Trie Data Structure

What is a Trie Data Structure?

A Trie, also known as a Prefix Tree or a Radix Tree, is a tree-based data structure that is used to store and retrieve **strings** efficiently. The word "trie" comes from the word "retrieval," which reflects its primary purpose of searching.

A trie is a collection of nodes, where each node represents a single character or a part of a string.

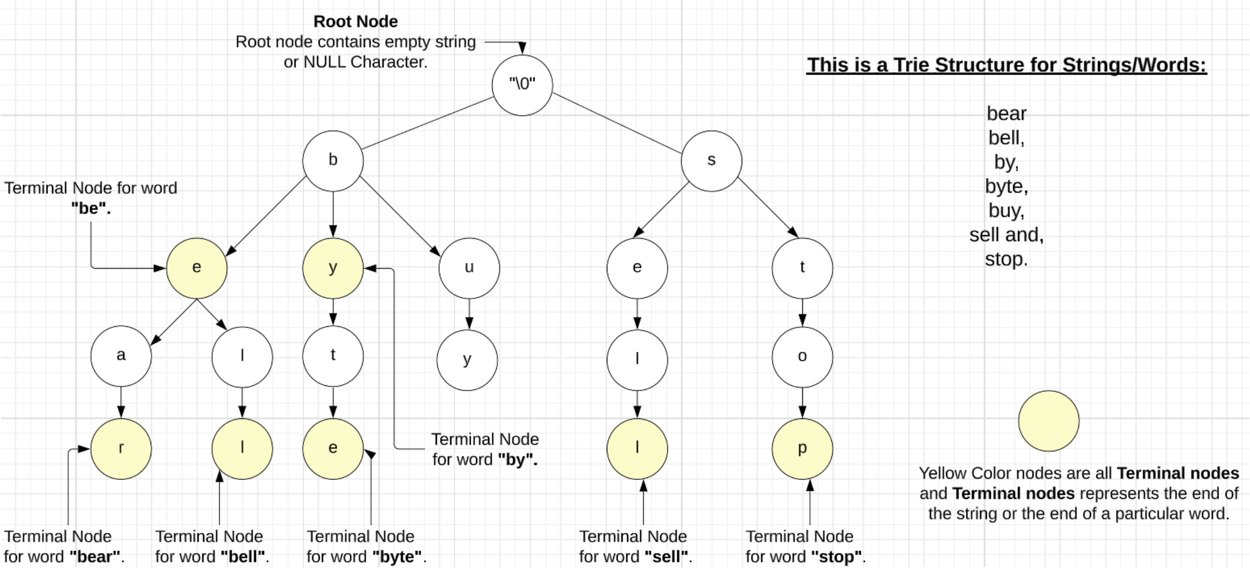
Properties of the Trie for a set of the string:

1. The root node of the trie always represents the null node.
2. Each child of nodes is sorted alphabetically.
3. Each node can have a maximum of **26** children (A to Z).
4. Each node (except the root) can store one letter of the alphabet.

Terminal Node in Trie:

In a trie data structure, a terminal node (also called an end node) is a node that represents the end of a string. Terminal nodes are marked in some way, such as by adding a flag to the node or by using a special character to represent the end of a string.

Example Trie:


Root Node
Root node contains empty string or NULL Character.
"\0"
This is a Trie Structure for Strings/Words:
bear
bell,
by,
byte,
buy,
sell and,
stop.
b
s
Terminal Node for word "be".
e
y
u
e
t
a
l
t
y
l
o
r
l
e
l
p
Terminal Node for word "by".
Yellow Color nodes are all Terminal nodes and Terminal nodes represents the end of the string or the end of a particular word.
Terminal Node for word "bear".
Terminal Node for word "bell".
Terminal Node for word "byte".
Terminal Node for word "sell".
Terminal Node for word "stop".

Trie data structures have several advantages and disadvantages, which are listed below:

Advantages:

1. **Fast string search:**
    a. Tries offer fast and efficient string search operations. Searching for a string in a trie takes **O(k) time**, where k is the length of the string.
    b. This is much faster than many other search algorithms, which take O(n) time in the worst case.
2. **Space-efficient:**
    a. Tries can be very space-efficient for storing large sets of strings. Because trie nodes can be shared among different strings, they require less storage space than storing each string individually.
3. **Prefix search:**
    a. Tries can be used to perform prefix search operations. This means that we can find all strings in a trie that start with a given prefix in O(k) time, where k is the length of the prefix.
4. **Ordered output:**

    a. Tries can produce ordered output when the strings are inserted in sorted order. This is because the trie stores the strings in a way that preserves their order.

Disadvantages:

1. **Space overhead:**
    a. While tries can be space-efficient for storing large sets of strings, they can also have a significant space overhead for small sets of strings. This is because each node in the trie has its own overhead, which can add up quickly for small strings.
2. **Limited to strings:**
    a. Tries are specialized data structures that are designed specifically for strings. They cannot be used for other types of data, such as numbers or objects.

Comparison of tries with hash table:

Tries and hash tables are two common data structures used for storing and searching large sets of data. Here is a comparison of some of the key features of these two data structures:

1. **Time Complexity:**
    a. Both tries and hash tables offer fast search times. Tries can search for a string in O(k) time (in both worst case as well as average case), where k is the length of the string, while hash tables can search for an element in O(1) time on average. However, in the worst case, hash table search can take O(n) time, where n is the number of elements in the table.
2. There is no hash function in trie.
3. There is no collisions in trie.
4. **Type of data:**
    a. Tries are specialized data structures designed for strings, while hash tables can be used for any type of data.
5. **Ordered output**:
    a. Tries can produce ordered output when the strings are inserted in sorted order. Hash tables do not preserve the order of elements.