CSC110 Assignment 5: Arrays

Objectives

Upon completion of this assignment, you need to be able to:

- Iterate through an array from its beginning index to the end.
- Determine cumulative information as you progress through an array.
- Determine equality of string objects.
- Access the individual characters of a string object in Java.
- Continue building good programming skills.

Introduction

An important task in bioinformatics is the identification of DNA and RNA sequences. In this assignment, we will be looking at *nucleic acid sequences*. These sequences contain up to four different *bases* denoted by letters: A for adenine, C for cytosine, G for guanine, and T for thymine. Sequence strings are compared in order to determine whether nucleic acid sequences match each other, or are related through *mutations*. Real sequence data, as used by biochemists and in bioinformatics research, consists of very long strings of bases. Determining relatedness can require the use of very complex algorithms, beyond the scope of this assignment.

The sequences in this assignment will all contain between two and four of the possible bases in $\{A, C, G, T\}$. Your task to to search through a collection of sequence data and count how many times a specific sequence occurs. For example, if the collection contains the following sequences: $\{ACTG, \underline{GATC}, ACT, GTC, AC, \underline{GATC}, GA\}$ and we search for the specific sequence $GATC$, we would report that it was found 2 times.

One of the challenges in this assignment will be dealing with *mutated* sequences. A mutation can occur due to insertions of additional bases within a sequence. For the purpose of this assignment, a mutated sequence contains at least two of the same bases occurring in a row; for example, in the sequence $GAAATC$, the A has mutated, and in the

sequence $CCGGAT$, both the C and G have mutated. Another task in this assignment is to detect how many of the sequences in the collection are mutated. The final task will be to search through the collection of sequence data for a specific sequence, but you must treat original and mutated sequences the same. For example, if the collection contains $\{\underline{TGC}, AC, \underline{TTGC}, TACG, \underline{TGGCC}, AGTC\}$ and we search for the specific sequence TGC, we would report that it was found 3 times, because $TTGC$ and $TGGCC$ are mutated forms of TGC.

Quick Start

(1) Download this pdf file and store it on your computer.
(2) Create a directory / folder on your computer specific to this assignment. for example, `CSC110/assn5` is a good name.
(3) Save a file called `DNASequencing.java` in your directory. Fill the source code with the class name and the methods outlined in the following specification document. Make sure the method headers are exact copies.
(4) Start with the easiest methods first. See the following detailed instructions below for some tips.
(5) Complete each method and test it thoroughly before moving onto the next one. Compile and run the program frequently; it is much easier to debug an error shortly after a successful and error-free run.

Detailed Instructions

We offer the following organization of the parts of the programming task. For more details about what each of the method *does*, check out the specification document. This document has several links to the Java `String` class. Look through the method summary for methods that let you manipulate string objects. You already know about the `length` method. You may find the `toCharArray` helpful as well.

In the following subsections, we provide a detailed description of how to approach the implementation of each of the methods named in the `DNASequencing` class:

Start with `printArray`

This is a good warmup exercise. Use a `for` loop to visit each element in the array input parameter. Remember that array indices start at 0, as do the indices of each character in a `String`.

`findLongest` and `findFrequency`

After completing and testing `printArray`, work on the `findLongest` or the `findFrequency` methods; they are similar. Within both methods, use a loop to visit each element in the array. In the `findLongest` method, you must keep track of which `String` object in the array contains the most characters, whereas in the `findFrequency` method, you must keep track of how many times a specific `String` object is found in the array. Make sure you finish and test both of these before moving to the next step.

The methods that involve mutations

These methods deal specifically with the DNA sequences, so you can assume that the input parameters are legitimate DNA sequences, even though they are of type `String`.

`hasMutation|`

This method is probably the easier one to implement, so you can start with that. *For those of you who like an extra challenge*, consider writing a more general method that deals with *any* string and returns `true` if there are consecutive characters that are identical. Experienced programmers often look for ways to make code that is re-usable, easy to maintain, and modular. For example, suppose that you wish to create a more general method (for possible future use) that deals with any string. And suppose it is called `hasCharacterRun`.

