Discovery Lab 1: Bioinformatics

CISC181 Fall 2013
Assigned: September 25
Due: October 1 at 11:55PM on Sakai
Text references: Chapters 5, 6, 7, 8, 10, 11

Discovery labs should be completed by a pair of students. You may work alone (although not recommended), but you may not work in a group of 3.

Discovery labs are designed to look at a particular area of Computer Science applications and research. In each lab you will develop a working prototype application that practices fundamental programming techniques. After completing the prototype, your group will apply these techniques to develop a checkpoint for your course project.

In this discovery lab you will explore the area of Bioinformatics.

Objectives:

1. Design, implement, test, and debug a program that uses: basic computation, standard conditional and iterative structures, and the definition of methods.
2. Choose appropriate conditional and iteration constructs for a given programming task.
3. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces.
4. Create algorithms for solving simple problems.
5. Write programs that use the following data structures: arrays, 2D arrays.

* Problems you need to solve and turn in are listed in green text.

One of the most important recent discoveries in biology is that organisms use similar genes to provide their biological functions. A goal of biologists is to perform comparisons between the genes found in one organism and those of another to predict biological functions. Of particular interest in this important area of research are the patterns found in viruses. Identifying similarities can help track the origin and development of virus strains, and can determine useful treatments based on existing practice.

We'll begin by noting that a genetic sequence can be represented as an array of character bases:

```
AGTGCTGAAAGTTGCGCCAGTGAC
```

To find matches between two sequences, we could simply check for equality at each character, giving us the following picture:

```
AGTGCTGAAAGTTGCGCCAGTGAC
|||||||||  | ||
AGTGCTGAAGTTCGCCAGTTGACG
```

However, the above picture can miss similarities. Some sequences may be similar but not detected because they are shifted, or offset from each other. To visualize this, a dot matrix will display a matrix of comparisons. For the above two sequences this looks like:

```
  A G T G C T G A A G T T C G C C A G T T G A C G
 +------------------------------------------------
A|x             x x               x         x
G|  x   x     x     x       x       x     x     x
T|    x     x         x x             x x
G|  x   x     x     x       x       x     x     x
C|        x               x   x x             x
T|    x     x         x x             x x
G|  x   x     x     x       x       x     x     x
A|x             x x               x         x
A|x             x x               x         x
A|x             x x               x         x
G|  x   x     x     x       x       x     x     x
T|    x     x         x x             x x
T|    x     x         x x             x x
G|  x   x     x     x       x       x     x     x
C|        x               x   x x             x
G|  x   x     x     x       x       x     x     x
C|        x               x   x x             x
C|        x               x   x x             x
A|x             x x               x         x
G|  x   x     x     x       x       x     x     x
T|    x     x         x x             x x
G|  x   x     x     x       x       x     x     x
A|x             x x               x         x
C|        x               x   x x             x
```

To get started, import DL1.zip as a new Project in Eclipse and open DL1_Tests.java. Look at the tests for each of the 4 things you will need to implement for this lab.

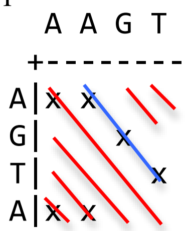
- Implement the constructor for DotMatrix so that it takes two character arrays as parameters and assigns the matrix (two dimensional array) of boolean values.

  There is a test written using several sequences for this method. If your code works correctly, it should also display the dot matrix pattern from the previous page.

The existing dot matrix display is cluttered with random matches and may not be easy for a human to interpret. Many programmatic scoring algorithms have been developed to aid in finding similar subsequences between two sequences. In this assignment we will find the maximum consecutive match in the dot matrix representation. For example:

```
  A A G T
 +--------
A|x x
G|    x
T|      x
A|x x
```

```
With diagonals
   marked:
```



has an alignment score of 3 for this algorithm since there are 3 consecutive matches starting at row 0 column 1 of the dot matrix (shown as the blue diagonal).

- Implement the diagonalMatchScore method in DotMatrix so that it will check the diagonal starting at the given row and column. You will need to iterate diagonally, keep track of how many consecutive matches (boolean true values) you find before you get to the "end" of the matrix. Make sure your code passes the included test.

- Implement the method maxDiagonalAlignment so that it calls the diagonalMatchScore method once for each of the diagonals pictured. Your method should return the maximum score returned from any of the diagonals. Make sure your code passes the included test.

- Implement the method findClosestMatch in the H1N1Virus class. This method will be called on a "mystery" H1N1Virus and we want to find the H1N1Virus in the list of possible sources that is the closest match to the mystery virus. You should be creating DotMatrix objects for each combination, and finding the one with the highest maxDiagonalAlignment. Make sure your code passes the included test.

A high score for a given dot matrix may indicate a close relation between the two tested sequences. This is particularly useful when comparing a virus sequence about which little is known with a database of known sequences. Using this technique can help make informed decisions about treatment options by first trying options that worked on the closest known sequence.

- Once your JUnit tests have passed, run the H1N1Virus class as a Java application. This will read four influenza sequences from a file and use your methods from above to compare the H1N1 influenza sequence from New York with the 3 other provided strains from Rome, Lipetsk, and Guam. Write a comment in the main method of H1N1Virus.java indicating which one of the locations is the most likely source for the New York strain.

Write a comment in the main method of your H1N1Virus.java that includes the name of both people in your group. Please export your modified DL1 project as an archive for submission to Sakai.

For additional information on the application of sequence alignment, see:
http://en.wikipedia.org/wiki/Sequence_alignment