CS99, FINAL PROJECT, Fall 2002

Due Friday 12/6/2002 4:00:00 PM

1. Goals

This assignment will help you develop skills in software development. You will:

- develop software that uses one- and two-dimensional arrays of characters
- determine good sample cases to test your software
- solve some problems in computational biology using your software

2. Motivation

Today's advanced technology can perform DNA sequencing on a truly industrial scale. Research generates new data at an explosive rate. Starting about 1984, the volume of DNA-sequence data has doubled approximately every 15 months! This flood of information provides one of Biology's greatest challenges – transforming this raw data into meaningful information. The field of ***computational genomics*** endeavors to provide biological "understanding" from this data-flood through mathematical models and computer algorithms. Since DNA is so fundamental to Biology, advances in this field will impact medicine and agriculture, and will improve insights into Biology.

3. Background

This section presents material for those who wish to know the underlying concepts. If you don't understand some of the biological details but understand enough to do the assignment, that's O.K.

3.1 Genome

Let's back up a little. What's a ***genome***? And, why is it important?

A genome is all the DNA in an organism, including the organism's genes. A ***gene*** carries information for making all the proteins required by all organisms. These proteins determine, among other things, how the organism looks, how well its body metabolizes food or fights infection, and even how the organism behaves.

3.2 DNA

DNA literally means *deoxyribonucleic acid*. But, what is it?

A group of similar chemicals called ***nucleotides*** make up DNA. The four types of ***nucleotides***, also called ***bases***, are Adenine, Cytosine, Guanine, and Thymine. Don't worry about the names of the four nucleotides. Instead, refer to each nucleic acid by its first letter. You can now refer to the four nucleotides as A, C, G, and T. These bases repeat millions or billions of times throughout a genome. The human genome, for example, has 3 billion pairs of bases.

You may have heard that DNA forms a ***double helix***. A double helix has two strands of DNA that bind together and twist into a helical shape. For this assignment, ignore this fact, and think of only a single strand of DNA. The other strand in the double helix is always "complimentary." Knowing what one strand "looks" like (i.e., which nucleotides compose the strand) uniquely determines what the complementary strand looks like. So, considering only one strand of DNA does not discard any essential data.

DNA is essentially a linear molecule. Each base strongly binds to no more than two other bases. These two bases bind before and after the original base. Considering other bases and their "links" provides a model of DNA-strand as a sequence of nucleotides. The particular order of As, Ts, Cs, and Gs is extremely important! The order underlies all of life's diversity, even dictating an organism's species. Different orders generate humans, yeast, rice, or even fruit flies.* Because all organisms relate through similarities in DNA sequences, insights gained from nonhuman genomes provide new knowledge about human biology.

---

*. In fact, each of these species has a full-scale genome project devoted to mapping, and understanding the entire genome.

3.3 A Computer Scientist's View of DNA Sequences

Computer scientists think of a DNA sequence as a string from an alphabet of 4 characters: A, T, C, and G. This model encompasses the idea that the four types of bases arrange in any order and that the order is important. Why? Just imagine that the English strings *mate*, *meat*, *team*, and *tame* all meant the same thing!

3.4 DNA Sequence Analysis

DNA Sequence Analysis describes the portion of computational genomics involved with creating and using tools. These tools perform ***data reduction*** by taking large amounts of DNA sequence data and deriving smaller amounts of data which are more easily understood. Some of this derived information includes:

- Comparing DNA sequences from different species or from different places in one species' genome to investigate similarities. Your assignment will focus on this approach.
- Determining which portions of a DNA sequence are the "coding regions" (genes) and which portions regulate the genes. Regulating a gene means determining when the gene is "turned on," meaning when the protein encoded by the gene is produced.
- Compiling statistics about large sections of DNA, such as the percentage of As or the percentage of As plus Ts, or the number of times the substring "ATA" appears.

3.5 Rationale for DNA Sequence Comparison

DNA sequence comparison provides many uses. Current scientific theories suggest that very similar DNA sequences have a common ancestor. The more similar two sequences are, the more recently they evolved from a single ancestral sequence. For this reason, measures of sequence similarity help reconstruct ***phylogenetic trees***, sometimes known as a "tree of life." A phylogenetic tree graphically shows how long ago various organisms diverged and which species are closely related.

