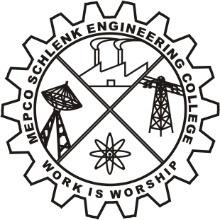
**TRIE DATASTRUCTURE**

**DICTIONARY**

##### A MINI PROJECT REPORT

###### ***Submitted by***

**DINESH KUMAR S (9517202109016)**

**MOHAMMED ASLAM K (9517202109034)**

**ABISHEAK A (9517202109004)**

**MANOJ S (9517202109032)**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**IN**

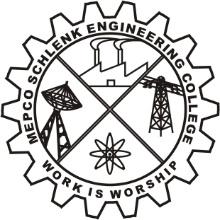
**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**

**ANNA UNIVERSITY : CHENNAI 600 025**

**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**

**AUTONOMOUS**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**BONAFIDE CERTIFICATE**

This is to certify that it is the bonafide work of “**Dinesh Kumar S (9517202109016), Manoj S(9517202109032), Abisheak A(9517202109004), Mohamed Aslam K(9517202109034)”** for the mini project titled **“Trie datastructure dictionary”** in 19IT351 - Data Structures Laboratory during the third semester August 2022 - December 2022 under my supervision.

**SIGNATURE SIGNATURE**

**Dr. P.Thendral, M.E.,Ph.D Dr.J.Angela Jennifa Sujana, M.E.,Ph.D**

**Assistant Professor(SG) Associate Professor(SG) & Head**

Artificial Intelligence and Data Science, Artificial Intelligence and Data Science ,

Mepco Schlenk Engineering College, Mepco Schlenk Engineering College,

Sivakasi- 626 005,Virudhunagar. Sivakasi- 626 005,Virudhunagar.

Submitted for the project viva-voce examination to be held on \_\_\_\_\_\_\_\_\_\_\_\_\_.

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

**ACKNOWLEGEMENT**

First and foremost we **praise and thank “The Almighty”,** the lord of all creations, who by his abundant grace has sustained us and helped us to work on this project successfully.

We  really  find  unique  pleasure  and  immense  gratitude  in  thanking our respected management members**,** who is the backbone of our college.

A deep bouquet of thanks to respected Principal **Dr.S.Arivazhagan  M.E.,Ph.D.,** for having provided the facilities required for our mini project.

We sincerely thank our Head of the Department  **Dr. J. Angela Jennifa Sujana M.E.,Ph.D.,** Associate Professor(SG) & Head, Department of Artificial Intelligence and Data Science, for her guidance and support throughout the mini project .

We also thank our guide **Dr.P.Thendral.,M.E.,Ph.D.,** Assistant Professor(SG), Department of Artificial Intelligence and Data Science for their valuable guidance and it is great privilege to express our gratitude to them.

We extremely thank our project coordinator **Dr.P.Thendral.,M.E.,Ph.D.,** Assistant Professor(SG),Department of Artificial Intelligence and Data Science, who inspired us and supported us throughout the mini project.

We extend our heartfelt thanks and profound gratitude to all the faculty members of Artificial Intelligence and Data Science department for their kind help during our mini project work. We also thank our parents and our friends who had been providing us with constant support during the course of the mini project work.

**TABLE OF CONTENTS**

| **S. no** | **Title** | **Page no.** |
| --- | --- | --- |
|  | Abstract  Objective | 5  6 |
| 1 | **Introduction**   * 1. Overview | 7 |
| * 1. Issues | 7 |
| 2 | Problem Statement | 8 |
| 3 | Modules  3.1 Structure definition  3.2 Load Function  3.3 Unload Function  3.4 Search Function  3.5 View Function  3.6 Add Function | 8  9  10  11  11  12 |
| 4 | Source Code | 14 |
| 5 | Result  5.1 Preview  5.2 View Result  5.3 Add Result  5.4 Search Result  5.5 Exit Result | 19  19  20  20  21 |
| 6 | Conclusion | 21 |

**ABSTRACT**

The purpose of Trie data structure is to speed up the process of the searching of a particular word in a group of words. The required software and hardware are easily available and easy to work with. The aim of this project is to create and implement the Trie data structure using the language c and perform the search engine operation with help of this Trie data structures and also count the number of results found.

It can lead to an error free and fast retrieval of the particular data we are searching.

In the traditional way of searching a word by humans in the means of skimming takes a lot a time to find a particular word. This trie will reduce the time of searching by computerizing all the data.

