Problem Set #4

Due Tuesday, February 2, 2010, at the **beginning** of class. Assignments turned in more than 10 minutes after the beginning of class will be penalized.

The first 4 questions use the following alignment:

```
Red Panda     ATCCGTATA
Giant Panda   ATCTGTAAA
Raccoon       ATTTGCAAA
Dog           CTCTGCACA
```

1. (9 points) Draw all possible unrooted trees of these four species.

2. (9 points) Mark the mutations in these data on each of your trees. Give the final parsimony score of each tree (minimal number of nt changes across whole tree).

3. (10 points) If we can assume that dogs are an outgroup (in other words, the root of this tree is on the branch leading to dogs), are giant pandas more closely related to red pandas or to raccoons?

4. (17 points) Make a distance matrix of the all pairwise raw distances among these four species. For distance use number of nt changes divided by alignment length.

5. (10 points) Print the corrected Jukes-Cantor distance for a raw distance of 7 changes in 100 nt. Hint: ln is the natural log, available as the log function in the math module. The Jukes-Cantor formula is:

$$D_{jc} = -\frac{3\ln(1-4D_r/3)}{4}$$

where ln is natural log, D_{jc} is corrected distance, and D_r is raw distance

6. (15 points) Write a program `jcd.py` that takes a raw distance as command line input and returns the Jukes-Cantor corrected distance. Print an error message if the raw distance is outside the range [0,0.75). "[" means including 0, ")" means excluding 0.75. Print an informative message if the user forgot to provide an argument. Be sure to test your program with values at or outside the boundaries. Hint: ln is the natural log, available as the log function in the math module.

7. (15 points) Write a program `seqdict.py` that reads a file of sequences (1st argument) and gives you nearly instantaneous access to any sequence from a file, based on the name assigned to the sequence. To simplify the file-reading, assume that sequence names and sequences alternate lines and that the whole sequence is on a single line, as below. Have the program take a sequence name (2nd argument) and print the sequence name followed by the sequence (same format as source file) or print "sequence x not found". Note - though you will only have the program print one sequence, it must be implemented so that ALL the sequences are nearly instantaneously accessible within the program (i.e. don't just read lines until you find the correct sequence name, print, and quit).

red_panda
ATCCGTATA
giant_panda
ATCTGTAAA
raccoon
ATTTGCAAA
(etc)

```
>python seqdict.py raccoon
raccoon
ATTTGCAAA
>python seqdict.py bigfoot
sequence bigfoot not found
```

8. (15 points) Write a program `distance.py` that reads a command-line specified file containing an alignment and prints a raw distance between the two sequences. For simplicity, assume the alignment in the file is on two lines with the format below. Don't use data at positions with a gap in one sequence (these are often interpreted as "uninformative" sites). For distance use number of nt changes divided by the number of informative sites. Remember to make it work with ANY valid alignment.

```
ATTGCTCTGGATCT
ATTCCATCGG-TCT
```

for the above alignment, the result would be:

```
>python distance.py align.txt
0.307692
```

Tip - if your result (for the alignment above) is 0.2857 or 0.3571 you are misinterpreting the question.

9. (challenge question) Write the same program as for question 8, but use one of the standard formats actually used for alignments (shown below) and print the Jukes-Cantor corrected distance. (by the way, you don't need to use the 2 and 15 values - these are useful for programs that need to allocate memory explicitly)

```
2<space>15
name1
ATTGCTCTGGATCT
name2
ATTCCATCGG-TCT
```

(the 2<space>15 is on the first line and means expect 2 sequences and an alignment of length 15; subsequent lines are alternating names and aligned sequence strings (the whole alignment on one line). This is usually called the sequential **phylip** format after the program that first used it, which was written by Joe Felsenstein at UW.)

10. (challenge question) Write the same program as for question 9, except make the output ALL the pairwise distances one per line, indicating which pair by their names.

For example, the following file contents would produce 3 lines of output, one for each pair.

```
3<space>15
name1
ATTGCTCTGGATCT
name2
ATTCCATCGG-TCT
name3
ATGCCATCGGATCT
```

11. (challenge question) Write (or borrow from my answer key) two versions of the programs in Problem 2 of Problem Set 3; one version that reads the whole file first and one version that reads only the necessary lines. Add a timer that starts when the program starts and ends near the program end. Print the elapsed time in milliseconds as a gauge of the relative speeds of the two programs. If you want, have both versions run on a longer sequence and see how the times change. Tip: the simple approach is to use the time() method in the time module, which returns a float value of the current time since some reference time. That is, if you execute time.time() twice the result of the second call minus the first call is the elapsed time (in seconds). Be aware of these possible limitations on accuracy though: "Note that even though the time is always returned as a floating point number, not all systems provide time with a better precision than 1 second. While this function normally returns non-decreasing values, it can return a lower value than a previous call if the system clock has been set back between the two calls." On my Windows desktop the method appears to be accurate to about 1 msec.