Very similar DNA sequences encode similar proteins, which perform similar functions. Biologists compare newly discovered genes with all known DNA sequences and hypothesize that the new gene functions similarly to the genes which it closely resembles. Laboratory biologists then test this hypothesis in the lab and greatly shorten the time learning the behavior of the new gene.

4. DNA Sequence Comparison

This section develops algorithms for performing a problem of computational genomics.

4.1 Simple Distance

The number of bases two DNA sequences have in common provides a simple measure of similarity. A converse criteria called ***simple distance*** measures the dissimilarity by counting the number of positions with different bases. The ***Hamming distance*** is a more formal term for simple distance†. For example, inspect the two DNA sequences shown in Figure 1. Because two nucleotides differ (2nd and 8th position), the DNA sequences have a Hamming distance of 2.

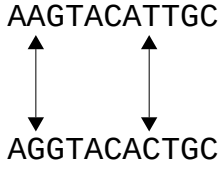

Figure 1: Hamming DIstance

4.2 Weighted Distance

The Hamming distance assumes that all differences are "created equal." For example, the difference between A and G matches the difference between A and C. However, the model of Hamming distance oversimplifies the actual biological process. Due to size, shape, charge, and other chemical properties of the various nucleotides, some bases

†. Check out EE 445 (Computer Networks and Telecommunications)

replace others more easily. For instance, a G more easily replaces an A than a C does. Therefore, applying a weight to each difference between two DNA sequences helps improve the model of distance. Substitutions more likely to occur during evolution have a lower weight than those less likely to occur.

4.3 DNA substitution matrices

A ***substitution matrix*** typically arranges the weights in a 2-dimensional array, as depicted in Figures 2, 3, and 4:

- Each row corresponds to the base in one sequence, or the ancestral base.
- Each column corresponds to the base in the other sequence, or the evolved base.

A number occupying (row, col) = (A, T) location represents the weight for changing A to T. The diagonal values (A, A), (C, C), (T, T), and (G, G) are zero. Why? There's no difference between A and A! Figure 3 demonstrates an example substitution matrix.

The Hamming distance can be considered a weighted distance with an equal distance between all differing nucleotides. Figure 4 shows the weighting matrix for the simple distance. Thus, the weighted distance described here is often called the ***weighted Hamming distance***.

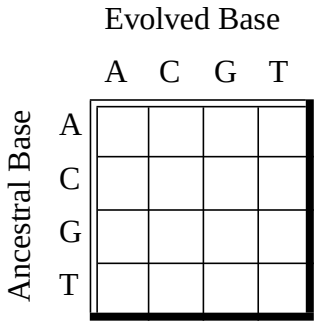

Figure 2: Substitution Matrix

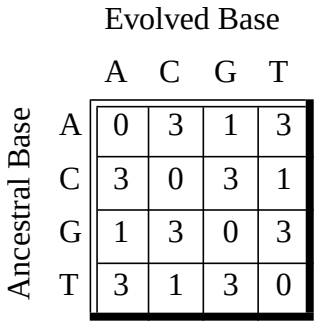

Figure 3: Example Values

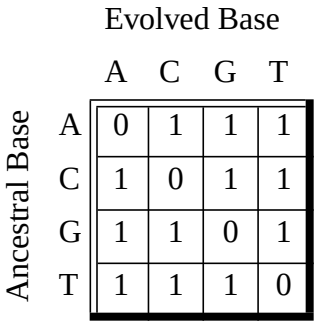

Figure 4: Simple DIstance

4.4 Matrix Notation

You can express the matrix in Figure 3 as M. To reference a particular element at the coordinate of row i and column j, use the notation M_{ij}. For instance, $M_{11} = 0$ and $M_{12} = 3$.

4.5 Matrix Symmetry

This matrix has a special feature you might notice. Inspect the matrix in Figure 5. The four shaded elements M_{11}, M_{22}, M_{33}, and M_{44} comprise the ***main diagonal*** of the matrix. Now, look at elements in directions perpendicular to the main diagonal. As shown in Figure 5, you will find that these elements match. For example, the fourth row has the same values as fourth column.

