CS1132 Spring 2015 Assignment 2 Due April 15th

Adhere to the Code of Academic Integrity. You may discuss background issues and general strategies with others and seek help from course staff, but the implementations that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is never OK for you to see or hear another student's code and it is never OK to copy code from published/Internet sources. If you feel that you cannot complete the assignment on you own, seek help from the course staff.

When submitting your assignment, follow the instructions summarized in Section 4 of this document.

Do not use the `break` or `continue` statement in any homework or test in CS1132.

1 Drawing Networks

Networks have become increasingly important in many areas of science and technology, from biology to sociology, to traffic modeling, and to computer and information science. Examples of networks in these areas include neurons in the brain, road networks, social networks such as *Facebook*, and, of course, the Internet. A network consists of a set of *nodes* and the connections between pairs of nodes, also known as *links* or *arcs*. Consider a road network: cities are represented as nodes while the highways that connect the cities are the links. For a social network, the nodes can represent people while the links represent friendships. Networks can be complex, with thousands or millions of nodes and complicated connection patterns among them. Therefore, network visualization poses a serious challenge to researchers.

In this problem, you will develop an interactive MATLAB application `networkEdit` to aid in the drawing of neat and easy to interpret networks, such as the one shown in Figure 1.

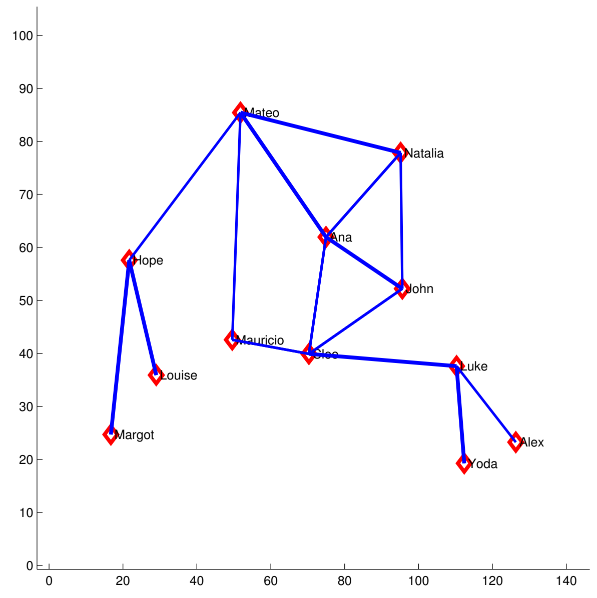

Figure 1: Final network configuration after running `networkEdit` and doing interactive edition.

Note that links are drawn in different widths, representing the "strength" of a link. In a social network, for example, the strength may be quantified as the time that two people have known one another. A simple way to encode the structural information of a network with n nodes is to set a matrix `weights`, where `weights(i,j)` contains the strength value of the link between nodes `i` and `j`. The minimum strength value is 0, indicating that there is no link between two nodes. Implement this function:

```
function  finalPositions = networkEdit(weights, names, nodeStyle, linkStyle)
% Draw a preliminary network, allow user to modify node positions by clicking on the figure,
% and return the final node positions.
% Input parameters:
%    weights  --  an n-by-n symmetric matrix of doubles between 0 and 1, representing
%                 the ``strength'' links
%                 weight(i,j) is > 0 if there is a connection between nodes i and j
%    names    --  a length n cell array of strings containing the text labels of
%                 the nodes.  The labels should be displayed next to the nodes.
%    nodeStyle--  format string for drawing the nodes.  E.g., 'k*' for black asterisks,
%                 'rd' for red diamonds, etc.
%    linkStyle -- format string for drawing the links.  E.g., 'b-' for blue solid line,
%                 'g:' for green dotted line, etc.
% Output parameter:
%    finalPositions  -- an n-by-2 matrix of doubles containing the final coordinates of
%                       the n nodes.  For k=1,2,...,n, finalPositions(k,:) contains the
%                       the final x and y coordinates of node k.
```

The example network shown was produced after running the following command and moving some nodes around:

```
finalPositions = networkEdit(weights, names, 'rd', 'b-')
```

The following are the detailed specifications of function `networkEdit`:

- Initialize the coordinates of the nodes to produce an initial network configuration.
- * Display the network configuration, including the nodes and their labels and the links. The width of the link between nodes `i` and `j` corresponds to `weight(i,j)`. Make things look nice, with appropriately sized node markers and thick enough lines.
- Ask the user to select a node to be moved using a mouse click. This instruction appears as the title of the plot. Clearly mark the selected node.
- Ask the user to select a destination position for the selected node. If the destination position is not close to any node, then update the selected node's position to the destination position. If the destination position is close to a second node, then swap the positions of the two nodes.
- Go back to * above
- Stop if the user clicks outside of the plot region at any time.
- Throughout the user interaction, refresh the plot title with appropriate instructions or messages. E.g., "Node 4–Natalia was selected. Select a destination position" or "Natalia and John have swapped positions"

1.1 Hints

- Take a close look at (and experiment with) the example in Module 2 Part 3 No. 3 to get a better feel for how to use graphic commands and interactivity.
- The "Example graphics files" section In Module 2 Part 3 No. 2 shows commonly used controls. Especially relevant items include "Axis Freezing," "Controlling Line Width," and "TextAlignment."
- The statement `[a,b]= ginput(1)` stores the location of a user mouse click in variables `a` (x-coordinate) and `b` (y-coordinate).
- Remember that you can read the documentation on MATLAB commands by typing `help` *command_name* in the Command Window.

