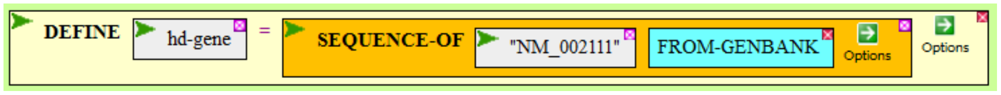
(8 April 2015)

Introduction to Bioinformatics
Problem Set 7: Search Tools and Tables

Search Tools

1. Huntington's disease is caused by an excess of repeats of CAG (encoding glutamine) in the *HUNTINGTIN* gene. Individuals with fewer than 28 repeated units have a normal phenotype, while those with greater than 36 express some symptoms of Huntington's disease. Obtain the sequence of the gene from some individual using the SEQUENCE-OF function with the FROM-GENBANK option, as shown below:



```
DEFINE hd-gene = SEQUENCE-OF "NM_002111" FROM-GENBANK Options Options
```

Use MATCHES-OF-PATTERN to identify the full extent of the CAG repeat in `hd-gene`. Finally, use the coordinates reported by this function to find the CAG repeat in the sequence of the gene, displayed with SEQUENCE-OF `hd-gene`. What do you predict is the phenotype of the person carrying this gene, what evidence do you have for that prediction, and what code did you use to find it? Where is the repeat relative to the gene? If it is in the gene, then are the CAG's in-frame (i.e. are they translated as glutamines)?

2. One of the most commonly used strategies in sequence analysis is the moving window, in which you consider small segments of a much larger sequence, one after the other. Here, you'll use this strategy to make your own DOTPLOT function. Dotplots are a means of comparing one sequence with another to look for local repeats, as discussed in the notes, Repeat Finding Tools.

Your strategy to construct a dotplot will be to slide a window across one sequence and compare that window against the second sequence. The matches for each window will provide information for one line of the dotplot, using the PLOT function to display the dotplot. For example, to construct a dotplot of Sequence **A** vs Sequence **B** (see below), you'll compare the first 11 nucleotides of Sequence **A** to every point in Sequence **B**, noting where matches occur.

Sequence **A**: **CAACCCTCCAT**CATAAAACTTGGGCTTGGGAGGCAGAGCCTAACCTCTCT

Sequence **B**: CAACCCTCCATCATAAAACAACCCTCCATCATAAAACTTGGGCTTGGGAG

A match of the first window occurs in Sequence **B** at positions 1 and 20, so the first line will contain a dot at each of those positions. Then you slide the window one nucleotide to the right and repeat the comparison to obtain the dots for the second line, and so forth.

As a test of the function (and to give us raw material as we build it), we’ll try it on the sequences of two small phage: Chlamydia phage 3 and Chlamydia phage phicpg1 (find the sequences in either Phantome/BioBIKE or ViroBIKE).

Your implementation of DOTPLOT will compare two sequences through a window of 11 nucleotides, grabbing 11 nucleotides at a time from the first sequence and searching the second sequence for matches to it. As you go through each step below, notice the result in the result pane and compare it with what you expect from the actual sequence.

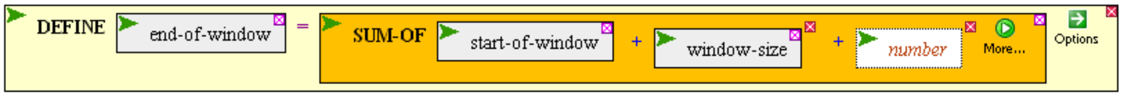
2a. In the end you will use the PLOT function to display the results of your sequence comparisons. To see where you're heading, start off by experimenting with the PLOT function. One way to use PLOT is by providing it with a set of number pairs, such as `((1 1)(2 4)(3 9))`. Play with PLOT until you understand what it does with the pairs.

2b. DEFINE `seq1` as the SEQUENCE-OF Chlamydia phage 3 and `seq2` as the SEQUENCE-OF Chlamydia phage phicpg1.[*]

2c. DEFINE `window-size` as 11 (each comparison will be 11 nucleotides)

2d. DEFINE `start-of-window` as 1 (we'll start the first window at coordinate 1

2e. DEFINE `end-of-window` as… as what? If `start-of-window` is the first coordinate of the window, what is the last? Knowing the start-of-window and the window-size, how would you calculate the end coordinate? Keep in mind that the values of these variables may change. Use SUM-OF to do the calculation. The function will look something like this:


DEFINE end-of-window = SUM-OF start-of-window + window-size + *number* More... Options

What value should go in the *number* box? Check this by counting!

