*Why Trie?*

Trie data structure has been used to store the various word used and thereafter, traversed to accomplish the autocomplete effect. The trie has been chosen for its fast retrieval and add time. In addition, the number of children nodes a trie can have also makes it an apt choice to be used.

*Class*

TrieNode

The following class contains the following class instances: a character (letter), a boolean that indicates end of a word and a HashMap that contains links to other TrieNodes with their character set as dictionary key. Each TrieNode could have up to 26 nodes (total number of alphabets). The HashMap is the main link between root/subroot to its children.

Trie

The following class contains a dummy root node as its class instance. Methods within Trie class has been discussed below.

*Methods*

insert()

The following method is used for the purpose of inserting elements to the Trie class. Starting from the root, the function looks for the first letter of the word as the key in the root node's HashMap.

If such key does not exist, a new TrieNode is created with the letter as key and a new HashMap as its value. Then set the current node to be the new TrieNode we just created and continue. If the key does exist, set current node to the child node we just found and continue.

When there is no more letters left in the word, set the boolean isLast to true to indicate that this is the end of the word.

Time and space complexity: O(n)

doesExist()

In order for the insert() function to work as intended, transverses the Trie to checks if the word exist.

search()

The search() function takes a combination of characters and first goes through the tree to see if the combinations exist, this takes O(n) with n being the length of the input phrase. If not, the search will be terminated and return empty results. If the exact match is found, the traverseTrie() function is then called to traverse the tree from the last TrieNode (letter) location and collect all possible words.

traverseTrie()

The following method checks every possible path using backtracking and saves the complete word to a list of String.

toChar()

As the name implies, converts integers to alphabets in the code.

*Miscellaneous*

Unless otherwise mentioned, the O(n) for the aforementioned methods are O(1). Also as you can tell, it passed all unit tests.

