Assignment Four

In this individual assignment, you will implement the **compact** representation of the compressed suffix trie ADT for DNA analyses.

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

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

   /** Constructor */

   public CompressedSuffixTrie( String  f )  // Create a 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 computing the degree of similarity of two DNA sequences stored
       in the text files f1 and f2 */

   public static float similarityAnalyser(String f1, String f2, String f3)
   { }
 }
```

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 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. The findString(*s*) method has only one parameter: a 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.

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

The method similarityAnalyser(String f1, String f2, String f3) performs the following tasks:

1. Computing the longest common subsequence of the two DNA sequences stored in the text files f1 and f2, respectively, and writing it to the file f3.
2. Returning the degree of similarity of two DNA sequences stored in the text files f1 and f2. The degree of similarity of two DNA sequences S1 and S2 is equal to |lcs(S1,S2)|/max{|S1|,|S2|}, where |lcs(S1,S2)|, |S1| and |S2| are the lengths of a longest common subsequence of S1 and S2, S1 and S2, respectively. For simplicity, you may assume that each file contains at most 1000 DNA characters. When your program reads a DNA sequence from a text file, it needs to ignore all non-DNA characters such as the newline character. Notice that this method does not need to use any compressed suffix trie. **The running time of your method similarityAnalyser(*f1, f2, f3*) is required to be at most O(mn) , 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 CompressedSuffixTrie class as comments.

How to submit your code?

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 CompressedSuffixTrie.java

Marking

This assignment is worth 7 marks. The breakdown is as follows.

1. Constructor (**2.5 marks**).
2. The method findString(String *s*) (**2.5 marks**).
3. The method similarityAnalyser(String *f1*, String *f2, String f3*) (**2 marks**).

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

Deadline: 11:59:59pm, 5 June

No late submission will be accepted!