Name: Kerry Zheng

Student ID: 28794346

FIT2004: Prac 10, Customised Auto-complete

Firstly, I read in all the words from the dictionary file into a list, where each element stores the word, frequency, and definition. This has O(T) time and space complexity, where T is the number of characters in the dictionary file.

In constructing the trie that is array-based, each node is of size 30, where index 1 to index 26 would correspond to the letters in the English alphabet A-Z in order. Index 27 corresponds to the end of string character which I have set to be "$". Index 28 corresponds to the number of words that have the prefix corresponding to the letters from the root node to the current node, index 29 corresponds to the highest frequency of the word with the prefix, and index 30 corresponds to the index location of the word that has the highest frequency in the dictionary word list.

Firstly, we initialise a root node. Then for each word in the dictionary, for each character in the word, if a node containing that character exists we move to it, otherwise we create a node and set the element corresponding to the index number of that character in the current node to point to the node we just created. Each time we are at a node, we increment the word count in index 28 by one. If the node we are currently was just created in the previous iteration, we set the maximum frequency and the word index location to the word we are inserting. If the word's frequency we are inserting is larger than the current maximum frequency word's, then we change the values in the node accordingly.

Since if words have the same frequency, and the auto complete suggestion should be the alphabetically smaller word, if the frequency is the same then we compare the two words character corresponding to the letter of the node in the trie we are currently at. Since we know exactly where the two letters are and strings are array-based, and we are only comparing one character for each node, this takes O(1) time. Boundary cases for this method include accounting for when the two strings don't have the same size, characters not have the same letter before reaching the end of one of the strings, and when we are at the root node. We do not need to consider if the word we are inserting is established to be alphabetically smaller than the first word it encounters to have the same frequency, and whether if it is alphabetically smaller than the second word it encounters, different from the first word, where it has the same frequency, because the second word is always alphabetically larger than the first word by the properties of a trie.

To update the location of the word in the node to correspond to the alphabetically smaller word, we keep track of when the frequency first started to be the same. Then after the word has finished traversing the trie, if the new word is alphabetically smaller than a word that has the same frequency to it, we traverse the same path the word we are inserting down the trie and changing all the entries for the highest frequency word index to correspond to the index location of the new word, starting from the node that corresponds to the node where the frequency first started to be the same. The construction of the trie has O(T) time and space complexity, where T is the number of characters in the dictionary file.

To return results for a user's given prefix, we traverse the path of the prefix given down the trie. For each character in the prefix given, if a node corresponding to that character exists we move to it. If the node we are currently on doesn't have a child node that corresponds to the next character of the prefix, then the dictionary doesn't contain a word with the given prefix. Once we are at the node that corresponds to the last letter of the given prefix, we output the word count stored at index 28 in that node, and the word and its definition that has the highest frequency for that given prefix, which its index in the dictionary word list is stored in index 29 at the node corresponding to the last letter of the given prefix. This has O(M + N) time and space complexity, where M is the length of the prefix entered by the user and N is the total number of characters in the word with the highest frequency and its definition.