Documentation of

Ternary Search Trie

Designed and Implemented by Patrick Lewis

11/2/17: Began Documentation

Statement of Purpose:

The purpose of this project to create a data-structure which handles the efficient searching of a large number of strings with features including pattern recognition. The initial motivation behind the need for this library has been the desire to search a DBC file for information regarding a specific message or signal using partial information.

Intent of Design:

The functional intent of this project can be presented as a list of function names and their arguments:

void Trie<T>();

void Add(string name, T item);

T Remove(string name);

SortedList<string, T> Search(string pattern, bool ignoreCase = false);

The first function will be the constructor of the Trie and initialize the root of our Ternary Tree.

The Insert function will insert one item into the tree which will be immediately available for deletion or searching after the function returns.

The Remove function will remove the given search string and its items from the Trie. The string will not exist in the Trie immediately on the return of this function.

The Search function will take some input pattern as a string to match against the contents of this data-structure. The following characters have special meaning:

* ‘\*’ will represent a free character or string of characters of any length.
* ‘?’ will match any single character.

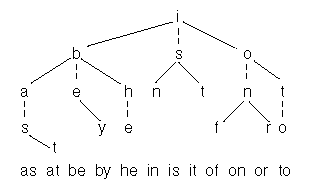
All other characters will be matched exactly unless the “ignoreCase” flag is set to True, in which there will be no distinguishing between upper case and lower.

Class Structure:

The structure of the project will all be under the Global namespace as it is a part of the “GlobalLibrary”. Additionally, the name of the main class will be “Trie<T>” and in addition to any tests will reside under the “TernarySearchTrie” namespace.

Details of Data-Structure:

A ternary search tree is a kind of trie which utilizes a binary search tree to improve space efficiency. Here is a very simple depiction of such a structure:



Any dashed line represents the next letter in a word. A solid line represents a separate option for a letter. In this way, the center-down line are the next characters and the two sides are the handling of collisions, which can be searched like a binary search tree.

The following time complexities apply: ([wikipedia](https://en.wikipedia.org/wiki/Ternary_search_tree))

|  |  |
| --- | --- |
| **Ternary Search Tree (TST)** | |
| **Type** | tree |
| **Time complexity in big O notation** | |
| |  |  |  |  | | --- | --- | --- | --- | | **Algorithm** |  | **Average** | **Worst Case** | | **Search** |  | O(log *n*) | O(*n*) | | **Insert** |  | O(log *n*) | O(*n*) | | **Delete** |  | O(log *n*) | O(*n*) | | |

Details of Implementation: