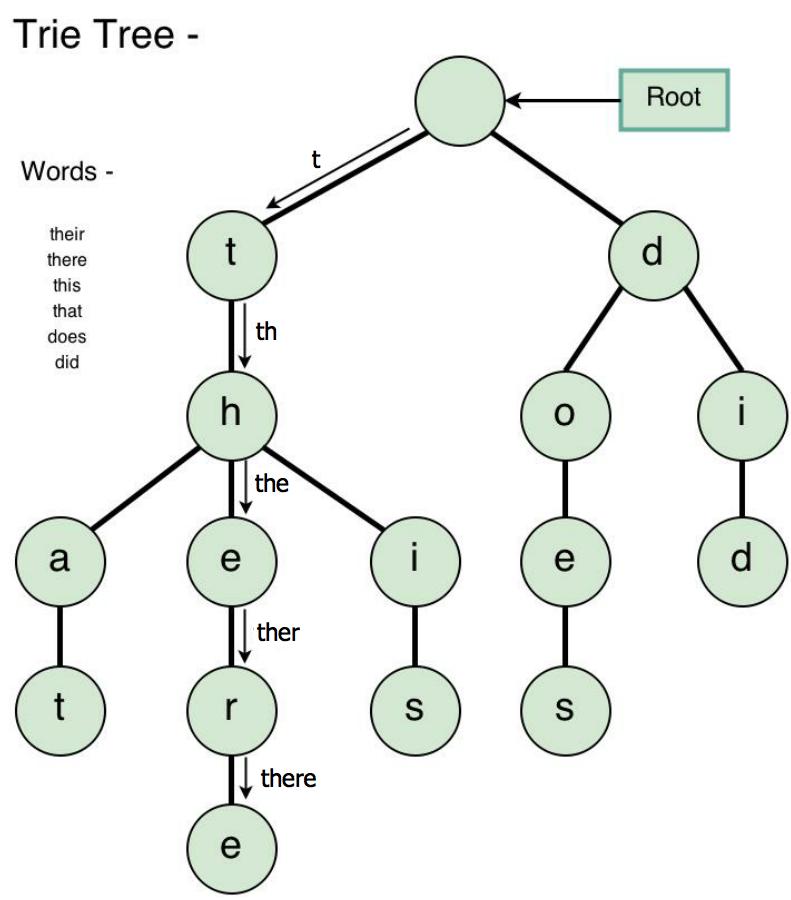
**Trie Tree**

# I. Introduction

A trie, also called digital tree or prefix tree, is a type of search tree, is an efficient information retrieval data structure.

It used for locating specific keys from within a set. These keys are most often strings.

Using Trie, we can search the key in **O(M)**, where M is maximum string length.



* Every node of Trie consists of mutiltiple 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 mark it.

We can represent Trie nodes as following:



# II. Implement Trie

1. **Inserting**

Inserting a key into Trie is very simple.

Every character of the input key is inserted as an individual Trie node. Note that the *children* is an array of pointer 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 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.



1. **Searching**

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.

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 cost for insert and search are **O(key\_length)**. The memory requirement of Trie is **O(Alphabet\_size \* key\_length \* N)** where N is number of keys in Trie.

*This is a program implement insert and search operations on Trie*



1. **Delete**

Algorithm how to delete a node from trie, we will delete the key in bottom up maner using recursion.

1. Key may not be there in Trie, not modify Trie

2. Key present as unique key (mean that no part of key contains another key (prefix), nor the key itself is prefix of another key in Trie). Delete all nodes.

3. Key is prefix of another long key in Trie. Unmark the leaf node.

4. Key present in Trie, having at least one other key as prefix key. Delete nodes from end of key until first leaf node of longest prefix key.





*This is a program implement delete operation on Trie* 

**The time complexity** of the deletion operation is **O(n)** where n is the key length.