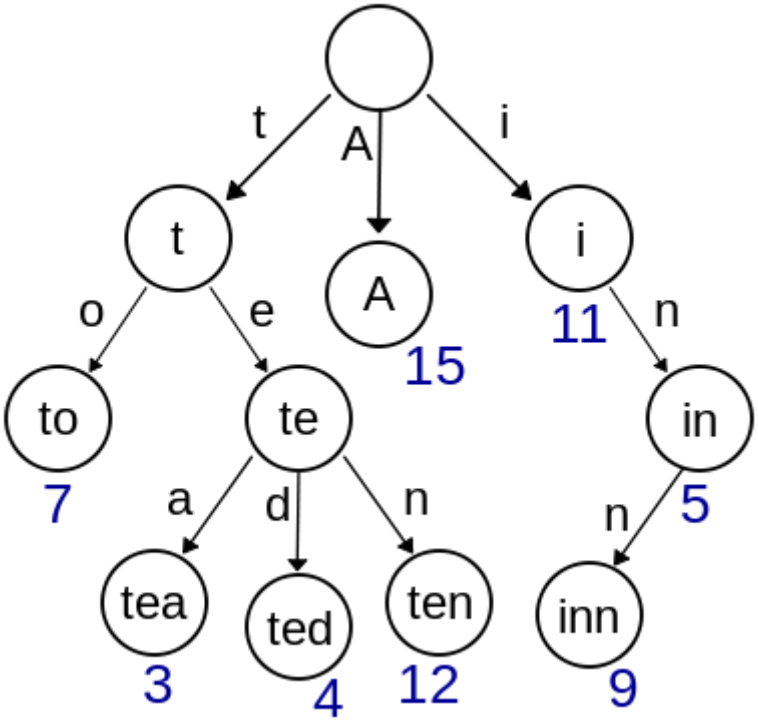
Trie

Trie is a data structure built from the characters of the string keys that allows us to use the characters of the search key to guide the search. 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 downside of Trie is that requires much more memory than trees and lists.



Basic Properties

As with search trees, tries are data structures composed of nodes that contain links that are either null or references to other nodes. Each node is pointed to by just one other node, which is called its parent (except for one node, the root, which has no nodes pointing to it), and each node has R links, where R is the alphabet size.
Each node also has a corresponding value, which may be null or the value associated with one of the string keys in the symbol table. Specifically, we store the value associated with each key in the node corresponding to its last character.

API Methods of Trie:

1. Search
2. Insert
3. Delete
4. `String longestPrefixOf(String s)` - the longest key that is a prefix of s
5. `Iterable<String> keysWithPrefix(String s)` - all the keys having s as a prefix
6. `Iterable<String> keysThatMatch(String s)` - all the keys that match s (where . matches any character)

Search in Trie

Finding the value associated with a given string key in a trie is a simple process, guided by the characters in the search key. We start at the root, then follow the link associated with the first character in the key, from that node we follow the link associated with the second character in the key and so on, until reaching the last character of the key or a null link.

At the last point one of the 3 cases are possible:

1. *Search hit* :
   The endOfWord boolean value is true, hence the complete word matched.
2. *Search miss* :
   a. The characters of the key word is exhausted but the endOfWord is false
   b. The endOfWord is true before the characters of the key word is exhausted or there is no child with current character in word.

Delete in Trie

If that node has a non-null link to a child, then no more work is required; if all the links are null, we need to remove the node from the data structure. If doing so leaves all the links null in its parent, we need to remove that node, and so forth.

Applications

- A trie can also be used to replace a hash table.
  Compared to default implementation of Hash table:
  - Looking up data in a trie is faster in the worst case, O(M) time (where M is the length of a search string), An imperfect hash table can have key collisions.The worst-case lookup speed in an imperfect hash table is O(N) time, but far more typically is O(1), with O(M) time spent evaluating the hash.
  - No collisions of different keys in a trie.

    - Buckets are needed in a trie, are necessary only if a single key is associated with more than one value.
    - There is no need to provide a hash function.
    - A trie can provide an alphabetical ordering of the entries by key.

    Drawback compared to hash tables:
    - Trie lookup can be slower than hash table lookup, especially if the data is directly accessed on a hard disk drive or some other secondary storage device where the random-access time is high.
    - Some tries can require more space than a hash table.
- Storing a predictive text or autocomplete dictionary
- Implementing approximate matching algorithms
    - Auto Complete
    - Spell Checkers
    - Longest Prefix Matching
    - Automatic Command completion
- Lexicographic sorting of a set of keys can be accomplished by building a trie from them, and traversing it in pre-order, printing only the leaves' values.

Algorithm Complexity

Considering *M* = key length, *N* = number of keys in trie, and *R* = size of character set ({a-z} = 26). Insert and search costs $O(M)$, however the memory requirements of Trie is $O(R * M * N)$

Types of Tries

1. Compressed Trie
2. Suffix Trie
3. Ternary search tries (TST)

    > To help us avoid the excessive space cost associated with R-way tries, we consider an alternative representation: the ternary search trie (TST). In a TST, each node has a character, *three* links, and a value. The three links correspond to keys whose current characters are less than, equal to, or greater than the node's character.