

Coursework 2

Overview

In this coursework, you are provided with a library (testlib.js) which you should not modify, and a template file (demo.js) you can base your implementation on.

Your implementation will provide solutions to each of the tasks described below.

Each task is about matching genetic strings: that is, strings consisting of the letters A, C, G, T.

The strings to be matched are in the taskN.seq files, where N is the task number; the strings inside which the taskN.seq strings need to be matched are in the taskN.data files.

All the files (*.js and task*) should reside inside the same directory.

demo.js includes how to connect to the testlib library, along with how to start the test. The source code of the library is documented on the functions available for your use.

To run the demo, open a terminal and type

```
node demo.js
```

Please note that you must have installed Node.js from <https://nodejs.org/en/download/>

There is a special rule for these tasks: loop structures are not allowed - this means you cannot use any for, while, or do/while, or any other form of these structures.

Instead, you should make extensive use of the Array functions (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) and Object functions

(https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object) to search, match and refine your data. The Mozilla JavaScript documentation can help you with this.

Take a look at the links above, along with the documentation link here:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>.

You will have to do some reading around the documentation to be able to complete these tasks!

You should submit one zip file containing one .js implementation file and a short report: the report is a chance to explain your design ideas, how you tested your code, any issue you faced, or any potentially interesting point arisen during your work.

Task 1: Frequency Counts

You've been asked to produce the counts of particular sequences handed to you from a test framework. The

framework itself has a function for you to call with the results.

In essence, if you have been given the following sequence of nucleotide letters:

CCAAATT

We should be able to generate the following counts of each pattern:

{ AA: 2, AT: 1, CC: 1, CA: 1, TT: 1 }

Note that 'AA' is detected as two patterns, as the first pair of 'A's then a second pair of overlapping 'A's:

CC **AA** ATT and CCA **AA** TT

Your code will be handed the input character-by-character, so it will be up to you how you decide to buffer and detect these sequences.

For this particular task, you are required to use the `testlib.frequencyTable()` function to report your frequency counts. This function expects that you supply it with an object containing keys as specified by the patterns array in the "ready" handler, each with a count of the number of times the program has seen that pattern.

Remember that you can access JavaScript object and array keys by square brackets.

An (extremely simple) example of how to do this might be:

```
let data = { "AA": 1, "TT": 12 };
Testlib.frequencyTable( data );
```

Note: To know when one sequence stop and another begins, use the "reset" hook via the `testlib.on()` function!

To test your code for this task, remember to set `testlib.setup(1)`

Task 2: Short Sequence Matching

Now that you have counting and matching working, you are now tasked with reporting the precise location of each match for each pattern.

`testlib` provides a `testlib.foundMatch(pattern, offset)` function which should be called when a match is found, passing it the pattern found along with the offset in characters from the start of the current sequence.

Again, in the library you have been given, this function will simply print the details it has been provided with.

To test your code for this task, remember to set `testlib.setup(2)`

Task 3: Complex Sequence Matching

You should have Tasks 1 and 2 working fully before attempting Task 3.

The letters A, C, G, T occurring in genetic strings identify distinct nucleotides in DNA sequences.

Unfortunately, in reality it can be quite difficult to identify the specific nucleotide when doing DNA sequencing, so often instead of a specific organic molecule, the sequencing machine is only able to guess at a number of options at a given position in a genetic string.

The machine indicates this by not using one of the definite chemical characters (A, C, G and T), but with one that indicates some combination of these four, according to the table below:

Recorded Character	Possible Actual Nucleotides			
R	G	A		
Y	T	C		
K	G	T		
M	A	C		
S	G	C		
W	A	T		
B	G	T	C	
D	G	A	T	
H	A	C	T	
V	G	C	A	
N	A	G	C	T

LZSCC.212: Advanced Programming

A successful solution to this should be able to report frequency tables and identify matches as per tasks 1 and 2.

Moreover, it should print the total matches for every sequence in task3.seq.

To test your code for this task, remember to set `testlib.setup(3)`

Marking

Marks will be awarded according the following table:

Task	Percentage
Task 1: Frequency Counts	15%
Task 2: Short Sequence Matching	20%
Task 3: Complex Sequence Matching	40%
Code Structure, Elegance, Commenting and Style	15%
Report	10%