2f. Now use the values you've defined to define the portion of the first sequence you'll try to match with the second sequence. DEFINE `window` as SUBSTRING `seq1` FROM `start-of-window` TO `end-of-window`

2g. DEFINE `matches` as MATCHES-OF-ITEM `window` IN `seq2` (set the options so it searches only one strand). It will be more convenient if the function returns only the position of the match, so specify the +COORDINATES option. Try it with and without the option so you can see what I mean.

2h. Now we need to turn this information into number pairs. The first number in each pair will be the coordinate of the window (for simplicity, we'll use `start-of-window`). The second number of each pair will be the coordinate of a match. A pair will be formed by:

```
LIST start-of-window  n
```

where n is one of the coordinates returned by MATCHES-OF. We don't know how many matches will be found, but we can MAP the LIST function over the list of matches:

```
APPLY-FUNCTION
    LIST start-of-window n
    REPLACING n WITH matches
```

DEFINE number-pairs as this function (i.e., APPLY-FUNCTION …)

If all has gone well, you should have the number pairs necessary for PLOT to display one line of a dotplot of `seq1` vs `seq2`. To generalize, i.e. repeat these operations over all windows within `seq1`, we need a loop.

[*] Of course the names of the first phage will not be literally "Chlamydia phage 3". That's just the human name for the phage. Use the usual tricks to find the name of the gene as known to BioBIKE. By the way, NAME-OF [organism] SHORT will give you a short nickname,.

2i. Bring down a FOR-EACH loop. We'll treat `seq1`, `seq2`, and `window-size` as constants for the moment. The loop will presume they already exist. The loop will repeat lines **2e** through **2h** and loop through all values of `start-of-window`, FROM 1 TO (nearly) the end of the sequence. Open up the body of the FOR-EACH loop and create holes for each function made in lines **2e** through **2h**. In the past you've made holes in the BODY of the loop and then dragged each function into those holes. That's relatively simple, but the loop will execute 2- to 3-times faster if you instead make it as I describe below.

2j. Mouse over the **update** icon (next to *VARIABLE UPDATE SECTION*) and click ASSIGN. Repeat this three more times (the **update** icon will move to the right). Now move the boxes within the four DEFINE blocks into the ASSIGN blocks, maintaining the order.

2k. Open up a primary control *number from n1 to n2* box. Replace *var*iable with `start-of-window`. The first value should be 1 of course, so that the window begins at the beginning of `seq2`. What about the last value? Surely it's related to the length of `seq2`. Is it equal to the length? Devise a calculation for the last value and put it in the last value box. If you don't know how to calculate the number, make up a simpler case, say a sequence composed of just 11 letters.

2l. Open up a Results Section, using WHEN… APPEND. Why WHEN? You don't always want to add to the results, just when you found a match. Why APPEND? The difference between APPEND and COLLECT is subtle. We may explore it later. For now, trust me.

2m. WHEN what? WHEN matches has a value. That's expressed simply as WHEN matches. Put the variable matches in the condition box.

2n. APPEND what? APPEND all the number-pairs you found. Put the variable number-pairs in the value box.

2o. Execute the loop. Do you get the expected collection of number pairs?

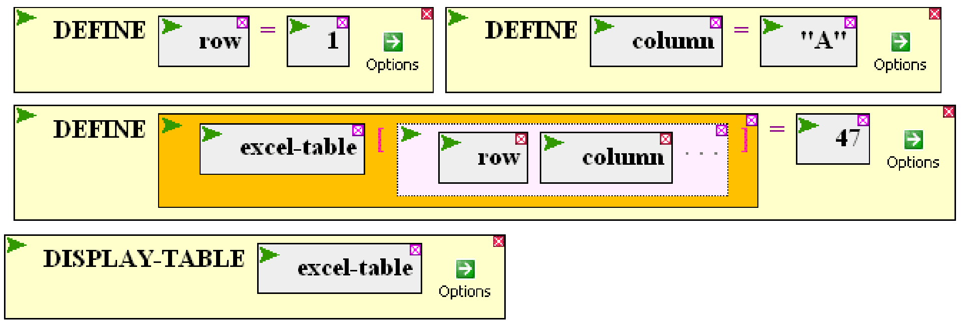
2p. Put the loop within a PLOT function. The easiest way to do this is to mouse over the action icon of FOR-EACH, click Surround, and then click PLOT in either the INPUT-OUTPUT menu or the ALL menu. The PLOT function by default plots lines, but you're doing a ***dot*** plot, so choose the POINTS option from the option menu of PLOT. Execute the resulting function.

