Lab8 Assignment – Tries

Do these problems in order. While implementing/debugging, hard-code the program's input in your Python file. Once your code is working you can prompt the user for input.

Problem 1a: Prefix-free Standard Tries

Implement a standard trie data structure to store a collection of words. You will need to read in a collection of words and insert them into a (standard) trie. After this, your program should ask the user for an input word and report whether or not it exists in the collection (in the trie). You may assume that no word in the collection is a prefix of another word. For e.g. your collection can not have the words *to,today* since the first word is a prefix of the second.

Problem 1b: Standard Tries

Modify the code in Problem 1a) to work for the case when the words in the collection can be a prefix of one another. One way to do this is to add a special character not part of the alphabet (like $ or #) to the *end of each word* in the input. So words like *to,today* would then be stored as *to#, today#* in the trie.

Problem 2: Finding occurrences using Tries

Use the implementation in problem 1 to write a program that will read in a **text file** and pre-process the text by building a trie. Your program should then allow users to search for all occurrences of a word in the text. You will need to print the all the locations (the indices) where the searched word appeared in the text.

Problem 3*: Compressed Tries

Implement Problems 1 and 2 using Compressed Tries.

Problem 4*: Suffix Trees

The code in problem 1 and 2 is limited in that it can only search for *words* in the input text. It requires the existence of whitespaces within the text which separates words. It can not thus, search for occurrences of a pattern *agag* in some input sequence like *agatagagccagagtccagt*. This problem is solved by the Suffix Trees. Read about this Data Structure and write a program that allows such pattern matching.