# Travelling Salesman Problem

# --Using Genetic Algorithms

Section 2 group 204

Team member:

Yunan Shao(001818832)

Yuchen Qiao(001293335)

Vinod Thiagarajan(001237129)

# Abstract

Genetic algorithm (GA) is a met heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

In this project, we will create a genetic algorithm that calculate best path. The path is represented by the order of city, the fitness is the total distance from start city to each other cities once then return back to start point.

# Implementation concepts

1. Seeding: When importing the city list into the original population as individual, the array list that for each individual will be shuffled for randomization.

2. Evolve: Produce next generation by eliminating the second half of population after sorting by fitness. Then mating/breeding to have children by crossover and mutation. And fill the rest space using the selected/survivor pool to reach the maximum population.

3. Culling: select the best half of the population. Individual class implements comparable, compareTo function is used to reverse the order when sorting.

4. Crossover: Select part of the gene from parent1 and fill the rest using missing parts from parent2.

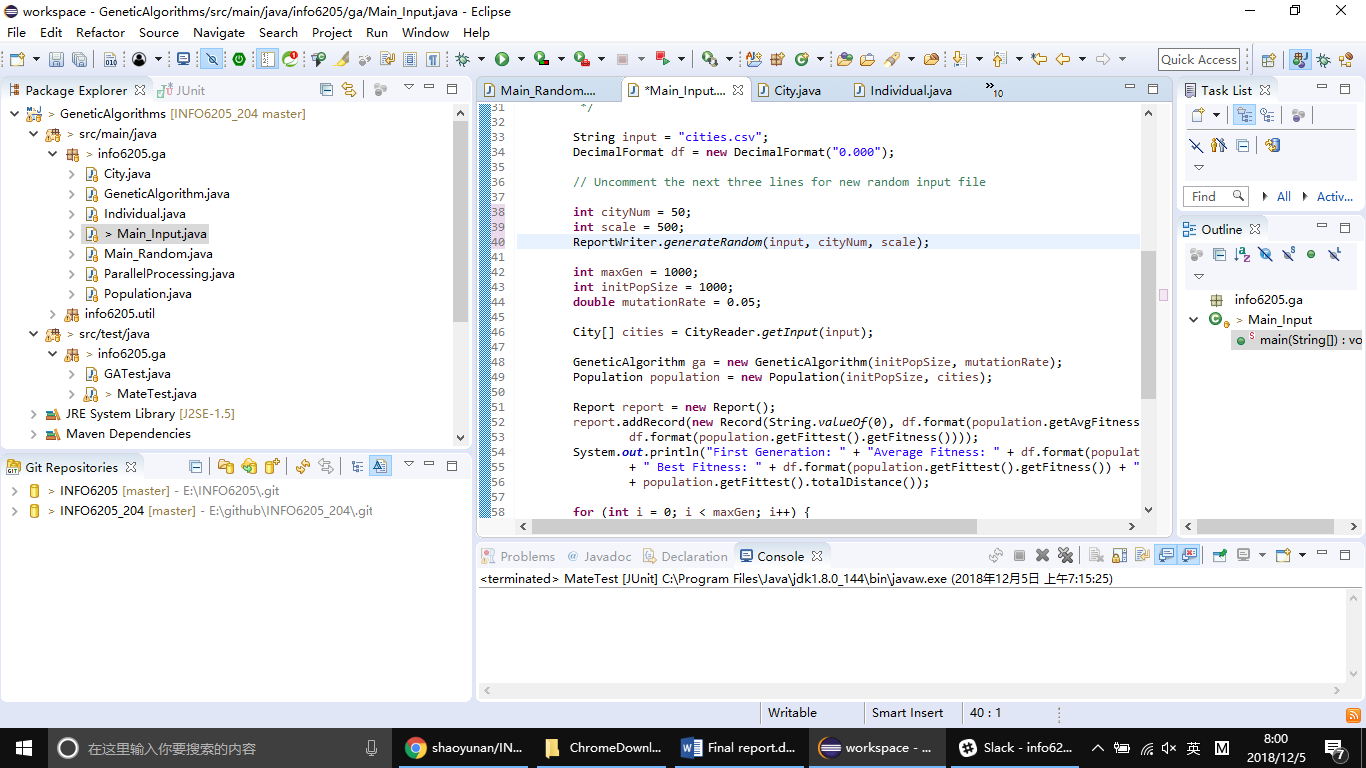
5. Mutate: randomly swap the order of gene (city) for each child, have two implementations. In this project, we have two kinds of mutate.

Mutate: every city has a chance decided by mutation rate to swap position with one of the rest cities

MutateAlt: alternative method, only swap once.

