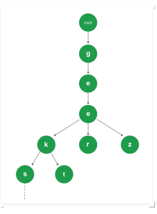
Trie | (Insert and Search)

Difficulty Level : **Medium** • Last Updated : 04 Sep, 2019

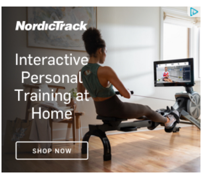
Trie is an efficient information re***Trie***val data structure. Using Trie, search complexities can be brought to optimal limit (key length). If we store keys in binary search tree, a well balanced BST will need time proportional to **M * log N**, where M is maximum string length and N is number of keys in tree. Using Trie, we can search the key in O(M) time. However the penalty is on Trie storage requirements (Please refer Applications of Trie for more details)



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,

```
// Trie node
struct TrieNode
{
     struct TrieNode *children[ALPHABET_SIZE];

     // isEndOfWord is true if the node
     // represents end of a word
     bool isEndOfWord;
};
```

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.



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.

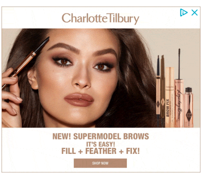
The following picture explains construction of trie using keys given in the example below,

```
                       root
                    /   \    \
                    t   a     b
                    |   |     |
                    h   n     y
                    |   |  \  |
                    e   s  y  e
                 /  |   |
                 i  r   w
                 |  |   |
                 r  e   e
                        |
                        r
```

In the picture, every character is of type *trie_node_t*. For example, the *root* is of type trie_node_t, and it's children *a*, *b* and *t* are filled, all other nodes of root will be NULL. Similarly, "a" at the next level is having only one child ("n"), all other children are NULL. The leaf nodes are in blue.

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Insert and search costs **O(key_length)**, however the memory requirements of Trie is **O(ALPHABET_SIZE * key_length * N)** where N is number of keys in Trie. There are efficient representation of trie nodes (e.g. compressed trie, ternary search tree, etc.) to minimize memory requirements of trie.

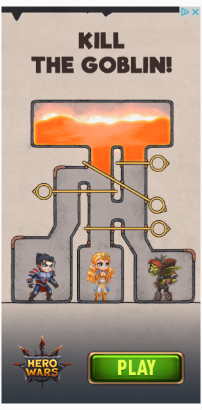


C++ C Java Python C#

```
// C++ implementation of search and insert
// operations on Trie
#include <bits/stdc++.h>
using namespace std;

const int ALPHABET_SIZE = 26;

// trie node
struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];

    // isEndOfWord is true if the node represents
    // end of a word
    bool isEndOfWord;
};

// Returns new trie node (initialized to NULLs)
struct TrieNode *getNode(void)
{
    struct TrieNode *pNode =  new TrieNode;

    pNode->isEndOfWord = false;

    for (int i = 0; i < ALPHABET_SIZE; i++)
        pNode->children[i] = NULL;

    return pNode;
}

// If not present, inserts key into trie
// If the key is prefix of trie node, just
// marks leaf node
void insert(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;

    for (int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();

        pCrawl = pCrawl->children[index];
    }

    // mark last node as leaf
    pCrawl->isEndOfWord = true;
}

// Returns true if key presents in trie, else
// false
bool search(struct TrieNode *root, string key)
{
    struct TrieNode *pCrawl = root;

    for (int i = 0; i < key.length(); i++)
    {
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            return false;

        pCrawl = pCrawl->children[index];
    }

    return (pCrawl != NULL && pCrawl->isEndOfWord);
}

// Driver
int main()
{
    // Input keys (use only 'a' through 'z'
    // and lower case)
    string keys[] = {"the", "a", "there",
                    "answer", "any", "by",
                     "bye", "their" };
    int n = sizeof(keys)/sizeof(keys[0]);

    struct TrieNode *root = getNode();

    // Construct trie
    for (int i = 0; i < n; i++)
        insert(root, keys[i]);

    // Search for different keys
    search(root, "the")? cout << "Yes\n" :
                         cout << "No\n";
    search(root, "these")? cout << "Yes\n" :
                           cout << "No\n";
    return 0;
}
```

