ALGORITHM PROJECT PEC CHANDIGARH

A SPELL CHECKER

(IN ENGLISH LANGUAGE)

SUBMITTED BY: -

PIYUSH SHARMA (17103110)

MEGHA SHARMA (17103101)

VIKRAM (17103088)

**CONTENTS**

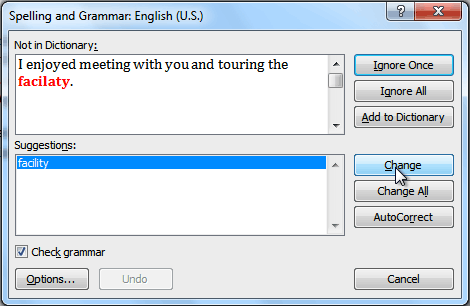
|  |  |  |
| --- | --- | --- |
| 1. | Objective | 1 |
| 2. | Literature Review | 2 |
| 3. | Why TRIE Data Structure? | 3-4 |
| 4. | Problem Statement | 5 |
| 5. | Solution Approach | 6 |
| 6. | Implementation C++ Code | 7-11 |
| 7. | Future Work | 12 |
| 8. | Applications | 13 |
| 9. | References | 14 |

**OBJECTIVE**

The most common & simplest tool which can assist the user with writing is a spell checker. However, the function of a spell checker is simple, input text and incorrect words will be identified.

The main aim of this project is to look at how a spell checker can be built to detect real word errors by using the Trie Data Structure and File Handling approach.

The size and content of the dictionary that is being used for checking the input can be altered for required purposes.



**LITERATURE REVIEW**

**A Comparison of Dictionary Implementations** by Mark P Neyer analyses the various methods by which insertions and mappings are possible in a dictionary, crucial for the implementation of the insert find and delete functions. Comparison was done between AVL tree, red black tree, skip lists and hash tables. The findings were that hash table was superior in performance in comparison to the trees and the skip lists.

**Spell Checker** by Vibhakti V. Bhaire, Pradnya A.Pashte and Mr Magdum P.G Rajendra Mane College of Engineering and Technology is a report about a spell checker which uses edit distance algorithm for it’s functioning. The resultant spell checker had the advantages of easy identification, good accuracy, identification of most spelling and typing errors but suffered from inability to differentiate homonyms (words with same sound) and identified words as mistakes if they were spelt using a different language form other than the program default.

**WHY TRIE DATA STRUCTURE?**

Interestingly, there are many data structures that can be used to implement a Spell Checker. But main of them in which we got confused were:

1. AVL Tree
2. Skip Trees
3. Hash Tables
4. TRIE Data Structures

However, inserting a value into an AVL tree often requires a tree search for the appropriate location along with the re-balancing after insertion. Both the re-balancing algorithm and the binary

search take O(log n) time. Also, the searching algorithm also takes O(log n) time.

Also skip lists, that consists of a parallel sorted linked lists, takes O(log n) time for inserting and searching of an item. But we want both these operations to happen in the smallest time possible as they will be most commonly used.

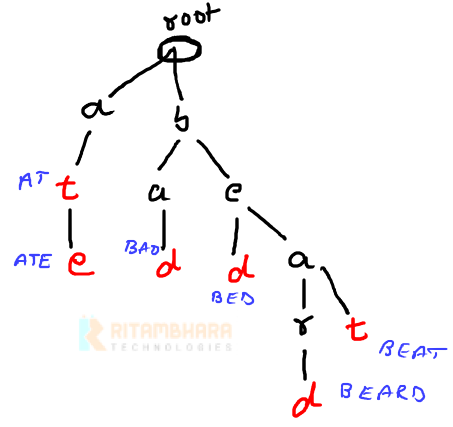
Insertion in Hashing takes O(1) time, so once we thought that this is the perfect data structure that we’ve been waiting for. But the story didn’t end up here.

Scrolling down stack overflow, we came across a comparison between hashing and TRIE data structure, which eventually changed our whole implementation of code.

