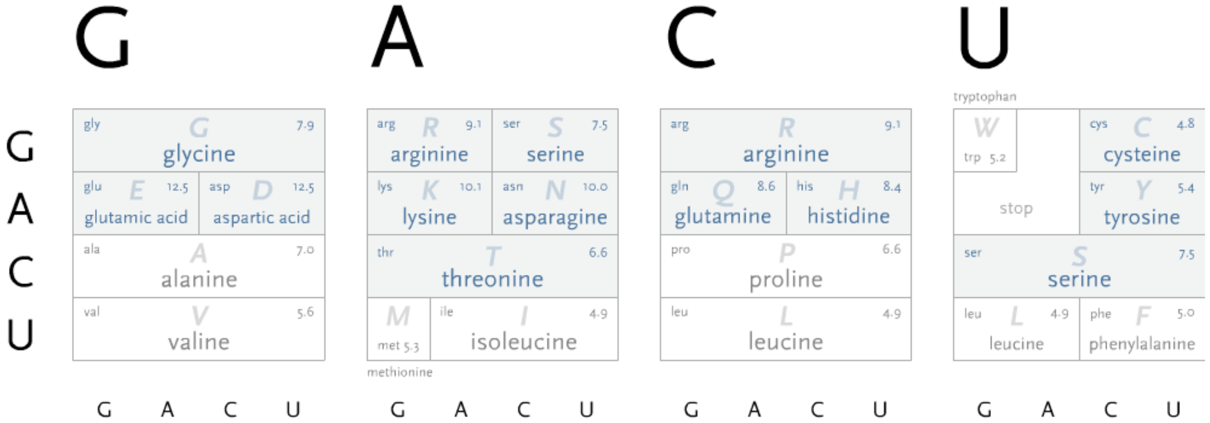
20.180 – Homework Assignment #3

Due on Tuesday, April 25 at 5pm


G
A
C
U
gly G 7.9 glycine
glu E 12.5 glutamic acid
asp D 12.5 aspartic acid
ala A 7.0 alanine
val V 5.6 valine
arg R 9.1 arginine
ser S 7.5 serine
lys K 10.1 lysine
asn N 10.0 asparagine
thr T 6.6 threonine
M met 5.3 methionine
ile I 4.9 isoleucine
arg R 9.1 arginine
gln Q 8.6 glutamine
his H 8.4 histidine
pro P 6.6 proline
leu L 4.9 leucine
tryptophan W trp 5.2
cys C 4.8 cysteine
stop
tyr Y 5.4 tyrosine
ser S 7.5 serine
leu L 4.9 leucine
phe F 5.0 phenylalanine
G A C U
G A C U
G A C U
G A C U

Standard genetic code table via Ben Fry, http://acg.media.mit.edu/people/fry/
Courtesy of Ben Fry. Used with permission.

Create a new Python script called `yourUserName_4.py`. This file should contain all of your code for this problem set, but you should clearly separate the code for the two problems (for example, with a line).

Problem 1: Reverse translation (25 points)

Most amino acids are coded for by more than one triplet codon (e.g., table above). In biological engineering, codon redundancy can allow the redesign of the exact DNA sequence of open reading frames. The goal of problem 1 is explore the redundancy in translational coding by "reverse translating" a protein's amino acid sequence in order to produce a mRNA sequence that would, when translated, yield the protein [please note that reverse translation is not known to occur inside living cells; it's "only" a computational tool]. You can read more about the genetic code here:
http://en.wikipedia.org/wiki/Genetic_code

To get started, read in the files GeneticCode.dict and Protein.txt, and open them to see what they contain. You can assume that the Protein.txt file will contain the amino acid sequence of a single protein (on one line), and that the standard single-letter abbreviations for amino acids will be used. **Your task is to write a computer program that produces the mRNA sequence that would, when translated, encode the protein.** You should use a standard degenerate nucleotide alphabet (below) in order to produce a single mRNA sequence.

Adenine and guanine are both 'purines', which consist of a pyrimidine ring fused to an imidazole ring. Cytosine and uracil are both 'pyrimidines', and contain a single pyrimidine ring. Thus, whenever an amino acid is encoded by more than one codon, you should use the following rules and symbols:

> If the first positions of all codons encoding an amino acid are the same, write that nucleotide's letter to the mRNA string (i.e., A, C, T, G).
> Otherwise:
> If all the nucleotides at the first position are purines (A or G), write the letter "R" to the mRNA.
> If all the nucleotides at the first position are pyrimidines (C or U), write the letter "Y" to the mRNA.
> If all the nucleotides at the first position are either U, C, or A, write the letter "H" to the mRNA.
> In all other cases, write "N" to the mRNA, indicating that any base is possible.
> (Repeat for the second and third position of the codon.)

For example, the amino acid methionine would be reverse translated into "AUG", whereas the amino acid cysteine would be reverse translated into "UGY". The protein "ENDY" should give the mRNA sequence "GARAAYGAYUAY". Use tester sequences like this one to make sure your code is working.

Store your final mRNA sequence (derived from the amino acid sequence in Protein.txt) as a string with the variable name "RT". Your mRNA should be in capital letters.

