

PES UNIVERSITY

100 feet Ring Road, BSK 3rd Stage, Bengaluru-560085

UE17CS251

Design Analysis and Algorithms

Project Report on

TRIE

**PRESENTED BY:**

Aishwarya D PES1201802426

Shraddha B PES1201802434

**ABSTRACT:**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), a **trie**, also called **digital tree**, **radix tree** or **prefix tree**, is a kind of [search tree](https://en.wikipedia.org/wiki/Search_tree)—an ordered [tree](https://en.wikipedia.org/wiki/Tree_(data_structure)) [data structure](https://en.wikipedia.org/wiki/Data_structure) used to store a [dynamic set](https://en.wikipedia.org/wiki/Set_(abstract_data_type)) or [associative array](https://en.wikipedia.org/wiki/Associative_array) where the keys are usually [strings](https://en.wikipedia.org/wiki/String_(computer_science)). Unlike a [binary search tree](https://en.wikipedia.org/wiki/Binary_search_tree), no node in the tree stores the key associated with that node instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common  [prefix](https://en.wikipedia.org/wiki/Prefix" \o "Prefix) of the string associated with that node, and the root is associated with the [empty string](https://en.wikipedia.org/wiki/Empty_string).

Keys tend to be associated with leaves, though some inner nodes may correspond to keys of interest. Hence, keys are not necessarily associated with every node. For the space-optimized presentation of prefix tree, see [compact prefix tree](https://en.wikipedia.org/wiki/Compact_prefix_tree).

keys are listed in the nodes and values. Each complete English word has an arbitrary integer value associated with it. A trie can be seen as a tree-shaped [deterministic finite automaton](https://en.wikipedia.org/wiki/Deterministic_finite_automaton). Each [finite language](https://en.wikipedia.org/wiki/Finite_language) is generated by a trie automaton, and each trie can be compressed into a [deterministic acyclic finite state automaton](https://en.wikipedia.org/wiki/Deterministic_acyclic_finite_state_automaton).

Though tries are usually keyed by character stringsthey need not be. The same algorithms can be adapted to serve similar functions of ordered lists of any construct, e.g. permutations on a list of digits or shapes. In particular, a **bitwise trie** is keyed on the individual bits making up any fixed-length binary datum, such as an integer or memory address

**PROBLEM DEFINITION:**

**INSERT:-**

Inserting a key into Trie is a simple approach. Every character of the input key is inserted as an individual Trie node. Note that the children is an array of pointers (or references) to next level trie nodes. The key character acts as an index into the array children. If the input key is new or an extension of the existing key, we need to construct non-existing nodes of the key, and mark end of the word for the last node. If the input key is a prefix of the existing key in Trie, we simply mark the last node of the key as the end of a word. The key length determines Trie depth.

ALGORITHM:

1. begin for 1=0 to string length of key

1) point temp to root

2) compute index as key[i]=’0’

3) check if a[index] is NULL

🡪>create a node

4) traverse tmp to a[index]

End for loop

2. insert value

**SEARCH:-**

Searching for a key is similar to insert operation, however, we only compare the characters and move down. The search can terminate due to the end of a string or lack of key in the trie. In the former case, if the isEndofWord field of the last node is true, then the key exists in the trie. In the second case, the search terminates without examining all the characters of the key, since the key is not present in the trie.

Algorithm:

1. begin for 1=0 to string length of key

1)point temp to root

2) compute index as key[i]=’0’

3) check if a[index] is NULL

🡪>KEY FOUND (return)

4) traverse tmp to a[index]

End for loop

2)print the value

**DELETE:**

During delete operation we delete the key in bottom up manner using recursion. The following are possible conditions when deleting key from trie,

1. Key may not be there in trie. Delete operation should not modify trie.
2. Key present as unique key (no part of key contains another key (prefix), nor the key itself is prefix of another key in trie). Delete all the nodes.
3. Key is prefix key of another long key in trie. Unmark the leaf node.
4. Key present in trie, having atleast one other key as prefix key. Delete nodes from end of key until first leaf node of longest prefix key