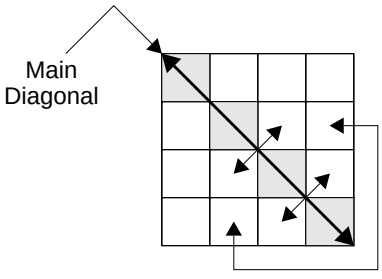

Figure 5: Matrix Symmetry

These conditions give ***matrix symmetry***. In general, a symmetric matrix has the condition

$$M_{ij} = M_{ji}. \quad (1)$$

A symmetric substitution matrix helps model distance. You can assume that the distance between two bases is identical regardless of which direction you choose. For instance, the distance going from A to G matches the distance going from G to A. Consequently, the distance from one DNA string to another is the same in either direction.

5. Problems

The following sections indicate the problems that you must solve. Be sure to read everything before starting. You should also review Section 6 before starting to program.

5.1 Hamming Distance

Create a function M-file called **`hammingistance.m`** that returns the hamming distance between two strings. The function must ensure that both strings have only "legal" characters and the same length. To check the strings, the function calls another function M-file called **`checkchars.m`** that handles just the checking. Note that "legal" characters consist of lowercase A, C, G, and T. When the strings contain "illegal" characters, **`checkchars`** should terminate the program and produce an error message.

5.2 Weighted Distance

Create a function M-file called **`weighteddistance.m`** that returns the weighted distance between two strings. The function must ensure that both strings have only "legal" characters and the same length by calling another function M-file called **`checkchars.m`**. To obtain the weights, create a text file called **`subsmatrix.dat`** that stores the substitution matrix in rows and columns, using the information in Figure 3. For **`weighteddistance`**, you might need one or more subfunctions to help with the computation.

5.3 Random Method

Create a function M-file called **`randommethod.m`** that outputs the hamming and weighted distances between two strings. This function chooses a random string length from 1 to 100. Using the randomly generated string length, the function fills two strings with randomly chosen, but "legal," characters. For this function, you might need one or more subfunctions to assist with the computation.

5.4 Manual Method

Create a function M-file called **`manualmethod.m`** that outputs the hamming and weighted distances between two strings. This function requires the user to enter two strings with equal length that contain only "legal" characters. You should allow the user to enter either lowercase characters because you will use MATLAB's **`upper`** function to convert the strings to uppercase. If the user enters "illegal" characters or mismatched string lengths, do *not* exit from the program. Instead, the function should force the user to re-enter one or both strings. The function (or a subfunction) must display the "illegal" string with an indication of which characters are "illegal." The following output demonstrates this behavior:

```
Enter first string: qwaq
This string is bad! Here is why:
QWAQ
^^ ^
Enter first string:
```

5.5 File I/O Method

Create a function M-file called **`fileiomethod.m`** that outputs the hamming distance for a collection of strings stored in a file. This function must prompt the user for an input file that stores the strings and an output file to store the results. You may assume that the input file contains strings with only "legal" characters. The strings may be different lengths, but you assume the strings are sorted from top to bottom in increasing string length. As your function reads each string, the function must compute just the simple distance with the next string. If the next string is a larger size, the function should output `No more lines of length` *`currentlength`*.

5.6 Driver

Create a script file **`finalproject.m`** that "drives" the entire project. This script will prompt the user to choose the random, manual, or file I/O methods, which are developed in Sections 5.3, 5.4, and 5.5. This script will generate the following output:

```
Welcome to the CS99 Final Project!
Press any key to continue
```
<clear screen>
```
Enter choice of input:
(r) Program generates 2 random strings
(u) User enters 2 strings
(f) Read strings from a file
[r,u,f]:
```

When the chosen task is completed, the script will call another script called **`restart.m`** that prompts the user to repeat or quit the program.

6. What To Hand In

You must avoid redundancy, which usually means creating functions and subfunctions. Judicious use of logical arrays and other built-in MATLAB features may also trim many portions of your code. Be sure to comment all major variables and portions of your code. Zip all of your M-Files together in a file called ***`yournet-id`*`_h8cs99.zip`**. E-mail the zip file to **`cs99@cs.cornell.edu`** *before* the deadline. You must submit a *properly* bound hardcopy of all your M-Files to a staff member (DIS, Gun, or Beth) during office hours on or before the deadline. Late submissions will not be accepted.