Assignment Four

Plagiarism is strictly prohibited!

In this individual assignment, you will implement the **compact** representation of the compressed suffix trie ADT, design and implement an algorithm for finding k longest substrings of two DNA sequences.

A template of the compact compressed suffix trie class is shown as follows:

```
public class CompactCompressedSuffixTrie
 {
   /** You need to define your data structures for the compressed trie */

   /** Constructor */

   public CompactCompressedSuffixTrie( String f ) // Create a compact compressed suffix trie from file f
    { }

   /** Method for finding the first occurrence of a pattern s in the DNA sequence */

    public int findString( String s )
    { }

   /** Method for finding k longest common substrings of two DNA sequences stored
       in the text files f1 and f2 */

   public static void kLongestSubstrings(String f1, String f2, String f3, int k)
   { }
}
```

The data structures for the compressed suffix trie are not given in the above template. You need to define them yourself. You may introduce any helper classes and methods to facilitate the implementation of these two methods.

The constructor creates a compact representation of the compressed suffix trie from an input text file *f* that stores a DNA sequence. All the characters of the DNA sequence are A, C, G and T. **There is no specific time complexity requirement for the constructor. However, you need to make it as time-efficient as possible.**

Bonus Marks: there are **two bonus marks** for a correct linear time constructor for the compact compressed trie.

The findString(String *s*) method has only one parameter: a DNA pattern *s*. If *s* appears in the DNA sequence, findString(*s*) will return the starting index of the first occurrence of *s* in the DNA sequence. Otherwise, it will return –1. For example, if the DNA sequence is AAACAACTTCGTAAGTATA, then findString("CAACT") will return 3 and findString("GAAG") will return –1. Note that the index of the first character of the DNA sequence is 0.

Time complexity requirement for the method findString(String s): If your findString(String s) method is slower than O(|s|) (|s| is the length of s), you will get **0 mark** for it.

The method kLongestSubstrings(String *f1*, String *f2, String f3, int k*) computes the k longest common substrings **such that any two of them do not overlap**.

A common substring of two DNA sequences is a substring that appears in both DNA sequences. For example, GTTAA is a common substring of AA**GTTAA**AAGT and GT**GTTAA**AATTGA. The longest common substring is a substring with the maximum length. For example, GTTAAAA is the longest common substring of AA**GTTAAAA**GT and GT**GTTAAAA**TTGA. Note that the longest common substring is not always unique.

Specifically, the method kLongestSubstrings(String *f1*, String *f2, String f3, int k*) performs the following task:

- Compute the k longest common substrings of the two DNA sequences stored in the text files named f1 and f2, respectively, and write them to the file f3. The k longest common substrings must be stored in non-increasing order of their lengths and each common substring starts in a new line with a heading "i:", where i is the sequence number of the longest common substring.

For example, assume we have two DNA sequences stored in the files f1 and f2, respectively as follows:

f1=ACGTCCACGGTTTGGATTGAATTT

f2=ACGTAAACGGTTTTTATTGAATTT

After the call kLongestSubstrings(*f1, f2, f3, 3*), f3 will contain the 3 longest common substrings shown as follows:

1: ATTGAATTT

2: ACGGTTT

3: ACGT

Time complexity requirement for the method kLongestSubstrings(String *f1*, String *f2, String f3, int k*): **The running time of your method kLongestSubstrings(String f1, String f2, String f3, int k) must be at most O(kmn), where m and n are the sizes of *f1* and *f2*, respectively. Any method with a higher time complexity will be given 0 mark.**

You need to give the running time analyses of all the methods in terms of the Big O notation. Include your running time analyses in the source file of the CompactCompressedSuffixTrie class and comment out them.

Input exceptions: You can use any reasonable methods to handle incorrect input such as non-DNA symbols in a DNA sequence.

How to submit?

Follow this link: https://cgi.cse.unsw.edu.au/~give/Student/give.php. Do the following:

1. Use your z-pass to log in.
2. Select current session, COMP9024 and assn4.
3. Submit CompactCompressedSuffixTrie.java

Marking

This assignment is worth 7 marks, excluding the bonus marks. The breakdown is as follows.

1. Constructor: **3 marks or 5 marks for a correct linear time constructor**.
2. The method findString(String *s*): **1 mark**.
3. The method kLongestSubstrings(String *f1*, String *f2*, *String f3*, *int k*): **3 marks**.

Marking will be based on the correctness, time efficiency and readability of your code.

Deadline: 11:59:59pm, 4 June

No late submission will be accepted!