CS223 Fall 2018
Module 1 Week 3 In Class Exercise 1

1. Begin this exercise by forming teams of at least 2 and no more than 3 students. You will work together on this in-class exercise.
2. Approximately 5 to 10 minutes before the end of class, all members of each team will upload one or more text or .py documents with the work completed by the end of class for the exercise. The uploaded document(s) should be exactly the same for all members -AND- contain the names of all 2 or 3 students working together on the exercise.
3. The exercise does not need to be completed by the end of the class, but significant progress, as evidenced by the submission at the end of class, must be evident. Full credit for the exercise will be given after exercise is completed and submitted to Canvas.
    a. If the assignment is successfully completed and submitted during the class, full credit will be attributed to each team member.
    b. If the assignment is not completed during the class, then submit what has been completed for temporary partial credit. Then when it is completed, all 2 or 3 members need to upload the full completed exercise document(s) (i.e., text or .py file(s)) again. NOTE: You will receive a "LATE" submission message in this case, but you will not be penalized if a submission of the incomplete exercise was made before the end of class.

INSTRUCTIONS:

In this in-class exercise, you and your team partners will develop and implement two modules, i.e., two separate .py files that perform genetic algorithm (GA) related tasks. One module should be named "ga_mutation.py" and the second module should be named "ga_crossover.py". At the top of each of these modules, insert the name of all members of the team that worked on the modules. By the end of class, upload both files to the first in-class exercise.

For the ga_mutation.py module:

1. Download the data set named "dna_sequences.dat" from the Module 1 Week 3 Data Set folder and save in your Python working directory. Each row in the file contains a DNA sequence that comes from a different human subject. The entire file makes up what we will call "the population."

2. For the ga_mutation.py module write a function named "mutate" that takes two required arguments and a third optional argument that by default is set to None. The first argument is a DNA sequence (e.g., one row from the "dna_sequences.dat" file). The second argument is a single letter code for the type of mutation to make in the DNA sequence specified in the first argument. The letter codes are as follows:
    a. 'M': Missense Mutation
    b. 'N': Nonsense Mutation
    c. 'I': Insertion Mutation
    d. 'D': Deletion Mutation
    e. 'U': Duplication Mutation
    f. 'R': Repeat Expansion Mutation

Each time the "mutate" function is called, the location within the DNA sequence to mutate must be randomly chosen. For a and c above, a different nucleotide from the one at the randomly chosen location must be selected. For example, if at the selected location for a 'M' type mutation, there is an 'A' nucleotide, then 'A' can be changed to any of 'G', 'T', or 'C' nucleotide. Note for a 'N' type mutation you will need to find a specific sub-sequence of 'AG' first. If there is no such sub-sequence in the provided DNA, then do nothing. If there is only one 'AG' sub-sequence then that will be the one to modify. If there are multiple 'AG' sub-sequences, then randomly select which one to modify.

For 'I' mutations, randomly select a nucleotide to insert and randomly select a location to insert it. For 'D' mutations, randomly select a location to delete the nucleotide.

For 'U' mutations, randomly select a location to make the mutation, and use the third integer optional argument (call it *n*) as the number of nucleotides to duplicate and insert adjacent to the original *n* nucleotides.

For the 'R' mutation, randomly select a location to make the mutation, and use the third integer optional argument (call it *n*) as both the number of nucleotides to duplicate and the number of duplicated to insert. Note, make sure your mutation starts at least *n+1* nucleotides from the end of the original DNA sequence.

After the mutation is made return a tuple that contains (<mutated seq>, location of mutation, letter code for the type of mutation). If for a 'N' type mutation, no mutation was made, then the tuple should be (<original DNA sequence>, None, 'N').

For the ga_crossover.py module:

1. Use the same data set as you used for the ga_mutation.py module.

2. Write a function named "crossover" that takes a single argument which is the DNA sequence (e.g., one row from the "dna_sequences.dat" file). The first step is et generate the complementary nucleotide strand to the provided DNA sequence. Then randomly select a location to start the crossover and randomly select the length of the DNA sequence to crossover. For example, suppose the specified DNA sequence is

   GCGGCCCAGGCCCGGAACCTTCCCTGGTC

   The complementation sequence is

   CGCCGGGTCCGGGCCTTGGAAGGGACCAG

Suppose the location to start the mutation is 4 (where the first nucleotide is at location 0). Further suppose the length of the crossover is 5, then respective sub-sequences of the DNA to cross over are indicated in red and blue below

GCGG**CCCAG**GCCCGGAACCTTCCCTGGTC
CGCC**GGGTC**CGGGCCTTGGAAGGGACCAG

Making the crossover involves switching the indicated (colored) segments between the sequences. This would result in the following

GCGG**GGGTC**GCCCGGAACCTTCCCTGGTC
CGCC**CCCAG**CGGGCCTTGGAAGGGACCAG

After the crossover is made return a tuple that contains (<original sequence with the crossover e.g., GCGG**GGGTC**GCCCGGAACCTTCCCTGGTC>, location of start of the crossover, length of the crossover). Note, make sure the length of the crossover does not extend past the end of the original DNA sequence. If it does, reduce the length of the crossover to end exactly at the end of the DNA sequence.

For the main() "driver" program:

Write a driver main program that reads in each DNA sequence (i.e., each row) of the "dna_sequences.dat" file and calls the mutate function with each DNA sequence, a randomly selected mutation code, and for the 'U' and 'R' mutations options, a randomly selected and appropriate integer number as described above. For each result returned, write the tuple to a file named "mutation_results.dat" .

Then in the main driver program, for each DNA sequence (i.e., each row) of the "dna_sequences.dat" file call the crossover function. For each result returned, write the tuple to a file named "crossover_results.dat" .

UPLOAD BOTH .py MODULE FILES TO THE CANVAS IN-CLASS EXERCISE 1 ASSIGNMENT. DO NOT UPLOAD THE "mutation_results.dat" AND "crossover_results.dat" FILES.