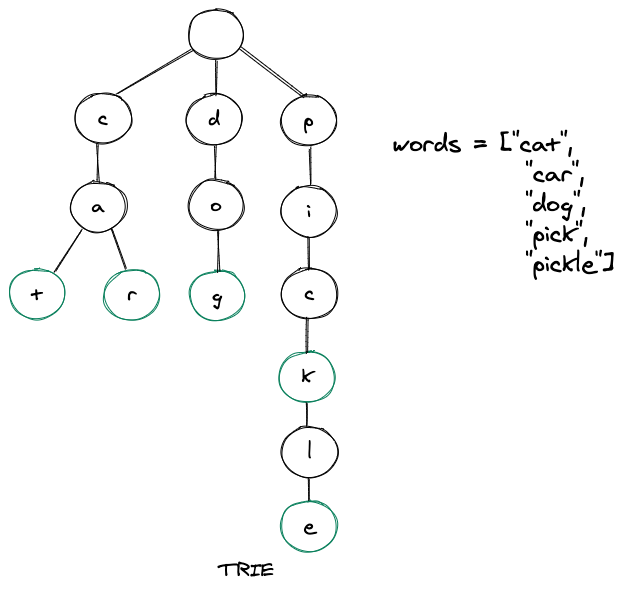
A Trie is an advanced data structure that is sometimes also known as prefix tree or digital tree. It is a tree that stores the data in an ordered and efficient way. We generally use trie's to store strings.

Each node of a trie consists of two things:

* A character
* A boolean value is used to implement whether this character represents the end of the word.

Tries in general are used to store English characters, hence each character can have 26 references. Nodes in a trie do not store entire keys, instead, they store a part of the key. When we traverse down from the root node to the leaf node, we can build the key from these small parts of the key.



When we talk about the fastest ways to retrieve values from a data structure, hash tables generally come to our mind. Though very efficient in nature still very less talked about as when compared to hash tables, trie's are much more efficient than hash tables and also they possess several advantages over the same.

## Basic operations of Trie

There are three operations in the Trie:

1. Insertion of a node
2. Searching a node
3. Deletion of a node

### Insert of a node in the Trie

When we insert a character(part of a key) into a trie, we start from the root node and then search for a reference, which corresponds to the first key character of the string whose character we are trying to insert in the trie.

### Searching a node in Trie

The second operation is to search for a node in a Trie. The searching operation is similar to the insertion operation. The search operation is used to search a key in the trie.

### Deletion of a node in the Trie

The Third operation is the deletion of a node in the Trie. Before we begin the implementation, it is important to understand some points: If the key is not found in the trie, the delete operation will stop and exit it. If the key is found in the trie, delete it from the trie.

## Applications of Trie

* The most common use of tries in real world is the autocomplete feature that we get on a search engine(now everywhere else too).
* Trie also has helped in checking the correct spellings of a word, as the path is similar for a slightly misspelled word.
* String matching is another case where tries to excel a lot.

## Key Points

* The time complexity of creating a trie is O(m\*n) where m = number of words in a trie and n = average length of each word.
* Inserting a node in a trie has a time complexity of O(n) where n = length of the word we are trying to insert.
* Inserting a node in a trie has a space complexity of O(n) where n = length of the word we are trying to insert.
* Time complexity for searching a key(word) in a trie is O(n) where n = length of the word we are searching.
* Space complexity for searching a key(word) in a trie is O(1).