Problem Set #5

Due Tuesday, February 9, 2010, at the **beginning** of class. Assignments turned in more than 10 minutes after the beginning of class will be penalized.

Be sure to read each question carefully and answer all the parts.

1. (20 points) From the following alignment, compute the UPGMA tree using Jukes-Cantor corrected distances (if you want, you can compute the correction using the program you've written). You can compute the tree by hand, but briefly give your logic step by step. Show the table of distances as well as the tree, and include your computed branch lengths on the tree rounded to three decimal points.

```
seq1  ATATATCCGATCTGGCGAGCCTCTCGGACGATCG
seq2  ACAGATCCGATCTGGCGGGCCTCTCCGACGGTCG
seq3  GTAGATCTGGTCTGATGCGTCGCTCTAACGGTCG
seq4  ATAGATCTGATCTGACGCGTTCCTCGAACGGTCG
```

2. (10 points) If you were given a fifth sequence and were told it is known to be an outgroup (first diverging sequence) to the above four sequences, what would be the smallest number of raw sequence changes you would expect it have relative to each of the four above.

3. (10 points) Imagine that you could take a time-machine to 50 million years in the future. What tree topology would you expect if you could sample the four sequences (from problem 1) from their descendants at that time? What qualitatively would happen to each of the branch lengths? (This alignment is so short that stochastic effects might play a big role - just assume that you are actually working with a much longer alignment that gives the same tree as you got in problem 1.)

4. (10 points) Figure out how many possible Nearest-Neighbor Interchanges there are on a specific unrooted tree with 8 leaves (that is, the number of competing trees that would be considered in one step of the hill-climbing method using NNIs). Hint: a subtree can be any part of the tree, including a single leaf.

5. (10 points) Modify the program below (my answer for problem 1 on PS3) so that the matrix-filling part is done in a function. The function should have a list of lines as its argument; the rest I leave up to you. Save your function in a module and load the module to use the function. Show BOTH the code in the module and the main program in your answer. As always, beware of newlines.

```
import sys
matrixFile = open(sys.argv[1], "r")
matrix = []                             # initialize empty matrix
line = matrixFile.readline().strip()    # read first line
while len(line) > 0:                    # until end of file
    fields = line.split("\t")           # split line on tabs,
giving a list of strings
```

```
    intList = []                          # create an int list to
fill
    for field in fields:                  # for each field in current
line
        intList.append(int(field))        # append the int value of
field to intList
    matrix.append(intList)                # after intList is filled,
append it to matrix
    line = matrixFile.readline().strip()    # read next line and
repeat loop
matrixFile.close()

for row in matrix:                        # go through the matrix row
by row
    for val in row:                       # go through each value in
the row
        print val,                        # print each value without
line break
    print ""                              # add a line break after
each row is done
```

6. (20 points) Write a program that finds the most common word (or words) with N or more letters and the number of times they appear in a text document, where N and the document name are command-line arguments. Ignore case (Natasha and natasha are the same word) and be sure to get rid of the common punctuation marks `(.,'"?;:-()!)`. Use a function as part of your program. In addition to showing your program, give the answer for N from 4 through 9 in Tolstoy's War and Peace (use this English translation so we should all get the same answer). By the way, you will learn a much better way to remove a set of characters from a string in the next few classes.

7. (10 points) Adapt your program in problem 6 so that it finds the most common word (or words) of length exactly N in a long DNA sequence. In this case, the definition of a word of length N is just N adjacent residues (i.e. a sequence of length X will contain X-N words). Don't worry about the opposite strand of the DNA, but be careful to account for newlines, ignore case, and exclude words containing any N residues (N as in the ambiguity nucleotide character, not the length). In addition to writing the program, run it on the chr21.txt file (linked from the web site) and give the answer for N from 8 through 10. Note - this will take several seconds to run unless you are very very clever. If you get a MemoryError, write another program to cut chr21.txt in half until your program runs.

8. (10 points) Write a program that sorts the words from a text file according to their word length, with the longest word first (descending sort), and prints the longest N words. If the words are equal in length, be sure to sort them alphabetically. Ignore case and remove punctuation as in problem 6. Also be sure to get rid of duplicates of the same word. In addition to showing your program, show the 20 longest words in War and Peace. (what the heck are tatterdemalions?) Hint - you only need one sort function; it should account for both length and (if equal length) alphabetical comparison.

9. Challenge problem. Modify your program in question 7 so that it is much more robust to memory errors. Hint - think about what objects are occupying a lot of memory and work to remove or reduce them. This may be tricky.

10. Challenge problem. Implement the Smith-Waterman algorithm, using a linear gap penalty. It will be easier if you do it first without the traceback and then add that afterwards (or skip it altogether). It will be simpler if you do it just for DNA and hard-code your own substitution matrix.

11. Challenge problem. Implement a parsimony tree-building algorithm from scratch. Hah hah, just kidding. Ain't he a riot.