We realised that Hashing won’t support operations like prefix search. i.e. the code dictionary shows all words starting with that prefix. Hashing also doesn’t support efficient printing of all words in dictionary in alphabetical order and nearest neighbour search.

In both these operations, **Trie** is suited. With Trie, we can support all operations like insert, search, delete in O(n) time where n is length of the word to be processed. Also, the scope of further work on our topic was greater in the case in which we were using TRIE data structure (as seen from our point of view).

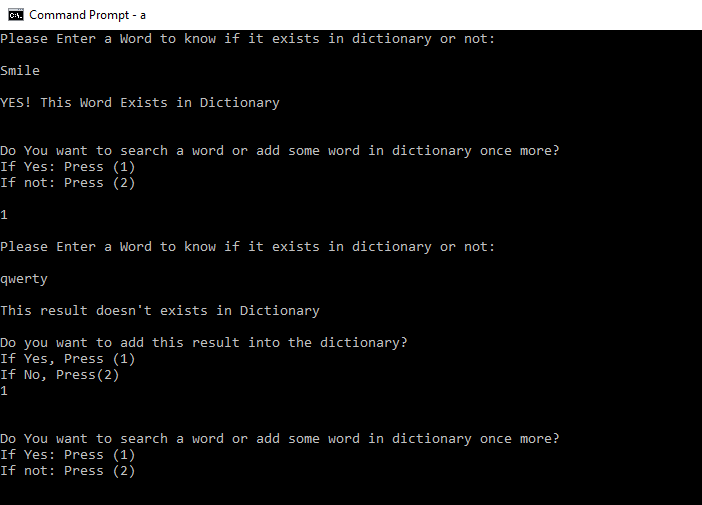
All these points combined together made us use the TRIE data structure for implementing this programme.



**PROBLEM STATEMENT**

The problem is quite simple, given a word by the user, you need to know whether this word in dictionary is correct or not. If the dictionary doesn’t contain that word, the user has an option to insert the word into the dictionary for further use.

***SAMPLE OUTPUT: -***



**SOLUTION APPROACH**

* Language used for Implementation is C++
* File used for importing data into C++ program is dictionary.txt file. This file contains every word that I imported into Trie data structure for further analysis.
* Dictionary.txt file contains words that would exist into a regular dictionary.
* Then each one of these words are correspondingly put into their respective Trie data structure branch using insert function.
* Every new node of Trie Data Structure contains 26 pointers as there are 26 alphabets in English language and one Boolean variable endofword(EOF).
* The input dictionary used by us actually contains all the words into lower case alphabets. Thus, the input given by the user is firstly transformed into lower case by using the transform function.
* Depending on whether the word exists in dictionary or not, it searches the word or inserts the word into the trie as well as into the dictionary of the file imported on the basis of the decision given by the user.
* The user may continue to search more words or end the programme.

**IMPLEMENTATION OF CODE**

#include <bits/stdc++.h>

#include<iostream>

#include<cstring>

#include<fstream>

using namespace std;

const int total=26;

struct node

{

struct node\* arr[total];

bool endofword;

};

struct node\* makenewnode()

{

struct node\* newnode=new node;

for(int i=0; i<total; i++)

{

newnode->arr[i]=NULL;

}

newnode->endofword=false;

return newnode;

}

void insert(struct node \*masterroot, string key)

{

struct node\* walker=masterroot;

int n=key.length();

for(int i=0; i<n; i++)

{

int index=key[i]-'a';

if(!walker->arr[index])

{

walker->arr[index]= makenewnode();

}

walker=walker->arr[index];

}

walker->endofword=true;

}

bool search(struct node \*masterroot, string key)

{

struct node\* walker=masterroot;

int n=key.length();

for(int i=0; i<n; i++)

{

int index=key[i]-'a';

if(!walker->arr[index])

return false;

else

walker=walker->arr[index];

}

if(walker->endofword==true)

return true;

return false;