Output :

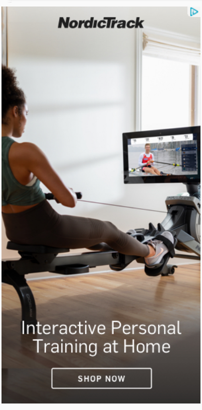
```
the --- Present in trie
these --- Not present in trie
their --- Present in trie
thaw --- Not present in trie
```



WHAT'S NEW

DSA Self Paced Course
View Details

Ad free experience with GeeksforGeeks Premium
View Details



MOST POPULAR IN ADVANCED DATA STRUCTURE

Decision Tree Introduction with example

Maximum Occurrence in a Given Range

Design a Chess Game

Disjoint Set Data Structures

Insert Operation in B-Tree

CALLING ALL CLASSICS
CLASSIC CROCS SLIDE
crocs
SHOP NOW

MORE RELATED ARTICLES IN ADVANCED DATA STRUCTURE

Generic Linked List in C

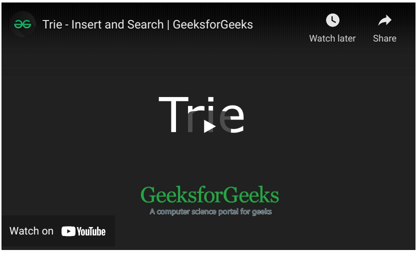
Binomial Heap

Red-Black Tree | Set 3 (Delete)

XOR Linked List - A Memory Efficient Doubly Linked List | Set 1

Delete Operation in B-Tree

Do you use Indian/Pakistani Recipe Masalas?
Yes
No


Trie - Insert and Search | GeeksforGeeks
Watch later
Share
Trie
GeeksforGeeks
A computer science portal for geeks
Watch on YouTube

NOTE : In video, **isEndOfWord** is referred as **isLeaf**.

Related Articles:

- Trie Delete
- Displaying content of Trie
- Applications of Trie
- Auto-complete feature using Trie
- Minimum Word Break
- Sorting array of strings (or words) using Trie
- Pattern Searching using a Trie of all Suffixes

Practice Problems :

1. Trie Search and Insert
2. Trie Delete
3. Unique rows in a binary matrix
4. Count of distinct substrings
5. Word Boggle

Recent Articles on Trie

This article is contributed by Venki. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course**.

♡ Like 0

< Previous

Advantages of Trie Data Structure

Next >

Trie | (Delete)

RECOMMENDED ARTICLES

Page : 1 2 3

01 Search in a trie Recursively
18, Jul 19

02 Design a data structure that supports insert, delete, search and getRandom in constant time
09, Apr 15

03 Treap | Set 2 (Implementation of Search, Insert and Delete)
22, Oct 15

04 2-3 Trees | (Search and Insert)
18, Oct 18

05 K Dimensional Tree | Set 1 (Search and Insert)
31, Oct 14

06 Overview of Data Structures | Set 3 (Graph, Trie, Segment Tree and Suffix Tree)
14, Feb 16

07 Trie Data Structure using smart pointer and OOP in C++
24, Jul 19

08 Auto-complete feature using Trie
07, Jun 17

● ● ●

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : princiraj1992, jarretchng02

Article Tags : Amazon, D-E-Shaw, FactSet, Trie, Advanced Data Structure

Practice Tags : Amazon, D-E-Shaw, FactSet, Trie

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company
About Us
Careers
Privacy Policy
Contact Us
Copyright Policy

Learn
Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

Practice
Courses
Company-wise
Topic-wise
How to begin?

Contribute
Write an Article
Write Interview Experience
Internships
Videos

We use cookies to ensure you have the best browsing experience on our website. By using our site, you acknowledge that you have read and understood our Cookie Policy & Privacy Policy

Got It !