- To refresh a figure, use the command `clf` followed by `hold on` to make sure that all subsequent graphics commands add to the current set of axes.
- *Encapsulate pieces of your code in subfunctions!* (Subfunctions are simply functions written in the same file as the main function, `networkEdit` in this case.) Consider writing subfunctions for each of the following tasks:
  - Initialize the positions of the nodes to some arbitrary positions
  - Draw (refresh) the network plot with given positions for the nodes
  - Determine whether a position clicked is inside or outside of the plotting area
  - Determine whether a position clicked is close enough to any of the nodes to consider that mouse click to have selected a node.
  - Translate a weight into an integer width to use when plotting a link
- If you want to test your function on the same example used to produce Figure 1, you can download file `networkData.mat` and load it into the workspace with the command `load networkData.mat`. This will load the `weights` matrix and the `names` cell array used to produce this example.

2 Finding Similar Genomic Sequences

Many questions in biology are investigated with the help of efficient computational methods for searching, comparing, and organizing huge amounts of sequence data. Both DNA sequences and protein sequences can be treated as character strings. For DNA there are 4 possible letters: A, C, G, and T corresponding to the 4 bases adenine, cytosine, guanine, and thymine, respectively. For proteins there are 20 possible letters: A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, each representing one of the 20 amino acids. One frequently used technique in computational biology is the comparison of sequences to find a "match"—or just similarity. For example, scientists have successfully predicted some proteins' function and/or 3-dimensional structure by extrapolating from similar proteins with known function and 3-d structure.

Consider the proteins that appear in, say, a mouse. For many of them there are similar proteins that appear in a rat or a cat. Although such proteins accomplish the same or similar functions, they are not identical; there are substitutions, insertions, and deletions of amino acids. There are corresponding substitutions, insertions, and deletions in the portions of the DNA that encode the information a cell uses to build the proteins.

You will write an application for finding subsequences (in a long sequence) that are similar to a query sequence. Solving this problem in full generality requires the use of Dynamic Programming, an algorithm design technique that is studied in upper level courses such as CS4820, Introduction to Analysis of Algorithms. In this assignment, we restrict our definition of "similar" to allow just substitutions and a limited number of insertions. You will write a function `proteinMatch` such that the statement

```
n= proteinMatch('genome.txt', 'protein.txt', 'result.txt')
```

reads sequence data from a file called `genome.txt` in the current directory, reads sequence data from a file called `protein.txt`, identifies all subsequences in the genome data that are similar to the protein sequence, and writes the findings to a file called `result.txt`. The number of similar subsequences found is stored in variable `n`.

2.1 Defining a Match

1. Allow one nucleotide to substitute for another. T is similar to A, and C is similar to G. Consider the following example:

```
AATGCCCAACCG
 ATCC
```

The top string is the data and the bottom string is the query. If we consider C and G to be similar then there is a match that begins at position 2 of the data string. We assign a "penalty" for each substitution: an A-T substitution draws a penalty of 1 and a C-G substitution draws a penalty of 2. We can say that a match is found based on some allowed penalty and the "match position" is the position (index) in the data where the beginning of the match occurs. For the above example:

Allowed Penalty	*Match Position(s)*
0	*none*
1	8
2	2, 8
3	2, 8

2. Allow the insertion of nucleotides. This is commonly described as a *gap*. In this example we do not consider substitution.

```
AATGCCCATGGG
 AT    AT
```

Our query string is ATAT, which is not found in the data. However, if we allow a gap of length 4 to be inserted in the query, as laid out above, then we say that we have a match. We restrict our search to allow only one continuous gap. The penalty for each gap space is one. (So a gap of length 4 draws a penalty of 4.)

Combining these two ideas, we define two (sub)sequences to be similar—we say they match—if the gap (if one is needed) has a length of no more than 3 and the combined penalty for matching is less than or equal to 4.

