Problem Set #4

Due Tuesday Feb 7, at the **beginning** of class. Assignments turned in more than 10 minutes after the beginning of class will be penalized.

Starting with this homework, code with poor style will be penalized even if it produces the correct output. Examples of poor style include ambiguous or misleading variable name choices, redundancy or inefficiency, and lack of comments. Use comments to summarize the logic of your program.

Remember to test your code, both on the provided and additional examples.

The first 2 questions refer to the following alignment:

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

1. (10 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.

2. (5 points) Compute the corrected Jukes-Cantor distance for a raw distance of 7 changes in 100 nt. (use your hand-done answer to be sure that your program in question 3 is correct)

3. (10 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.

4. (15 points) Write a program `seqdict.py` that reads a file of sequences (1st argument) and gives you **rapid** access to any sequence from a file, based on the name assigned to the sequence. 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 rapidly accessible within the program (i.e. don't just read lines until you find the correct sequence name, print, and quit).

```
>cat seqs.txt
red_panda
ATCCGTATA
giant_panda
ATCTGTAAA
raccoon
ATTTGCAAA
dog
CTCTGCACA
```

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

5. (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 usually 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.

```
>cat align.txt
ATTGCTCTGGATCT
ATTCCATCGG-TCT
>python distance.py align.txt
0.307692
```

6. (20 points) Write a program `repeated.py` that reads a command-line specified file and prints all lines that occur multiple times in the file (in no particular order), and how many times each repeated line occurs (separated by a tab).

```
>cat colors.txt
red
blue
green
orange
red
purple
green
green
>python repeated.py colors.txt
green   3
red     2
```

7. (25 points) Write a program kmers.py that reads a command-line specified file containing a single sequence (DNA or protein) like `chr21.txt` and an integer k, and prints all k-mers that occur at least once in the file and the number of occurrences of each k-mer (sequence of length k), separated by a space. Print the k-mers in alphabetical order, and make sure to ignore the case of the letters (don't treat lowercase and uppercase letters separately). Note: chr21.txt is quite large, so this program may take a while to run (it took ~30 seconds on my computer; if it's taking much longer, check if you could make your program more efficient). You may find it useful to create some smaller test files and hand-calculate the expected outputs.

```
>python kmers.py chr21.txt 1
A 10422923
C 7160210
G 7174720
N 3612060
T 10348782
>python kmers.py chr21.txt 2
```

```
AA 3478029
AC 1783676
AG 2417672
AN 1
AT 2743545
CA 2558856
CC 1817796
CG 380444
CN 10
CT 2403104
GA 2080230
GC 1496587
GG 1824254
GN 4
GT 1773645
NA 5
NC 4
NG 4
NN 3612044
NT 2
TA 2305803
TC 2062146
TG 2552346
TN 1
TT 3428486
```

Challenge questions:

1. (challenge question) Write the same program as for question 5, 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.)

2. (challenge question) Write the same program as for question 5, 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
```