INFAIA01
Practical assignment: Genetic Algorithm

GOAL

In this assignment you must build a genetic algorithm and apply it to a simple problem. The simple problem is the maximization of the function

$$f(x) = -(x^2) + 7x$$

This means that we want to find the value of x in correspondence of which the function $f(x)$ is highest.

We restrict our search only to integer values of x between 0 and 31 (included). You must encode the value of x as a binary number, meaning that the individual of the genetic algorithm will be a **binary string of 5 digits**. For example, the individual 00010 represents the value $x = 2$, the individual 01001 represents the value $x = 9$, and so on.

INPUTS

The user-specified parameters of the program should be:

- *Crossover rate* (value between 0 and 1 indicating the probability of actually carrying out the crossover between parents)
- *Mutation rate* (value between 0 and 1 indicating the probability of carrying out a mutation)
- *Elitism* (Boolean indicating if elitism is used or not in the algorithm)
- *Population size* (integer indicating the amount of individuals in the population)
- *Number of iterations* (integer indicating after how many iterations/generations the algorithm will stop)

ALGORITHM – MAIN LOOP

The main loop of the genetic algorithm is explained in the slides of the course.

Moreover, a C# sample containing only the main loop of a generic genetic algorithm is available and can be used as a starting point. If you use this code, you will have to program by yourself some specific functions to make it work. The functions are:

- Func<Ind> **createIndividual** ==> input is nothing, output is a new individual;
- Func<Ind,double> **computeFitness** ==> input is one individual, output is its fitness;
- Func<Ind[],double[],Func<Tuple<Ind,Ind>>> **selectTwoParents** ==> input is an array of individuals (population) and an array of corresponding fitnesses, output is a function which (without any input) returns a tuple with two individuals (parents);
- Func<Tuple<Ind, Ind>, Tuple<Ind, Ind>> **crossover** ==> input is a tuple with two individuals (parents), output is a tuple with two individuals (offspring/children);
- Func<Ind, double, Ind> **mutation** ==> input is one individual and mutation rate, output is the mutated individual

where **Ind** is the data structure which encodes the individual. You need to define concretely this data structure by yourself.

OUTPUT

At the end of the specified number of iterations, you should print out the following information:

- *Average fitness* of the last population
- *Best fitness* of the last population
- *Best individual* (that is, the individual associated to the best fitness) of the last population