**CS1401 – Programming Assignment 5**

**Due: Monday October 12, 2015 by 8:00 a.m. for those of you taking the MW labs**

**Tuesday October 13, 2015 by 8:00 a.m. for those of you taking the TR labs**

**To be submitted via Piazza in the folder “lab5”**

**Welcome to Lab 5!**

|  |
| --- |
| This week, you will be working on an assignment that will allow you to review concepts covered in class or in labs in previous weeks. We have prepared a skeleton code for you to tinker with. This lab will give you the opportunity to reinforce your knowledge of:   * Input / output * Algorithms * Loops * Strings   We hope you have fun!  Ok, let’s get started! Here is the activity you will be working on. |

**You should expect to work about 3 to 4 extra hours outside the lab session to complete this assignment.**

**It means that you need to make sure that Java works on your own computer, or that you go to open labs to work some more.**

**Extra time on labs includes completing the activities and taking the time to make sure that your submission is picture perfect!**

**Note: you have 2 weeks to complete this lab!**

**Activity 1. Matching Strings.**   
*(Inspired by Problem Set 3 from MIT OpenCourseWare, made available on Google’s EngageCSEdu)*

String matching is very valuable in settings such as biology. A common problem in modern biology is to understand the structure of DNA molecules, and the role of specific structures in determining the function of the molecule. A DNA sequence is commonly represented as a sequence of one of four nucleotides – adenine (A), cytosine (C), guanine (G), or thymine (T) –and hence a DNA molecule or strand is represented by a string composed of elements from an alphabet of only four symbols, for example, the string AAACAACTTCGTAAGTATA represents a particular strand of DNA.

One way to understand the function of a particular strand of DNA (or even a sub-strand of DNA) is to match that strand against a library of known DNA sequences – that is, sequences whose function and structure is known – with the idea that similar structure tends to imply similar function. Simple organisms such as bacteria may have millions of nucleotides in their DNA sequence, and the human chromosome is believed to have on the order of 246 million bases, so any matching scheme must be very efficient in order to be useful.

In this activity, we won’t ask you to build a practically useful tool, but hope to give you a sense of some of the issues involved, by exploring some simple matching schemes. We are asking you to program the following functionalities:

1. Write a method, called **lastIndexSubStringMatch**, which takes three arguments (parameters): a key string, a target string, and an integer index. This method checks if the key string appears in the target string after (including) character at location index.

It will return the last index of the first occurrence if the key string does appear in the target string after location index.

It will return -1 otherwise.

Example 1:

lastIndexSubStringMatch("atgc","atgacatgcacaagtatgcat",3) should return: 8.

because “atgc” appears in the target string starting at location 5 and ends at location 8.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | T | G | A | C | A | T | G | C | A | C | A | A | G | T | A | T | G | C | A | T |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | A | T | G | C |  |  |  |  |  |  |  |  |  |  |  |  |

We see that we matched “atgc” to a substring of the target string, starting at an index larger than 3 (all smaller indices were grayed), and the last index of this substring is 8: this is what the method needs to return.

*Note: “atgc” appears later in the target string (between indices 15 and 18) but your method “lastIndexSubStringMatch” only looks at the first occurrence of the key string.*

Example 2:

lastIndexSubStringMatch("atga","atgacatgcacaagtatgcat",3) should return: -1.

because “atga” appears in the target string only starting at location 0, which is before the index parameter (=3).

1. Write a method, called **countSubStringMatch**, which takes two arguments: a key string and a target string. This method iteratively counts the number of instances of the key string in the target string.

Example 1:

countSubStringMatch("atgc","atgacatgcacaagtatgcat") should return: 2.

because “atgc” appears in the target string starting at location 5 and at location 15.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | T | G | A | C | A | T | G | C | A | C | A | A | G | T | A | T | G | C | A | T |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | A | T | G | C |  |  |  |  |  |  | A | T | G | C |  |  |

1. Write a method, called **ratioSubStringMatch**, which takes two arguments: a key string and a target string. This method will return the percentage of likeliness of the target string and the key string. We defined this percentage of likeliness as follows:

ratioSubStringMatch(str1,str2) = {(length of str1)

\*(number of occurrences of str1 in str2)

/ (length of str2) } \* 100

ratioSubStringMatch("atgc","atgacatgcacaagtatgcat") should return: 38% (=(4\*2/21)\*100.

1. Write a method called **SubStringMatch**, which takes three arguments: the name of a file, and two integers, called first and second, representing line numbers in the file. This method returns the result of ratioSubStringMatch applied to the string at line “first” and the string at line “second”.

Note: In the target file, you expect to find one string per line and we will also assume that we will not pass as parameters invalid line numbers (e.g., if the file contains only 5 lines, we will not pass a parameter with value 6).

1. **Challenge activity (25 extra points)**

Write a method MaxRatioSubString, which takes one argument: the name of a file, containing one string per line. It finds the two strings in the file with the maximum percentage of likeliness (value of ratioSubStringMatch), displays these two strings along with their percentage of likeliness (value of ratioSubStringMatch), and also writes this information at the bottom of the given file.

**What you have to turn in:**

* A docx file in which you describe the pseudocode of each of the above methods.
* A **single** java file that contains all methods.

**Important notes:**

* Indent your code properly following guidelines available at: <http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-136091.html>. 15 points will be reserved for correct indentation. Failing to indent properly puts you at risk of losing 15 points.
* Spend time working on your pseudocode as the amount of points you get for the pseudocode is bigger than the amount of points you get for your code (usually, close to a 60/40 ratio).
* Do not write your methods inside the main method. Each method has to be written where instructed in the code provided to you. Failing to do so puts you at risk of losing 20 points on your lab grade.
* Do not submit more than the files that are requested from you: one docx file and one java file.

**That’s it! Looking forward to seeing you in lab!**