2.2 File Format

Your program writes the match results in an ASCII (plain text) file. Write three lines of text for each match found:

1. The "match position," which is the position (index) in the genome string at which the match occurs
2. The subsequence of the genome string that matches, starting from the "match position"
3. The protein sequence with any inserted gap filled in by the `*` character

2.3 Example

Example of the genome data file:

```
//A bogus example genome
ATGAAAGGTCATGGGTATTAGTCATAGCTGATCATGATCATGAT
TGATGATAGATTATATGCTCGCGTAGATGCTATATTATGCGCTAGAATC
```

Example of the protein data file:

```
//A bogus example protein
TGAATC
```

Given the genome and protein data above, the contents of an example output file is shown below. There are 11 matches. Depending on your algorithm, the matches may appear in a different order in your output file.

```
2
TGAAAG
TGAATC
2
TGAAAGGTC
TGAA***TC
2
```

```
TGAAAGG
TGAAT*C
2
TGAAAGGTC
TGAAT***C
29
TGATCATG
TGA**ATC
32
TCATGATC
TGA**ATC
35
TGATCATG
TGA**ATC
41
TGATTG
TGAATC
48
TGATAG
TGAATC
87
TAGAATC
T*GAATC
88
AGAATC
TGAATC
```

2.4 Data Files

The data files `humanC5_?.txt` where `?` is 1, 2, or 3, are portions of the human chromosome 5 nucleotide sequence downloaded from the data bank of the European Bioinformatics Institute. The sequences in the files are approximately 100, 1000, and 10000 bases long, respectively. The files contain ASCII characters (plain text). Each file begins with a line of text that identifies the sequence but *is not* part of the nucleotide sequence. The remaining lines in the file contain the sequence in capital letters and are of varying lengths. Your code should work with the given files—do not modify the files in any way!

2.5 Built-in functions

Below is a list of useful built-in functions for handling characters, strings, and files. *Use only these built-in functions* for handling characters, strings, and files. You may not need all of them. You can of course still use general built-in functions not related specifically to strings and files, such as length, zeros, rem,$\cdots$ etc.

You can use:

- `fopen, feof, fgetl, fclose` (Hint: Module 2 Part 4 (File I/O) Items 1a and 1b are extremely relevant.)
- `strcmp`

You must NOT use Matlab built-in functions `find`, `strfind`, `findstr`.

2.6 Hints

- Break down the problem and make use of subfunctions! For example, a really useful subfunction is one that determines whether there is a match and calculates the penalty given two strings of the same length. For example, given the strings `TCATGATC` and `TGA**ATC` the subfunction should return 4 (there is a match) and given the strings `ATATATAT` and `CGCGCGCG` the subfunction can return 5 (let a value greater than the maximum allowed penalty indicate no match; write the subfunction such that it returns as soon as the accumulated penalty is greater than the maximum allowed penalty).

- You can design your own algorithm, but here are some ideas ... Consider each genome substring that begins at index j (what is the correct range for j?). (a) Is there a match without any gap? (b) What are the matches when a gap is present? Problem (b) decomposes further: A gap can be inserted at positions $2, 3, \ldots, n$ where n is the length of the protein. At each of those insertion positions, the gap can be length 1, 2, or 3. Be very careful with the bounds of your loops. Similarly, be careful with subarray indices. For instance, the length n substring of s that begins at index j is `s(j:j+n-1)`, not `s(j:n-1)` or `s(j:j+n)`.

- Copy and paste the example genome and protein data above into plain text files so that you can use them for testing when you develop your program.

3 Self-check list

The following is a list of the minimum *necessary* criteria that your assignment must meet in order to be considered *satisfactory.* Failure to satisfy any of these conditions will result in an immediate request to resubmit your assignment. Save yourself and the graders time and effort by going over it before submitting your assignment for the first time.
Note that, although all of these are necessary, meeting all of them might still not be *sufficient* to consider your submission satisfactory. We cannot list everything that could be possibly wrong with any particular assignment!

- Δ Comment your code! If any of your functions is not properly commented, regarding function purpose and input/output arguments, you will be asked to resubmit.
- Δ Suppress all unnecessary output by placing semicolons (;) appropriately. At the same time, make sure that all output that your program intentionally produces is formatted in a user-friendly way.
- Δ Make sure your functions names are *exactly* the ones we have specified, *including* case.
- Δ Check that the number and order of input and output arguments for each of the functions matches exactly the specifications we have given.
- Δ Test each one of your functions independently, whenever possible, or write short scripts to test them.
- Δ Check that your scripts do not crash (i.e., end unexpectedly with an error message) or run into infinite loops. Check this by running each script several times in a row. Before each test run, you should type the commands `clear all; close all;` to delete all variables in the workspace and close all figure windows.

s

4 Submission Instructions

1. Upload files `networkEdit.m`, `proteinMatch.m` onto the Assignment 2 area in CMS.
2. Please don't make another submission until you have received and read the grader's comments.
3. Wait for the grader's comments and be patient.
4. Read the grader's comments carefully and think for a while.
5. If you are asked to resubmit, fix all the problems and go back to Step 1! Otherwise you are done with this assignment. Relax.