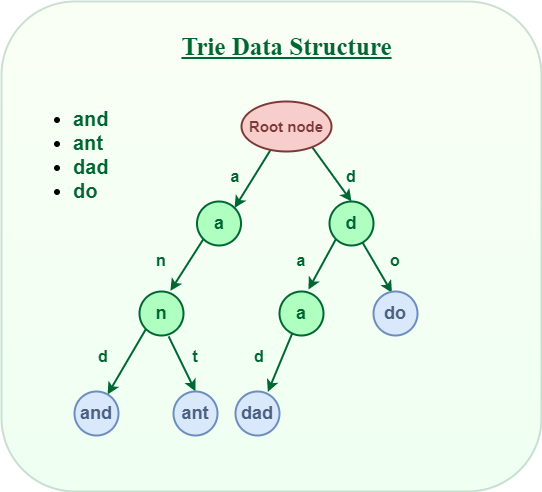
**Trie Data Structure**

*Trie is a type of k-ary search tree used for storing and searching a specific key from a set. Using Trie, search complexities can be brought to optimal limit (key length).*

If we store keys in a [binary search tree](https://www.geeksforgeeks.org/binary-search-tree-data-structure/), a well balanced BST will need time proportional to **M \* log N**, where **M** is the maximum string length and **N** is the number of keys in the tree. Using Trie, the key can be searched in **O(M)** time. However, the penalty is on Trie storage requirements (Please refer to [Applications of Trie](https://www.geeksforgeeks.org/advantages-trie-data-structure/) for more details).

Trie is also known as **digital tree** or **prefix tree**. Refer to [**this**](https://www.geeksforgeeks.org/introduction-to-trie-data-structure-and-algorithm-tutorials/) article for more detailed information.



*Trie data structure*

## **Structure of Trie node:**

Every node of Trie consists of multiple branches. Each branch represents a possible character of keys. Mark the last node of every key as the end of the word node. A Trie node field **isEndOfWord** is used to distinguish the node as the end of the word node.

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

| // Trie node  class TrieNode  {  TrieNode[] children = new TrieNode[ALPHABET\_SIZE];  // isEndOfWord is true if the node  // represents end of a word  boolean isEndOfWord;  } |
| --- |

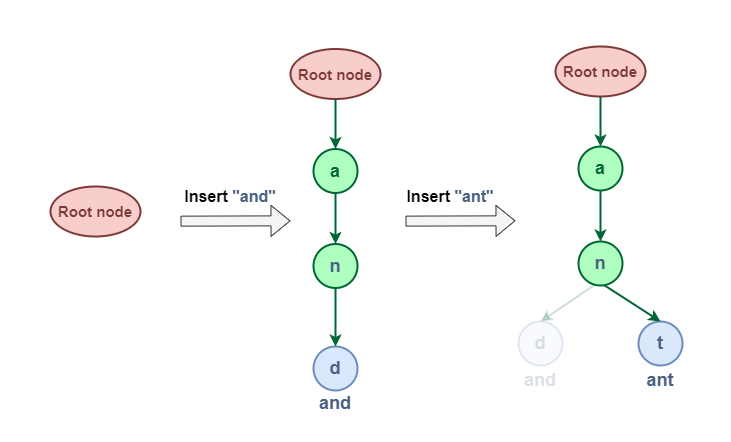
## **Insert Operation in** [Trie](https://www.geeksforgeeks.org/introduction-to-trie-data-structure-and-algorithm-tutorials/)**:**

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 to the array **children**.
* If the input key is new or an extension of the existing key, construct non-existing nodes of the key, and mark the end of the word for the last node.
* If the input key is a prefix of the existing key in Trie, Simply mark the last node of the key as the end of a word.

The key length determines Trie depth.

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

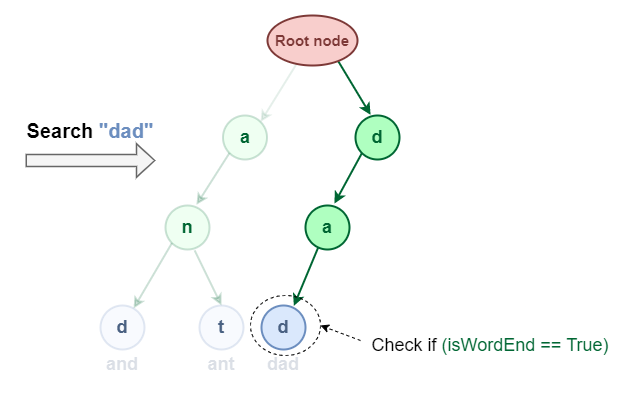


*Insertion operation*

## **Search Operation in** [Trie](https://www.geeksforgeeks.org/introduction-to-trie-data-structure-and-algorithm-tutorials/)**:**

Searching for a key is similar to the insert operation. However, It only **compares the characters and moves 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.



*Searching in Trie*

**Note:** Insert and search costs **O(key\_length)**, however, the memory requirements of Trie is **O(ALPHABET\_SIZE \* key\_length \* N)** where N is the number of keys in Trie. There are efficient representations of trie nodes (e.g. compressed trie, [ternary search tree](http://en.wikipedia.org/wiki/Ternary_search_tree), etc.) to minimize the memory requirements of the trie.

Recommended Problem

Trie | (Insert and Search)

[Trie](https://practice.geeksforgeeks.org/explore?page=1&category%5B%5D=Trie&sortBy=submissions)

[Design-Pattern](https://practice.geeksforgeeks.org/explore?page=1&category%5B%5D=Design-Pattern&sortBy=submissions)

+1 more

[Accolite](https://practice.geeksforgeeks.org/explore?page=1&company%5B%5D=Accolite&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5B%5D=Amazon&sortBy=submissions)

+3 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/trie-insert-and-search0651/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 28.8K

## **How to implement a Trie Data Structure?**

* Create a root node with the help of **TrieNode()** constructor.
* Store a collection of strings that have to be inserted in the trie in a vector of strings say, **arr**.
* Inserting all strings in Triewith the help of the **insert()** function,
* Search strings with the help of **search()** function.