CIS 579 Assignment 2
Summer 2021

In this assignment, you will implement a simple genetic algorithm in the language of your choice with fitness-proportionate selection (roulette-wheel sampling), population size 50, single-point crossover rate $p_c = 0.7$, and bitwise mutation rate $p_m = 0.001$. Try it on the following fitness function: f(x) = number of ones in x, where x is a genome of length 10. Perform 30 runs, and measure the average generation at which the string of all ones is discovered. Perform the same experiment with crossover turned off ($p_c = 0$). If it turns out that mutation with crossover is better than mutation alone, why? You may work with a partner and turn in a joint write-up – please be sure the name and contribution of each lab partner is described in final report.

Your solution should include the following functions or procedures and then use them as building blocks in your main GA program. Feel free to write any other helper functions that you like.

- **randomGenome(*length*)** returns a random genome (bit string) of a given length.
- **makePopulation(*size, length*)** returns a new randomly created population of the specified size, represented as a list of genomes of the specified length.
- **fitness(*genome*)** returns the fitness value of a genome.
- **evaluateFitness(*population*)** returns a pair of values: the average fitness of the population as a whole and the fitness of the best individual in the population.
- **selectPair(*population*)** selects and returns two genomes from the given population using fitness-proportionate selection.
- **crossover(*genome1, genome2*)** returns two new genomes produced by crossing over the given genomes at a random crossover point.
- **mutate(*genome, mutationRate*)** returns a new mutated version of the given genome.
- **runGA(*populationSize, crossoverRate, mutationRate*)** is the main GA procedure, which takes the population size, crossover rate (p_c), and mutation rate (p_m) as parameters. The GA terminates at 30 runs or when which the string of all ones was found. This function should return the generation number when it terminates.

Your GA program should print out the fitness of the best individual in the current population and the average fitness of the population as a whole. A run terminates if you find a string of ten ones "1111111111". Here is an example of the kind of output your program should produce:

```
>>> runGA(50, 0.7, 0.001)

Population size: 50

Genome length: 10

Generation    0: average fitness 5.07, best fitness 5.00

Generation    1: average fitness 5.91, best fitness 5.00
```

```
Generation    2: average fitness 5.45, best fitness 6.00

Generation    3: average fitness 6.02, best fitness 6.00

...

Generation   18: average fitness 8.09, best fitness 9.00

Generation   19: average fitness 8.38, best fitness 10.00
```

You will need to turn in an annotated file listing of your agent code, report, and test run results. You also need to turn in whatever hardcopy you are able to generate documenting the results of your test data runs. Your write-up for this project should follow the experiment write-up temple and in the procedures and discussion include: the behavior intended for your modified eater and the relevant data displays (tables and/or graphs). The discussion of your results should include things that went right, things that went wrong, and lessons learned.