Antony Holshouser

3/14/2020

Professor Peng. Du

CSS 143

**Autocomplete**

For the “Autocomplete” homework, I will be using an ADT called Trie. Trie is similar to a binary tree in which binary nodes in the binary tree point to 2 other binary nodes, hence the use of “binary,” while Trie nodes hold an array of Trie Nodes. This is useful for our particular problem because we can set the array size to the size of the alphabet in the Trie nodes (26).

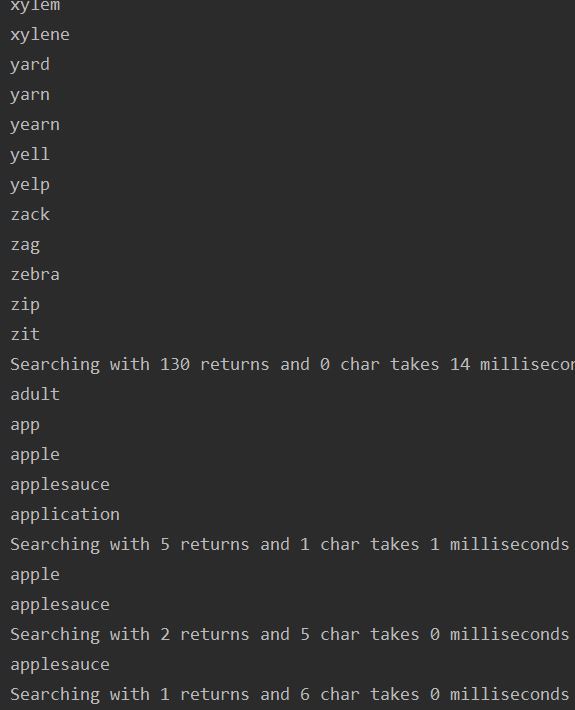
The other possible solution to this autocomplete homework is to compare every single string in an array that holds words. This might be a simple and easy approach to the problem, but the time complexity is much worse than using a Trie. If you were to compare every single word, the time complexity would be O(n \* k), where n is the number of entries into the dictionary, which is just the list of words, and k would be the number of characters searched for, as a “sea” would mean k = 3, and a “hawk” would mean k = 4.

In a Trie, we reduce the time complexity considerably, especially if we are dealing with a big list of words. But in a Trie, it’s not a “list of words” really, but it’s more like “what possible ways can the word expand from here on out.” An example would be if a word “seahawk” were to be inserted into the Trie, and we go to the TrieNode that holds ‘e’, the TrieNode array will only hold the letter ‘a’, with the others being null, as the only current way to expand the word, according to the Trie, is to insert an ‘a’. However, if “see” is entered into the Trie, then if we go to the “Seahawk”’s ‘e’, we will see that both ‘a’ and ‘e’ are now possible ways to extend the word. Within each word is a boolean and a value. The boolean is meant to store whether this TrieNode is marked as a word (“sea” and “seahawk,” the first ‘a’ and ‘k’ will be marked true, rest as false), and a character value is to store the character it resembles to the parent node.

This would reduce complexity. Instead of the complexity of O(n \* k), we will have a complexity O(c \* k) where c is the number of words that have to be returned. Accessing into an array is considered to be O(1), and we check all 26 of the possible routes. If we are searching the word “sea,” we first go to the ‘s’, ‘e’, and then ‘a’, and then use recursive backtracking from there on forth to get to all of the words that can be made with this start.

This makes the time complexity based on how many strings the search would return, instead of the whole size of the list the Trie is searching through.

After using a Trie with 130 total words, and testing it using 105 test cases,



It becomes apparent that the more characters that are searched, the execution time decreases.

This method has to be the fastest because it (1) reduces the number of comparisons. If a search of “se” goes through and a word “hawk” exists in the list, sure ‘h’ would be checked but it would even consider the rest of the word or any word that starts with ‘h’, or any letter that doesn’t start with ‘s’, and then does the same with ‘e’ and so forth. The comparing each string method requires checking every single word whether it starts with an ‘s’, and then ‘e’, and ignores the word if any one of them doesn’t match. In a Trie, it “landslide ignores” or ignores a bunch of possible retrieval words at once. (2) there doesn’t seem to be any obvious way to increase the speed in which it returns a list unless there is a way to program previously made searches so that if “se” was searched through before, it can remember what was searched when “se” was entered and just return that instead of doing the whole problem again.

The only improvements I could see being made are (1) indicated in “(2)” above. (2) making the Trie compatible with spaces, not just letters. (3) instead of using an array with a distinct size, use an ArrayList to reduce the capacity of the letters (might be especially useful for Chinese, for example, with thousands of letters). This also reduces limitations due to characters and spaces, because using this would allow for punctuations, symbols, and numbers, along with spaces.