**Genetic algorithms – part 1**

**Before the next classes you should have following parts of projects ready:**

1. Parser function that is able to read the data from \*.tsp files and load it into some python data structure e.g. pandas dataframe, numpy array, your own class object or other
   1. Your data structure should provide easy access to key information for each city – ordering number, x\_location and y\_location
   2. Be careful about datatypes
   3. Test on at least two different files
2. Write a function that return distance between two give cities provided as arguments (in your project they are reflected by pandas row, numpy array row, custom object or other chosen solution. Returned value should be probably of float type (but you can decide otherwise if you have other idea, and you are sure what are you doing).
   1. If you need help at the math here - <https://www.wikihow.com/Find-the-Distance-Between-Two-Points>
3. You should decide how you will be storing the “solution” – i.e. the ordered list of the cities. There are multiple options like string, python list, numpy list, linked list and many others. Chose one that you think will be easy for you to use in further stages of the project. Remember, that your decisions now will affect your workflow later.
4. For chosen way of storing solution create a random one based on file berlin11\_modified.tsp (you can use ready functions like random.shuffle or np.random.permutation)
   1. In your random solution you should have include all the cities from given file and there should not be any repetition – check if everything is okay

**Genetic algorithms – part 2**

**Before the next classes you should have following parts of projects ready:**

1. Write a function that is able to calculate “fitness” of given solution (passed as argument of this function – the type of solution depends on what you decide in point 3). The “fitness” is the total distance between cities for this solution. To calculate fitness use written before function calculating distance between any 2 given cities.
2. Write a “info” function that will be printing on the screen your solution in readable format i.e. 1 5 7 2 10 … with information about it “score” (value of fitness function).
3. Write a greedy algorithm to solve the problem (start in chosen node and then always chose closest city as the next)
   1. Remember you already have a parser (to load the data from tsp files), function to calculate distance between any two cities, also you decide how you store your solution
4. Run greedy algorithm for every possible starting city for berlin11 and print info about them using function written in point 6. Do you get the same score for every starting city? If not find the best starting point and save its score as a reference.
5. Generate 100 random solution for the same problem, calculate fitness for them and also print information about them. Compare scores that you get by random choice with this from greedy algorithm. Save the results, we will be using them later.
6. Repeat 8) and 9) for berlin52 file.

**Genetic algorithms – part 3**

**Before the next classes you should have following parts of projects ready:**

1. In the next steps we will be using the term “population”. Population is a set of solutions. You should decide now how you will be storing populations – once again it can be list, numpy array, pandas dataframe, string, custom object or other methods. We will be dealing with population later a lot, so chose something that you are familiar with and you think will be easy to handle.
2. Write a function that will be returning starting (initial) population. Your function should at least take number of individuals (as we will be calling ‘solutios’) as an argument, but you can include additional parameters.
   1. The simplest version is to just randomly choose all the individuals.
   2. You should also include the possibility of including some of greedy solutions in the starting population
3. Write a function that will be printing information about population passed as an argument – size, **the best score in population** (this mandatory), median, the worst or other (whatever you think is useful)
4. Implement at least one selection function, which for given population will return chosen individual. Test it on your initial population. You can chose between:
   1. Tournament
   2. Elite
   3. Roulette
   4. Other method not mentioned during classes
5. Implement at least one crossover function, which for given parent(s) will return chosen individual. Test it on the parents chosen by selection method from previous point. You can chose between:
   1. Ordered crossover (OC)
   2. Partially matched-crossover (PMX)
   3. Cycle crossover (CX)
   4. Other method not mentioned during classes

**Genetic algorithms – part 4**

**Before the next classes you should have following parts of projects ready:**

1. Implement at least one mutation function, which for given individual will return the same or mutated (with probability of the mutation passed as argument) individual. Test it on the individual created by crossover method from previous point. You can chose between:
   1. Swap (checked for every city in solution independently)
   2. Inversion (checked for whole solution at once)
   3. Other method not mentioned during classes
2. Write a function that will create new epoch (epoch 0 is your starting population, you will create all following by running this function). You can follow pseudocode (but this is not the only way – you can be creative, or search for other options)

**Obraz zawierający tekst, Czcionka, biały, design

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

Basically in this function you should do the following: you create new individuals from previous population by first selecting parents, then doing crossover, and eventually mutating (by random chance). Once you matched assumed number of individuals in the population you are ready to return new population . Remember to evaluate (calculate fitness score) new individuals in the process and to save best of them and whatever else you need for your print population info function. Test the function on your initial population

1. Use the loops to repeat generating new population by function from point 17). Choose some number of epochs and run your code for the berlin11 and berlin52 to check if everything works. You should find the best solution across the epochs. Compare them with results obtained in 9) and 10) and also ideal solution from the provided file. Your results should be better than greedy results. If your results are comparable with random solution, something is not good with your code.

**Genetic algorithms – part 5**

**Before the next classes you should have following parts of projects ready:**

1. To your code add features that will show the results I graphical form (for example best score in the function of the epoch).
2. Prepare code that will enable you to plot charts comparing quality for different initial parameters
3. Start working on your final report. Improve code if necessary
4. Improve parts of your code to enhance the results. Check which files you are able to test (i.e. how efficient is your code).