CS 412 4th credit project: Due sometime around finals week, exact due date to be announced later.

To be performed in teams of three. (Feel free to use piazza to form teams.)

Submission (by email): A PDF document briefly describing your algorithm, charts with performance evaluation, and your observations from the evaluations.

Demos: ~15 min demos will be scheduled (also around finals week), where each team will explain their report to the instructor and/or TA.

Please consult with the TA if you have any doubts.

The project involves developing a “motif finding” program *and testing it*. The three major components that you have to think about and implement are:

- Building a benchmark
- Implementing the “motif finder”
- Evaluating the motif finder on the benchmark and making intelligent inferences.

Word of advice: Do not wait until the last week to do everything. In the past, some teams have tried this, confident in their programming abilities, but in many cases they failed to do the last step (of performance evaluation) satisfactorily, because this step took more running time than they had budgeted for!

Background knowledge:
1) A ‘motif’ is a pattern in a sequence. For example, in DNA sequences (which are sequences over the alphabet {A,C,G,T}), an example of a motif is the pattern ‘TCACGTG’. A slightly more complex motif is the pattern TC[A/C]CGTG, which represents ‘either TCACGTG or TCCCGTG’. An occurrence of a motif in a given DNA sequence is called a ‘site’.

2) A more popular form of a motif is that of the ‘position weight matrix’ (PWM). This is a probabilistic pattern. An example of a PWM is shown below:

0.7	0.1	0.3	0.1	0.1	0.3	0.1	**A**
0.1	0.7	0.6	0.3	0.1	0.05	0.1	**C**
0.1	0.1	0.05	0.3	0.7	0.05	0.1	**G**
0.1	0.1	0.05	0.3	0.1	0.6	0.7	**T**

This PWM ‘W’ represents a pattern of length 7, and each column prescribes the probability distribution over the four possible characters at each position in the pattern. Thus for example, the first position is very likely (70%) to be an ‘A’ and less likely to be a ‘C’, ‘G’ or ‘T’ (10% each). Any string ‘s’ of length 7 can be assigned a probability score Pr (s | W) that quantifies the probability of sampling ‘s’ from W. For example, the probability score of s = ACCGGTT is:
Pr (s | W) = 0.7 x 0.7 x 0.6 x 0.3 x 0.7 x 0.6 x 0.7.

The ‘information content’ of a PWM ‘W’ of length L is defined by:

$$\sum_{i=1}^{L} \sum_{\alpha \in \{A,C,G,T\}} W_{i\alpha} \log \left(\frac{W_{i\alpha}}{0.25}\right)$$

(Note: if $W_{i\alpha}$ is 0, use $W_{i\alpha}\log(W_{i\alpha}) = 0$ to calculate.)

The information content represents how 'sharp' the pattern is. For example, if every position is uniformly distributed among the 4 characters, the information content is 0. If a position prescribes an 'A' with probability 1 and all other characters are disallowed (probability 0), that position contributes log(4) to the information content, the maximum possible contribution of a single position of the motif.

Step 1: Building a benchmark for motif finding

A benchmark is a collection of synthetic data sets. A synthetic data set is a set of DNA sequences, into which a "motif" (position weight matrix) has been "planted". To construct a synthetic data set, you will do the following:

1) Take as input:
(a) A positive number called ICPC ("information content per column")
(b) A positive integer called ML ("motif length")
(c) A positive integer called SL ("sequence length")
(d) A positive integer called SC ("sequence count")

2) Generate SC random sequences (with uniform frequencies of A,C,G,T). Each random sequence has length SL.

3) Generate a random motif (position weight matrix) of length ML, with total information content being ICPC * ML.

4) Generate SC strings of length ML each, by sampling from this random motif. Each of these strings will be called a 'site'.

5) "Plant" one sampled site at a random location in each random sequence generated in step 2. "Planting" a site means overwriting the substring at that location with the site.

6) Write out the SC sequences into a FASTA format file called "sequences.fa" (Search the web for information on this file format.)

7) In a separate text file (called "sites.txt") write down the location of the planted site in each sequence. (Use any format, your code will be reading this file later.)

8) In a separate text file (called "motif.txt") write down the motif that was generated in step 3. The motif should be stored in the format shown in the following example:

```
>MOTIF1   5
0.5   0.1   0.2   0.2
0.3   0.2   0.4   0.1
0.1   0.1   0.1   0.7
0.8   0.1   0.1   0.0
```

**0.1	0.1	0.1	0.7**
<

(This motif has name "MOTIF1", length 5, and each row after the header is one column of the PWM, with the four numbers representing frequencies of the characters A,C,G,T respectively in positions 1, 2, … 5.)

9) In a separate test file (called "motiflength.txt") write down the motif length.

The four files generated in steps 6-9 are stored in a subdirectory, this subdirectory is a "data set". The numbers ICPC, ML, SL, SC are called the "experiment parameters". Create data sets for different combinations of these parameters as follows:

Let the default parameter combination be "ICPC = 2, ML = 8, SL = 500, SC = 10"

a) set ICPC = 1, 1.5, 2, while other parameters are at default values.
b) Set ML = 6,7,8, while other parameters are at default values.
c) Set SC = 5, 10, 20, while other parameters are at default values.

You may keep SL fixed at 500.

For each parameter combination prescribed in a-c above, generate 10 data sets. Note that each data set will be different from other data sets, even those with the same parameter combination, because the data set generation is stochastic.

So, there are (1+2+2+2) x 10 = 70 data sets in the benchmark.

Write a program that will generate a benchmark.

<u>**Step 2:** Implementing the motif-finder</u>

Write a program that will read the "sequences.fa" file and "motiflength.txt" file in a data set and find a motif (position weight matrix) of length given by the "motiflength.txt" file. How you find the motif (i.e., the algorithm) is entirely up to you. Needless to say, your program should not "look at" motif.txt or sites.txt.
The program should produce two output files for the data set:

1) "predictedmotif.txt", in the same format as "motif.txt" from the data set.
2) "predictedsites.txt", in the same format as "sites.txt" from the data set. This file will have your program's prediction of one site per sequence.

<u>**Step 3:** Evaluating the motif finder on the benchmark</u>

You will write a program/script that will run and evaluate the motif finder on each data set. The evaluation criteria are:

1) Relative Entropy between “motif.txt” and “predictedmotif.txt”. Relative entropy is another name for the Kullback-Leibler (KL) divergence, a measure of difference between two probability distributions.

2) Number of overlapping positions between “sites.txt” and “predictedsites.txt”.

3) Number of overlapping sites (two sites overlap if at least ML/2 of their positions are common) between “sites.txt” and “predictedsites.txt”.

4) Running time.

Show the results of your evaluations graphically using charts. Show how the performance varies with each experimental parameter. For each value of the experimental parameter being varied, compute the average performance over the 10 data sets for that setting of the experimental parameter, and plot the averages and standard errors in a line chart.