**OBJECTIVES :**

* To create a trie data structure using c.
* To perform a search engine operation with the use of the created trie data structures.
* To use the various features of the language C .
* To create a structure for creating trie.
* To made a runtime memory allocation and store the data using the malloc inbuilt function.
* To learn about the creation and implement the Trie data structure.

**INTRODUCTION**

# OVERVIEW

In the traditional method of searching a word in the group of word takes more time to found a particular word. So, we created this mini project to computerize the searching process of the word. To count the number of occurrence of the particular word in the group of words and check whether the searched word is found are not.

Here we use the structure definition TrieNode having the structure member character data, integer is leaf, count, array of size 26 to store the pointers of the structure TrieNode. The size of 26 because of the 26 alphabets in English.

**ISSUES**

The major issue we are facing here is the memory allocation . It consumes large amount memory to store the structure . And also the traversal inside the trie also a time consuming process. The worst – case runtime creating a trie is a combination of m, the length of the longest key in the trie and n, the total number of keys in the trie. Thus, the worst case runtime of creating a trie *is* ***O(m,n)***

**PROBLEM STATEMENT**

Data structure implementation should implements its interface correctly. Time Complexity – Running time or the execution time of operations of data structure must be as small as possible. Space Complexity – Memory usage of a data structure operation should be as little as possible.

The worst – case runtime creating a trie is a combination of m, the length of the longest key in the trie and n, the total number of keys in the trie. Thus, the worst case runtime of creating a trie is ***O(m,n)***

**MODULES**

In this mini project we used lots of functions to implements the creation and searching of the trie data structure.

**STRUCTURE DEFINITION**

Structure are a way to group several related variables into one place. A Structure may contain different data types.

struct node

{

struct node\* child[N];

int end;

char meaning[100];

};

typedef struct node node;

**LOAD FUNCTION**

Here we are going to create a trie by reading a words from the file. And it returns the root of the trie.

node\* load()

{

int i;

node\* root=NULL;

if ((root = (node\*)malloc (sizeof(node))) == NULL)

{

printf ("Out of memory. Dictionary could not be loaded.\n");

}

for (i = 0; i < N; i++)

root -> child[i] = NULL;

root->end=0;

int index;

node\* tmp = root;

FILE \*dict\_file=fopen("dict.txt","r+");

if (dict\_file == NULL)

printf("Error opening dictionary");

char word\_str[20];

char meaning[200];

while((fscanf(dict\_file,"%s",word\_str))!=EOF)

{ fgets(meaning,200,dict\_file);

int word\_len = strlen(word\_str);

for(i=0;i<word\_len;i++)

{

if (isalpha (word\_str[i]))

index = (int)tolower (word\_str[i]) - 'a';

if (index > 25 || index < 0)

continue;

if (tmp -> child[index] == NULL)

{

if ((tmp -> child[index] = (node\*)malloc ( sizeof (node))) == NULL)

{

printf ("Out of memory. Dictionary could not be loaded.\n");

}

tmp = tmp -> child[index];

int j;

for ( j = 0; j < 26; j++)

tmp -> child[j] = NULL;

tmp->end=0;

}

else

tmp = tmp -> child[index];

if (i == word\_len - 1){

tmp -> end =1;

strcpy(tmp->meaning,meaning);

}

} tmp=root;

} fclose(dict\_file);

return root;

}

**UNLOAD FUNCTION**

Here we going to free the created trie after the use of the trie is completed.

void unload\_rec (node\* dict\_rem)

{

node\* tmp = dict\_rem;

int i;

for (i = 0; i < N; i++)

if (tmp -> child[i] != NULL)

unload\_rec (tmp -> child[i]);

free (tmp);

tmp = NULL;

}

**SEARCH FUNCTION**

Here we are going to perform a search operation by traversal on the trie. And it returns 1 when it is present and 0 when it is not present.

char\* search(node \*root, char\* key)

{

node \*p = root;

int i;

for (i = 0;key[i]!='\0'; i++)

{

int index = (int)tolower(key[i]) - 'a';

if (p->child[index]==NULL)

return NULL;

p = p->child[index];

}

if((p->end==1))

return p->meaning;

else return NULL;

}

**VIEW FUNCTION**

Here we are going to display the available words in the text file by reading it from the file word by word.

void view()

{

FILE \*dict\_file=fopen("dict.txt","r");

if (dict\_file == NULL)

printf("Error opening dictionary");

char word\_str[20];

char meaning[200];

while((fscanf(dict\_file,"%s",word\_str))!=EOF)

{ fgets(meaning,200,dict\_file);

printf("%s\n",word\_str);

}

fclose(dict\_file);

