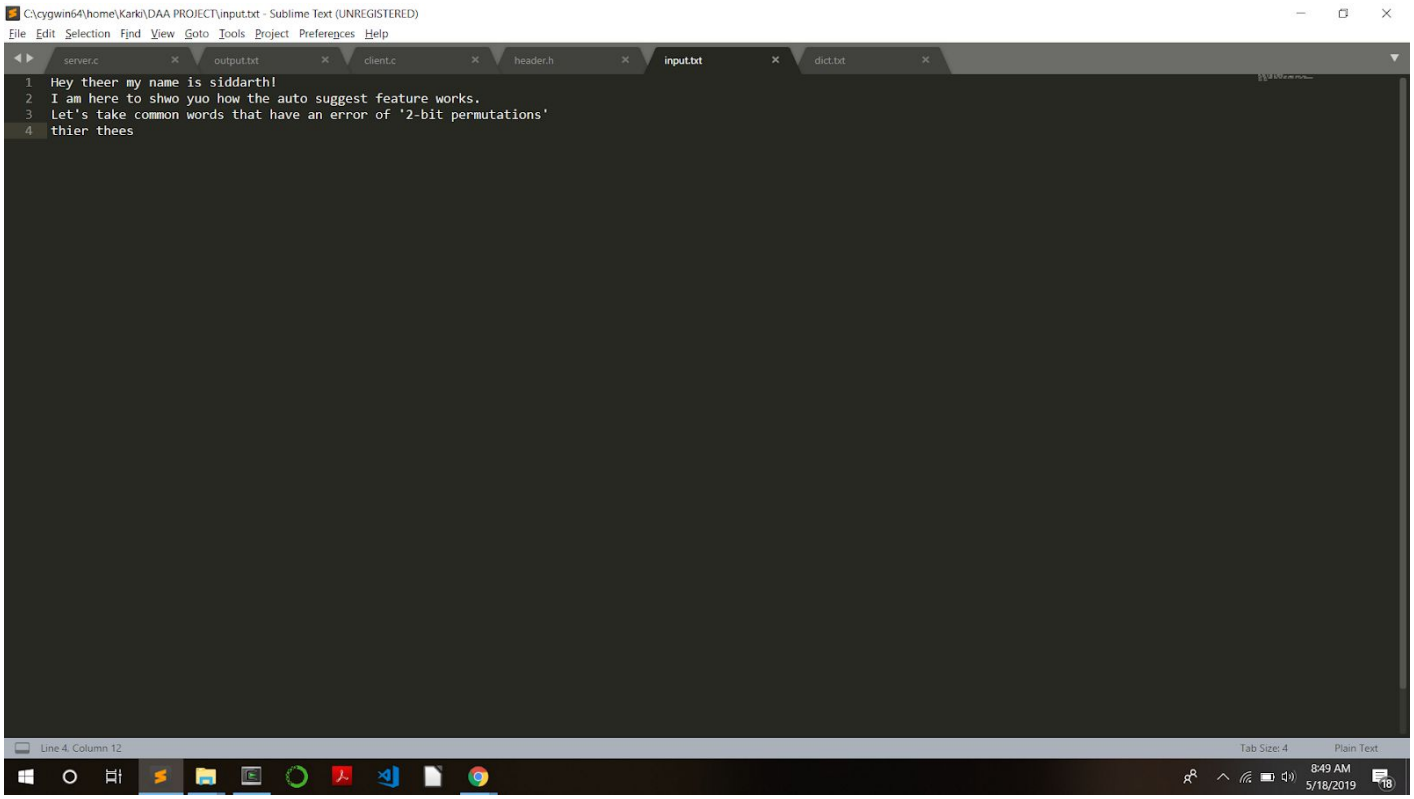
//to search in a trie tree

//input : a string key[0....n-1] and the root of the trie tree

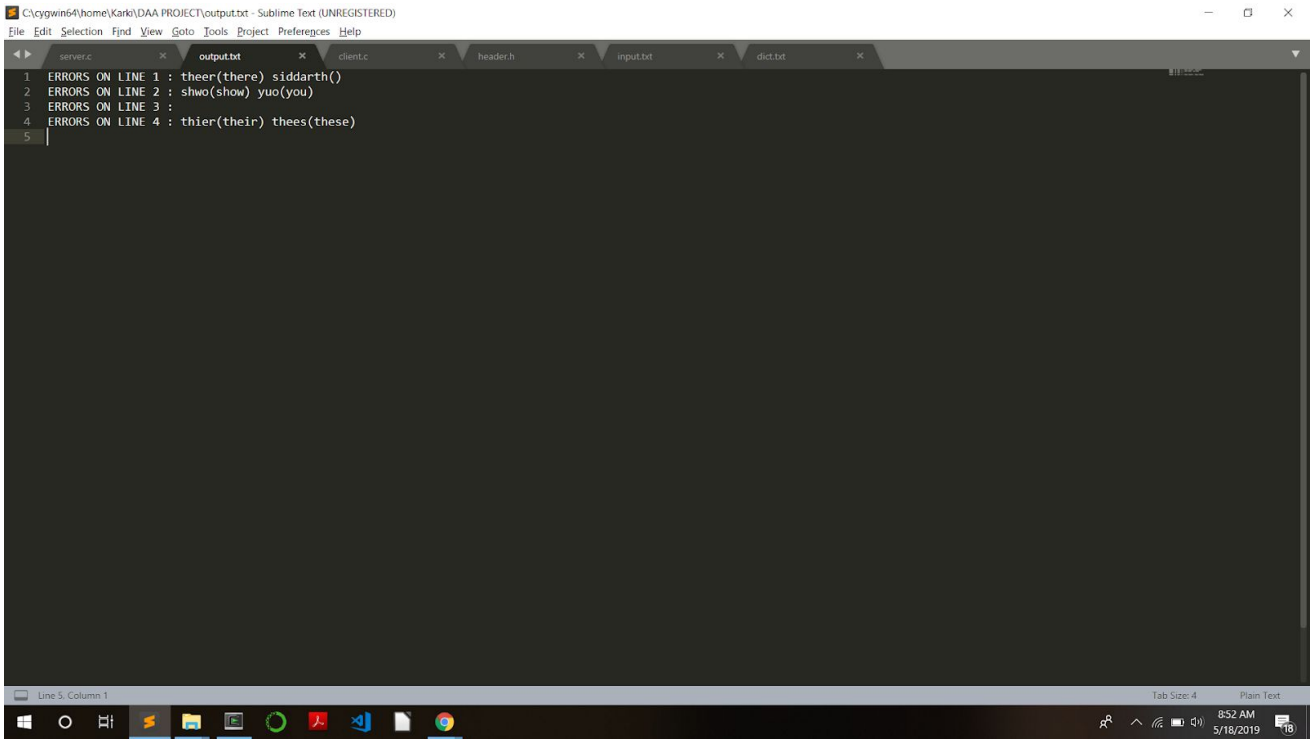
```
//output: 1 if found in the trie, 0 otherwise


search_trie(string key[0...n-1], root):
        temp_root <- root
        for( i<-0 to len(key)-1):
                if(temp_root->children[char_to_index(key[i])] = NULL)
                        return 0;
                else
                        temp_root <- temp_root->children[char_to_index(key[i]]
        return (temp_root!=null and temp_root->IsEndOfWord= true)


//to find the permutation of a wrong word
//input : a string key[0....n-1] and the root of the trie tree
//output: the permutation of the string that exists in the trie, empty string if //there's no such string


check_permutations(string key[0...n-1], root):
        for(i<- 0 to len(key)-1):
                temp_key = key
                swap temp_key[i] to the characters adjacent to it.
                check if the resultant temp_key is in trie using search function
                if there, return the temp_key after swapping
        continue
        return ""
```

Snapshots of input and output file:

INPUT FILE (INPUT.TXT):



OUTPUT FILE (OUTPUT.TXT)



GITHUB LINK

HTTPS://GITHUB.COM/KARKI23/SPELLCHECKER

****