Problem 2: Signing DNA (50 points)

Engineers whose work impacts the public take responsibility for the safety of their designs. For any engineered DNA fragment that might escape a controlled laboratory setting this could require taking steps to ensure that it is easy to detect the DNA fragment within an uncharacterized environmental sample. One possible way to enable easy detection would be to add a signature tag to the DNA itself (for example, a short sequence of DNA that could be readily recognized by a unique PCR primer). Such a signature tag would be most useful if its sequence was not found or, at least, was maximally different from known natural DNA sequences. More information on these ideas are online here:
http://parts.mit.edu/r/parts/htdocs/barcodes.cgi
http://openwetware.org/wiki/Barcodes

For Problem 2, you will create all possible sequences of 8 bases ("8-mers") and return the one(s) with the lowest sequence homology to a given viral genome. Your code for this problem should be placed below and clearly separated from your code for Problem 1. You should place all functions for this problem above the main code.

Read in the genome of the Bhendi yellow vein mosaic virus (found in the file NC_003418.txt) and store the genome as a string. For more information, go to http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Genome and search for NC_003418.

a. First, write a function called `createMers` that will create all possible 8-mers from the four bases of DNA. This function will take in no arguments and will return all possible 8-mers as a list. Before coding, think about the number of 8-mers you would expect. For example, the list of all 2-mers is
['AA', 'CA', 'GA', 'TA', 'AC', 'CC', 'GC', 'TC', 'AG', 'CG', 'GG', 'TG', 'AT', 'CT', 'GT', 'TT']
Call the function from the main body of your program and store the output to the variable `merList`.

b. Notice that in part (a), you had to hard-code the length of the mer into the function, thus making it impossible to reuse what you have already written to create mers of a different length. You will now write a function called `createMersRec`, which will take in the length of the mer and use **recursion** to create all possible n-mers.

Recursion involves a function calling on itself over and over until it reaches an end state. You can read about recursion at http://en.wikipedia.org/wiki/Recursion. Make sure you understand the example there with factorial before you try to write your own recursive function. In our case, the end state will be when `createMersRec` calls itself with a length of one. As practice, try writing the function to create all 2-mers first.

As in part (a), return all possible n-mers as a list from within your function definition. Call the function from the main body of your program with 8 as the argument to create all possible 8-mers. Store the result to the variable `merListRec`, and note that this list should be equivalent to `merList`.

c. Here, you will go through `merListRec` and find the mers with the lowest overall homology to the viral genome you read in. You should comment out (but not delete) your call to the function `createMers` since there is no reason to create the list of mers twice. When coding, you should create short, practice genomes and short n-mer lists to make sure your code is working the way you would like.

Loop through each of the mers in `merListRec`. For a given mer, slide it along the genome, and score each alignment as you go. Think about how many alignments of that mer to subsequences of the genome there should be. Create a scoring function called `scoreMer` that takes in two arguments. The first argument is the subsequence of the genome that you want to compare with the specific mer from `merListRec`, and the second argument is that mer. For example, if you wanted to align the 2-mer "GT" and the genome "AGTCT", you would call `scoreMer`("AG", "GT"), then `scoreMer`("GT", "GT"), then `scoreMer`("TC", "GT"), then scoreMer(“CT”, “GT”). The score for each alignment is the total number of exact matches. Thus, for the above example, the function scoreMer would return 0, then 2, then 0, then 1.

For a given mer, you should keep track of the worst (‘worst’ meaning highest, or best alignment) score of all of its sliding window alignments with the genome. For the above example, this would be the number 2. Once you have gone through each mer in merListRec, you will need to pick out the best mer(s). The set of best mers (will use the plural, but there may be only one) is the set of mers with the lowest worst score. Store this score in the variable bestScore, and the list of all mers with this score in the list listOfBestMers.

For example, the mer “TT” gives scores [0, 1, 1, 1] for the genome “AGTCT”. The worst score for this mer would be 1. To obtain the set of best mers, you would compare 2 and 1, the worst scores for the two mers in question. The lowest worst score is 1, so you would store the value 1 in the variable bestScore, and the mer “TT” in the listOfBestMers.

You may have noticed that for a merListRec of 8-mers and a ~3,000bp genome, your program may take a long time to run. Shorten your runtime to under 10 minutes by only keeping track of the mer(s) with the best score so far, for example as a variable bestScoreSoFar. In addition, when sliding a given mer along the genome, notice that you can stop sliding once you encounter an alignment that produces a score that is larger than the best score so far. You can use break to break a loop.

Problem 3: Dangerous DNA? (25 points)

Should the DNA sequence information specifying the genomes of human pathogens that are difficult or impossible to obtain from nature (e.g., smallpox, Ebola, 1918 influenza) be freely available via the internet? Please explain your decision but keep your answers shorter than 200 words in length. Please submit your answers via the body of the email containing your answers to Q1 and Q2.

If you would like to consider other's opinions on this question, here are two representative points of view:

a. "Recipe for Destruction," Ray Kurzweil & Bill Joy, NY Times, 17 October 2005
http://www.commondreams.org/views05/1017-23.htm

b. "1918 Flu & Responsible Science," Phillip Sharp, Science, 7 October 2005
http://www.sciencemag.org/cgi/reprint/310/5745/17.pdf