2r. How do you interpret the dotplot?

Tables and the [ ] notation

It is often incredibly helpful to be able to create a set of information and refer to individual elements of the set. For example, you might consider the subjects of an experiment as S_1, S_2, S_3, etc. Or you might want to divide information into different categories, such as GNP_{Europe}, GNP_{China}, etc. In computer languages, this is achieved through data structures sometimes called arrays (if the subscripts are numbers) or hashes (if the subscripts are anything else). In BioBIKE, the two are combined into a concept called *Tables*.

3. You can create something like an Excel table by defining the contents of each cell in the way shown at the top of the next page. Note the use of the **[ ]** function (obtainable from the **LISTS-TABLE** menu) to access a specific cell of a table. You can think of **[ ]** as meaning "subscript". Replace `row` and `column` with different numbers and letters and re-execute the definition. Then display the table. From these experiments, how does BioBIKE think of tables?

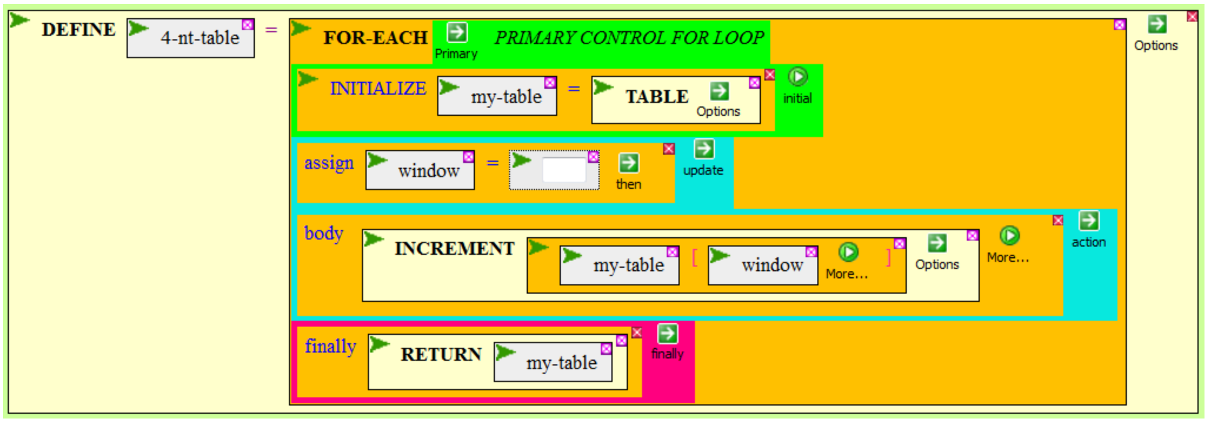


```
DEFINE row = 1
DEFINE column = "A"
DEFINE excel-table [ row column ... ] = 47
DISPLAY-TABLE excel-table
```

4. Now for something a bit more bioinformational. Suppose you want a count of all 4-nt sequences in a genome. Of course you could do this as you have done previously, using mapping COUNT-OF over a list of all possible 4-nt sequences. But that would take a while – after all, you're searching the entire genome 256 times! It's much faster to search it only once, using a moving 4-nt window and counting each 4-nt instance as you go.

4a. Make a loop that moves a 4-nt window over the lambda genome as shown below. I've left the steps of moving the window to you. You'll have to complete the primary control section and the variable update section. I've done the rest.

Each iteration of the loop, the sequence found in the moving window will serve as the index of a table that will count how many times you've seen that sequence. The table is initialized before the loop begins, incremented each iteration in the body of the loop, and returned after the loop has finished its final iteration.[†] DEFINE a variable to contain the table returned by the loop, then use DISPLAY-TABLE to show you what you've got.



```
DEFINE 4-nt-table = FOR-EACH  PRIMARY CONTROL FOR LOOP
                      INITIALIZE my-table = TABLE
                      assign window =
                      body INCREMENT my-table [ window ]
                      finally RETURN my-table
```

4b. The results may be more comprehensible if you sort the loop. But only lists, not tables, can be sorted. So CONVERT the table to a List, SORT the resulting list by counts (not sequence), then use DISPLAY-LIST to display the sorted list. Look carefully at the sorted list. What sense can you make of it?

4c. Solve the same results using COUNTS-OF-K-MERS and compare the results with those of **4b**. Do you think that the algorithm implemented in **4b** is the same as that implemented by COUNTS-OF-K-MERS?

[†] If you're unclear on the parts of a loop, see Elements of a Loop.