}

**ADD FUNCTION**

Here we are going to add a new word into the trie and also on the text file.

void add(node\* root)

{

char word[20];

char meaning[200];

char tmpmeaning[20];

printf("Enter the word to add\n");

scanf("%s",word);

printf("Enter the meaning\n");

scanf("%s",meaning);

int word\_len = strlen(word);

int index,i;

node\* tmp=root;

for(i=0;i<word\_len;i++)

{

if (isalpha (word[i]))

index = (int)tolower (word[i]) - 'a';

if (index > 25 || index < 0)

continue;

if (tmp -> child[index] == NULL)

{

if ((tmp -> child[index] = (node\*)malloc (sizeof (node))) == NULL)

{

printf ("Out of memory. Dictionary could not be loaded.\n");

}

tmp = tmp -> child[index];

int j;

for (j = 0; j < 26; j++)

tmp -> child[j] = NULL;

tmp->end=0;

}

else

tmp = tmp -> child[index];

if (i == word\_len - 1){

tmp -> end =1;

strcpy(tmp->meaning,meaning);

}

}

FILE \*dict\_file=fopen("dict.txt","a");

fprintf(dict\_file,"\n%s ",word);fputs(meaning,dict\_file);

fclose(dict\_file);

}

**SOURCE CODE**

#include<stdio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

#define N 26

struct node{

struct node\* child[N];

int end;

char meaning[100];

};

typedef struct node node;

node\* load()

{

int i;

node\* root=NULL;

if ((root = (node\*)malloc (sizeof(node))) == NULL)

{

printf ("Out of memory. Dictionary could not be loaded.\n");

}

for (i = 0; i < N; i++)

root -> child[i] = NULL;

root->end=0;

int index;

node\* tmp = root;

FILE \*dict\_file=fopen("dict.txt","r+");

if (dict\_file == NULL)

printf("Error opening dictionary");

char word\_str[20];

char meaning[200];

while((fscanf(dict\_file,"%s",word\_str))!=EOF)

{ fgets(meaning,200,dict\_file);

int word\_len = strlen(word\_str);

for(i=0;i<word\_len;i++)

{

if (isalpha (word\_str[i]))

index = (int)tolower (word\_str[i]) - 'a';

if (index > 25 || index < 0)

continue;

if (tmp -> child[index] == NULL)

{

if ((tmp -> child[index] = (node\*)malloc ( sizeof (node))) == NULL)

{

printf ("Out of memory. Dictionary could not be loaded.\n");

}

tmp = tmp -> child[index];

int j;

for ( j = 0; j < 26; j++)

tmp -> child[j] = NULL;

tmp->end=0;

}

else

tmp = tmp -> child[index];

if (i == word\_len - 1){

tmp -> end =1;

strcpy(tmp->meaning,meaning);

}

} tmp=root;

} fclose(dict\_file);

return root;

}

void unload\_rec (node\* dict\_rem)

{

node\* tmp = dict\_rem;

int i;

for (i = 0; i < N; i++)

if (tmp -> child[i] != NULL)

unload\_rec (tmp -> child[i]);

free (tmp);

tmp = NULL;

}

char\* search(node \*root, char\* key)

{

node \*p = root;

int i;

for (i = 0;key[i]!='\0'; i++)

{

int index = (int)tolower(key[i]) - 'a';

if (p->child[index]==NULL)

return NULL;

p = p->child[index];

}

if((p->end==1))

return p->meaning;

else return NULL;

}

void view()

{

FILE \*dict\_file=fopen("dict.txt","r");

if (dict\_file == NULL)

printf("Error opening dictionary");

char word\_str[20];

char meaning[200];

while((fscanf(dict\_file,"%s",word\_str))!=EOF)

{ fgets(meaning,200,dict\_file);

printf("%s\n",word\_str);

}

fclose(dict\_file);

}

void add(node\* root)

{

char word[20];

char meaning[200];

char tmpmeaning[20];

printf("Enter the word to add\n");

scanf("%s",word);

printf("Enter the meaning\n");

scanf("%s",meaning);

int word\_len = strlen(word);

int index,i;

node\* tmp=root;

for(i=0;i<word\_len;i++)

{

if (isalpha (word[i]))

index = (int)tolower (word[i]) - 'a';

if (index > 25 || index < 0)

continue;

if (tmp -> child[index] == NULL)

{

if ((tmp -> child[index] = (node\*)malloc (sizeof (node))) == NULL)

{

printf ("Out of memory. Dictionary could not be loaded.\n");

