IT251 Lab Assignment 7 – Tries

Note: For all the following problems, read in the input using a text file, and NOT by typing it in the console. The input file should be given as an argument while running your code. For e.g. for a file solution.cpp, compile it by 'g++ test.cpp' and run it by './a.out input.txt' , where 'input.txt' contains the input to the problem.

The problem we are trying to solve is the *Multiple Pattern Matching Problem* which is the following:

Input: A string **Text** and a collection **Patterns** containing shorter strings.
Ouput: All starting positions in **Text** where a string from **Patterns** appears as a substring.

To solve this problem we will first building a (standard) Trie from the set of strings in **Patterns** such that each string in **Patterns** has a unique root to leaf path in the Trie (and vice-versa). Then, we 'run' the Trie over the input **Text:** starting from each character in the input **Text,** and going rightwards, we see if we can find a root to leaf path in the Trie.

Problem 1: Building a Trie from a Collection of Patterns

In this part we will build a standard Trie for the set of strings in **Patterns**. The Trie we build will have the following properties:

1. The unique root node has a in-degree of 0.
2. Each edge of the Trie is labelled with a letter of the alphabet.
3. No two edges leading out of a given node have the same label.
4. The concatenation of the labels of any root to leaf path in the Trie gives a unique string in **Patterns**, and for every string in **Patterns** there is a unique such root to leaf path in the Trie.

Input: An integer n and a collection of strings **Patterns.** (each string is given on a separate line). Read in the input from an input file '**input.txt**'

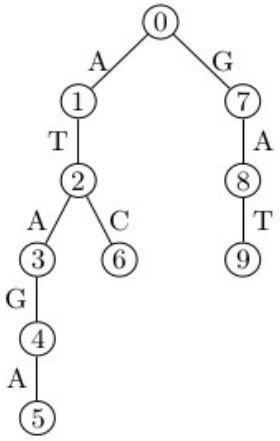
Output: The adjacency list corresponding to the Trie. If the Trie has k nodes, label the nodes from 0 to k-1. The root is labelled 0. For e.g. to display a node from node 0 to 1 with label 'A' print 0->1:A.

Constraints: $1 \leq |\textbf{Text}| \leq 10000$; $1 \leq n \leq 5000$; $1 \leq |p| \leq 100$ for all strings p in **Patterns;** all strings contain only symbols A, C, G, T; no string in **Patterns** is a prefix of another string in **Patterns.**

Sample Input/Output:
Input:
3
ATAGA
ATC
GAT

Output:
0->1:A
1->2:T
2->3:A
3->4:G
4->5:A
2->6:C
0->7:G
7->8:A
8->9:T

The output corresponds to following Trie:



Problem 2: Finding occurrences using Tries
In this part we will implement the pattern matching. Starting from each character in **Text**, we will read the characters to its right one by one, simultaneously tracing a path down from the root in the Trie. For each new character in **Text,** if we find an edge from the current node in the Trie which has an outgoing edge labelled with that character, we will go down along this edge and repeat the step for the next character. If we reach a leaf, we will declare that the pattern corresponding to this root to leaf path exists in the text. If at any internal node, no outgoing edge is found with the label of the new character, then we conclude that no string in **Patterns** matches with the **Text** at the current index. We then advance to the next position in **Text** and restart the matching process with the Trie.

Input: The first line of the input containts the string **Text**. The second line is an integer n and the next n lines is the collection of strings **Patterns.** (each string is given on a separate line).

Output: All starting positions (in increasing order) in **Text** where a string of **Patterns** appears as a substring.

Constraints: $1 \leq |\textbf{Text}| \leq 10000$; $1 \leq n \leq 5000$; $1 \leq |p| \leq 100$ for all strings p in **Patterns;** all strings contain only symbols A, C, G, T; no string in **Patterns** is a prefix of another string in **Patterns.**

Sample Input/Output:
Input:
AATCGGGTTCAATCGGGGT
2
ATCG
GGGT

Output:
1 4 11 15

The pattern ATCG appears at positions 1 and 11, the pattern GGGT appears at positions 4 and 15 in **Text**.

Problem 3: Extending Problem 2.

Implement the pattern matching of Problem 2 now without the prefix-free constraints of the strings in **Patterns.** The problem description is the same as in the Problem 2, except that the strings in **Patterns** can now be substrings of each other.

Sample Input/Output:

Input:
ACATA
3
AT
A
AG

Output:
0 2 4