}

int main()

{

struct node\* masterroot=makenewnode();

string line;

ifstream fin;

fin.open("dictionary.txt");

while(!fin.eof())

{

getline(fin,line);

insert(masterroot,line);

}

fin.close();

int i=1;

string word;

cout<<"\nWELCOME TO PIYUSH-MEGHA-VIKRAM SPELL CHECKER\n";

while(i==1)

{

cout<<"\nPlease Enter a Word to know if it exists in dictionary or not: \n\n";

string key;

cin>>key;

transform(key.begin(), key.end(), key.begin(), ::tolower

bool result;

result=search(masterroot,key);

if(result==true)

{

cout<<"\nYES! This Word Exists in Dictionary\n";

}

else

{

cout<<"\nThis result doesn't exists in Dictionary\n";

int u=1;

while(u)

{

cout<<"\nDo you want to add this result into the dictionary? \nIf Yes, Press (1) \nIf No, Press(2)\n";

int s;

cin>>s;

if(s==1)

{

ofstream fout;

fout.open("dictionary.txt",ios::app);

fout<<"\n"<<key<<"\n";

insert(masterroot, key);

u=0;

}

else if(s==2)

u=0;

else

cout<<"\nPlease Enter a Valid Input! ENTER AGAIN\n";

}

}

int x=1;

while(x)

{

cout<<"\n\nDo You want to search a word or add some word in dictionary once more?\nIf Yes: Press (1) \nIf not: Press (2)\n\n";

int ch;

cin>>ch;

if (ch==1)

x=0;

else if (ch==2)

{

x=0;

i=0;

}

else

cout<<"Please Enter a Valid Input! ENTER AGAIN \n\n";

}

}

}

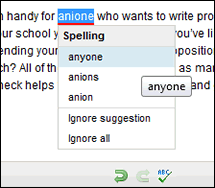
**FUTURE WORK**

The program can be easily adjusted due to flexibility resulting from file handling application by changing the backend files i.e. the dictionary and can also be extended to a spell corrector which also gives the nearest correct words to the input by the user. We can extend our project’s implementation to a way in which the code also gives the antonyms and synonyms of the word that is input by the user. It can also work for various languages provided a suitable dictionary file at backend is provided and suitable input provided.

**APPLICATIONS**

**Based on Concept of SPELL CHECKER**

* The concept of Spell Checker can be used to detect typing errors, and in some cases may also give the corrected word suggestions and will be known as Spell Corrector.
* Almost every person while using social networking sites like Facebook, Gmail, Twitter etc uses spell checkers for typing texts or email.
* Microsoft's experts in languages also track word requests, as well as frequently corrected 'words,' to assess whether those words should be added to the Speller dictionary.
* This is used by students to make their projects and presentations flawless. Producing flawless copy is a mark of professionalism into the student and most companies expect nothing less in their documentation.
* People running spell checkers on their files rather than using dictionary are also seen to save enough of their time in working.
* One main benefit of using a spell checker is its Accuracy. Running a spell checker ensures that the number of typos in your document decreases significantly.



**REFERENCES**

* <https://stackoverflow.com/questions/53231921/spell-checker-in-c>
* <http://www.cs.unc.edu/~plaisted/comp550/Neyer%20paper.pdf>
* <http://norvig.com/spell-correct.html>
* <https://en.wikipedia.org/wiki/Trie>
* <https://www.geeksforgeeks.org/data-structure-dictionary-spell-checker/>
* <https://www.youtube.com/watch?v=tc7CjrhdAaY&index=50&list=PLLYz8uHU480j37APNXBdPz7YzAi4XlQUF>
* <https://www.youtube.com/watch?v=hCkfBhaW6E0&list=PLLYz8uHU480j37APNXBdPz7YzAi4XlQUF&index=51>
* <https://www.youtube.com/watch?v=CpFx-5J7D7s&list=PLLYz8uHU480j37APNXBdPz7YzAi4XlQUF&index=49>