}

tmp = tmp -> child[index];

int j;

for (j = 0; j < 26; j++)

tmp -> child[j] = NULL;

tmp->end=0;

}

else

tmp = tmp -> child[index];

if (i == word\_len - 1){

tmp -> end =1;

strcpy(tmp->meaning,meaning);

}

}

FILE \*dict\_file=fopen("dict.txt","a");

fprintf(dict\_file,"\n%s ",word);fputs(meaning,dict\_file);

fclose(dict\_file);

}

int main(){

node\* root=load();

int ch;

char word[50];

char meaning[200];

printf("\n=============TRIE DICTIONARY==============\n");

while(1)

{

printf("Enter your choice.\n1.View\n2.Add\n3.Search\n0.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 0: printf("BYE!!!!");

unload\_rec(root);

exit(0);

break;

case 1: view();

break;

case 2: add(root);

break;

case 3: printf("Enter the word:\n");

scanf("%s",word);

if(search(root,word))

{

strcpy(meaning,search(root,word));

printf("Meaning:%s\n",meaning);

}

else printf("Word not found\n");

break;

default : printf("\nenter a valid key\n");

}

}

return 0;

}

**RESULT**

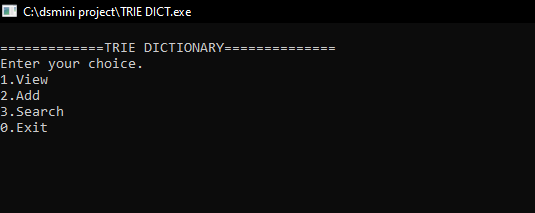


Fig. 5.1 Preview

**VIEW RESULT**

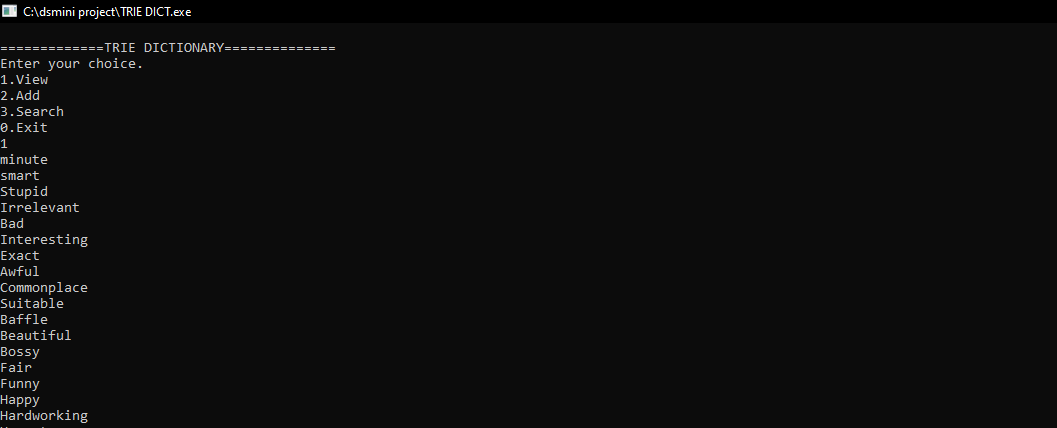


Fig. 5.2 View Result

**ADD RESULT**

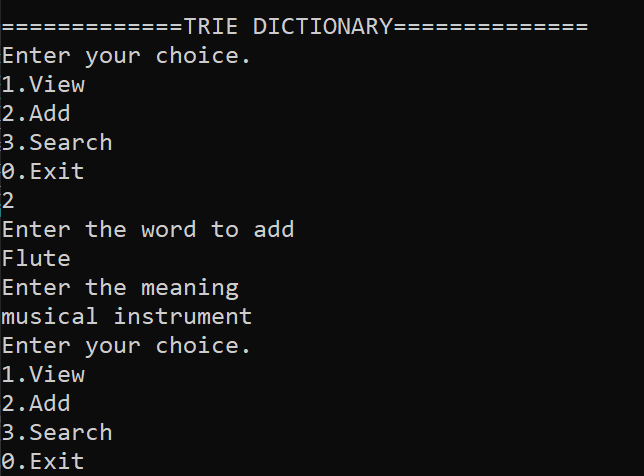


Fig. 5.3 Add Result

**SEARCH RESULT**



Fig. 5.4 Search Result

**EXIT**



Fig. 5.5 Exit Result

**CONCLUSION**

We gained lot of coding experience in designing forms and managing a database from this project. We would like to thank our HOD for giving this wonderful opportunity. Hope this project will help improve our career in future.