ALGORITHM:

1. begin for 1=0 to string length of key

1)point temp to root

2) compute index as key[i]=’0’

3) travserse to the endofword

4) set wordflag to 0

End for loop

5)return 1

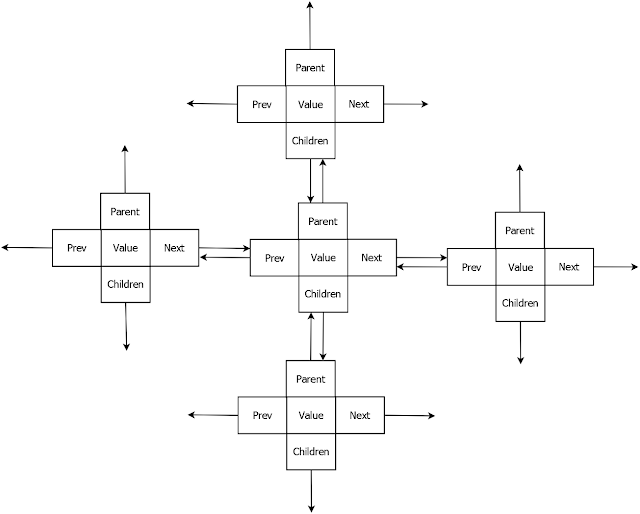
**INTRODUCTION:**

Trie is an efficient information reTrieval 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.

### Structure of Trie(Specific to this implementation):

The trie implemented here consists of nodes. Each node has these fields:

1. **Key** - Part of the string to be serached,inserted or deleted.
2. **Value** -  The value associated with a string (e.g In a dictionary it could be the meaning of the word which we are searching)
3. **Neighbour node address** - It consists of the address of the neighbouring node at the same level.
4. **Previous neighbour address** - It consists of the address of the previous node at the same level.
5. **Children node address** - It consists of the address of the child nodes of the current node.
6. **Parent node address** - It consists of the address of the parent node of the current node.

[](http://1.bp.blogspot.com/-MCLlgXnzSd8/UJu1MxpcMbI/AAAAAAAAAS4/yBIBV57q0ro/s1600/TrieNodes.png)

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. A simple structure to represent nodes of the English alphabet can be as following,

**DESIGN IMPLEMENTATION :**

#include<stdio.h>

#include<stdlib.h>

struct trie

{

struct trie \*alpha[26];

int wordflag;

};

typedef struct trie trie;

trie \* root;

trie \* createnode()

{

trie \* node=(trie\*) malloc(sizeof(trie));

int i=0;

for(;i<26;i++)

node->alpha[i]=NULL;

node->wordflag=0;

return node;

}

int search\_insert\_delete(trie \*node, char \*word, char mode)

{

if(!node)

{

if(mode=='i')

{

printf("\nCreating root node...");

node=createnode();

root=node;

}

else if(mode=='s' || mode=='d')

{

printf("\nRoot node does not exist.");

return 1;

}

}

while(\*word)

{

int index=(\*word|('A'^'a'))-'a';

if(!node->alpha[index])

{

if(mode=='i')

{

printf("\nCreating node for character '%c'...",\*word);

node->alpha[index]=createnode();

}

else if(mode=='s' || mode=='d')

{

printf("\nNo node found for character '%c'.",\*word);

return 1;

}

}

node=node->alpha[index];

word++;

}

if(!node->wordflag)

{

if(mode=='i')

{

printf("\nSetting wordflag...");

node->wordflag=1;

return 0;

}

else if(mode=='s' || mode=='d')

{

printf("\nWordflag not set."); return 1; }

}

else if(mode=='d')

{

printf("\nResetting wordflag...");

node->wordflag=0;

return 0;

}

else return 0;

}

main()