After fully testing it, it is stored in your personal library (or toolkit) of code. Now, the `hasMutation` method is very simple and requires no testing:

```
/*
 * Determines if a DNA sequence is mutated, i.e. has repeated characters.
 * input: sequence A String sequence of two to four
 * of the following characters: {A,C,G,T}
 * returns: true if the DNA sequence is mutated, false if it is not.
 */
 public static boolean hasMutation(String sequence) {
   return hasCharacterRun(sequence);
}
```

`countTotalMutations`

Implement this method by building on the technique for `findFrequency` and adding calls to `hasMutation`.

`findFreqWithMutations`

This method examines the contents of the sequence array, looking for a specific sequence and its mutations. By the time you have done all the other methods, this should be reasonably simple. You may want to create a *helper* method, but it is not required.

Building and testing

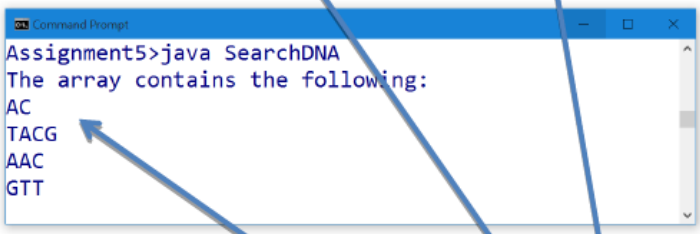
As you work through a solution, we strongly recommend that you save, compile and test the code after every line or two. This can be something as easy as printing out the value of a variable, or calling a method to print out the value returned. It is important to do this to confirm a component of the code works correctly, so you can be confident using that component throughout the code in later steps.

For each of the methods listed in the specifications, you must provide an internal test call from the main method If the method does not behave as expected, then debug and adjust the method. To receive full marks for testing, each method must be tested, even if it does not work.

Examples

We provide a couple of internal test cases you can use to test the correctness of the methods. The following diagrams demonstrate some sample testing for the `printArray` and `findLongest` methods.

```java
public static void main(String[] args) {
    String[] testArray = {"AC", "TACG", "AAC", "GTT"};
    printArray(testArray);
}
```

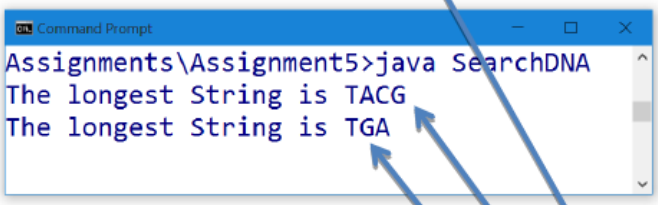


What to look for:

- Are all of the words printed?
- Does the method accept an array of Strings as a parameter?
- Is each word on a new line?

```java
public static void main(String[] args) {
    String[] testArray = {"AC", "TACG", "AAC", "GTT"};
    String longest = findLongest(testArray);
    System.out.println("The longest String is "+longest);

    String[] test2 = {"TG", "TGA", "ACT", "GA", "CCT"};
    longest = findLongest(test2);
    System.out.println("The longest String is "+longest);
}
```



What to look for:

- Does the method accept an array of Strings as a parameter?
- Is the longest word in the array returned?
- If there is a tie, is the word that comes first in the array returned?

Submission

Submit the following completed file to the Assignment folder on conneX.

- `DNASequencing.java`

Please make sure you have submitted the required file(s) and conneX has sent you a confirmation email. Do not send [`.class`] (the byte code) files. Also, make sure you *submit* your assignment, not just save a draft. Draft copies are not made available to the instructors, so they are not collected with the other submissions. We *can* find your draft submission, but only if we know that it's there.

A note about academic integrity

It is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

Grading

Marks are allocated for . . .

- No errors during compilation of the source code.
- The method headers must be exactly as specified and the methods must perform as specified. Be sure to read the **Method Details** in the specification document.
- Each method must have a test call inside the `main` method.
- Style of the source code meets the requirements outlined in the Coding conventions document available in the Lab Resources folder of conneX.