{

int ch;

do

{

char word[10];

int result;

printf("\n\n1. Enter into trie \n2. Search from trie \n3. Delete from trie \n4. Exit \n\nEnter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\nEnter the word to be inserted: ");

scanf("%s",word);

result=search\_insert\_delete(root,word,'i');

if(!result) { printf("\nInsertion successful!"); }

break;

case 2: printf("\nEnter the word to be searched: ");

scanf("%s",word);

result=search\_insert\_delete(root,word,'s');

if(result==0) printf("\nSearch successful!\nThe word is present in the trie.");

else printf("\nSearch unsuccessful!\nThe word is not present in the trie!");

break;

case 3 : printf("\nEnter the word to be deleted: ");

scanf("%s",word);

result=search\_insert\_delete(root,word,'d');

if(result==0) printf("\nDeletion successful!");

else printf("\nSearch unsuccessful!\nThe word is not present in the trie!");

break;

case 4 : break;

default: break;

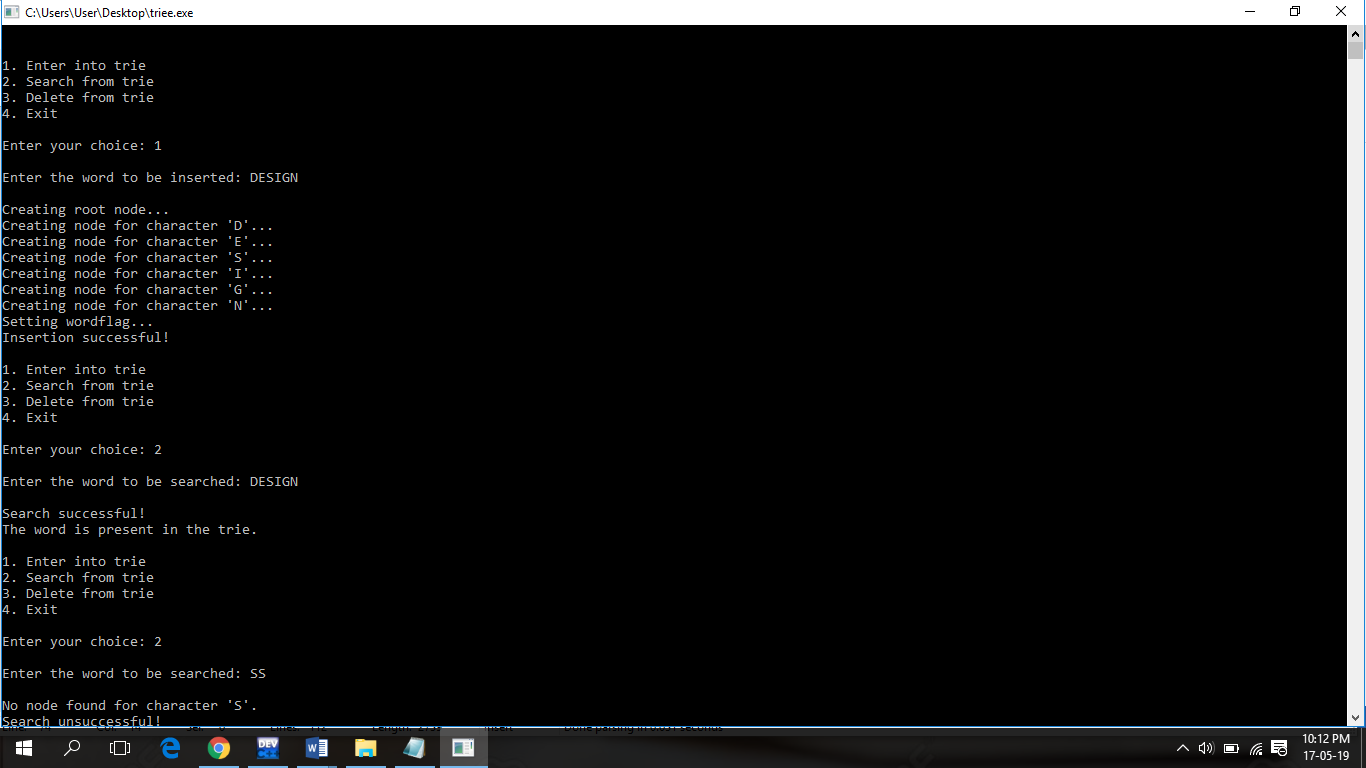
}

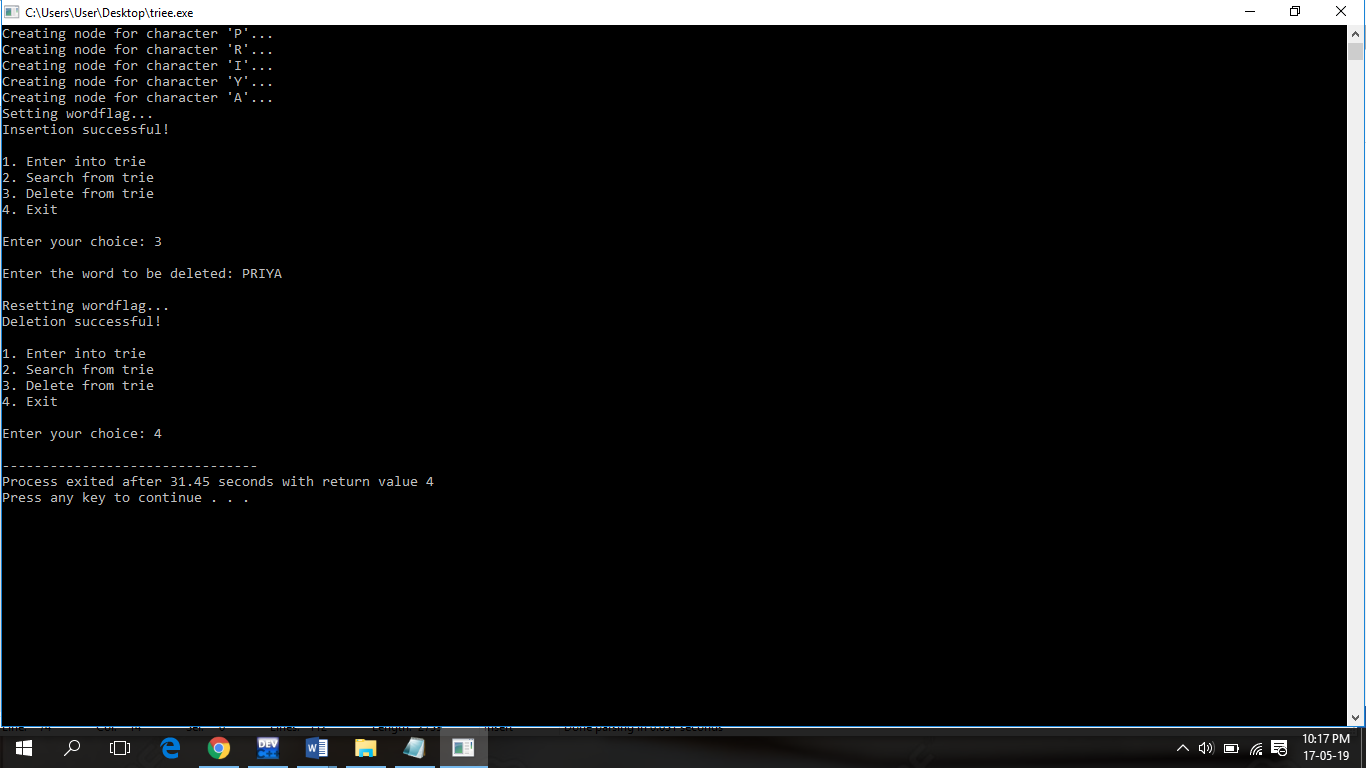
}

while(ch!=4);

}

**OUTPUT:**





**REFERENCE :**

(<https://www.sanfoundry.com/c-program-implement-schonhage-strassen-algorithm-multiplication-two-numbers/>)

<https://tonjanee.home.xs4all.nl/SSAdescription.pdf>

sanfoundry.com