# Parameters Setting

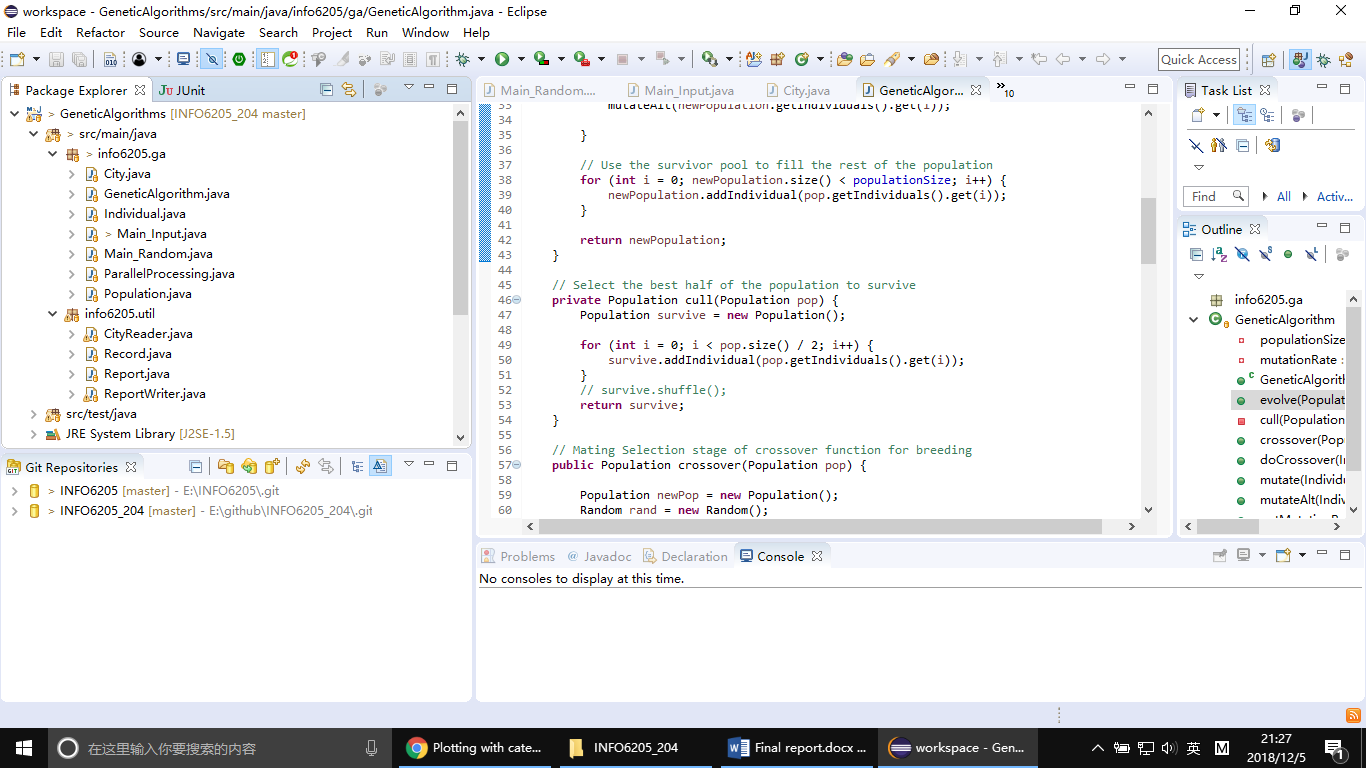
In this project, we set default parameters as follow:



The initial population is 1000 and max generation is 1000. When reach max population, the program will stop and print result.

# Program Structure

Src folder contains ga and util parts.



Ga part is the core of this project, including:

City.java

GeneticAlgorithm.java

Individual.java comments

Main\_Input.java

Main\_Random.java

ParallelProcessing.java

Population.java

Util part mainly read file and output log files, including:

CityReader.java

Record.java

Report.java

ReportWriter.java

# Work flow

Basic step of genetic algorithm

1. The project initializes some cities at the beginning Coordinate of cities is randomly generate to make sure they cover all aspects.

2. Then, the population is assessed by assigning fitness values ​​to each individual in the population. At this stage, we often pay attention to the most suitable solution at present, as well as the average fitness of the population.

3. After the evaluation, the algorithm determines whether the search should be terminated based on the set termination conditions. Usually this is because the algorithm has reached a fixed algebra or an appropriate solution has been found.

4. If the termination condition is not satisfied, the group goes through a selection phase in which individuals from the group are selected based on their health score - the higher the fitness, the greater the chance that the individual will be selected.

5. The next stage is to apply crossover and variation to selected individuals. This stage is where new people are created for the next generation.

6. At this point, the new population will return to the assessment step and start the process again. We call each cycle of this loop a generation.

7. When the termination condition is finally met, the algorithm will jump out of the loop and return its final results.

Here are steps our project works:

First, evolve the population to next generation and create survivor pool, where off springs generate.

In crossover part, we randomly select two individuals as parents. The constraint is that parents can't be the same individual.

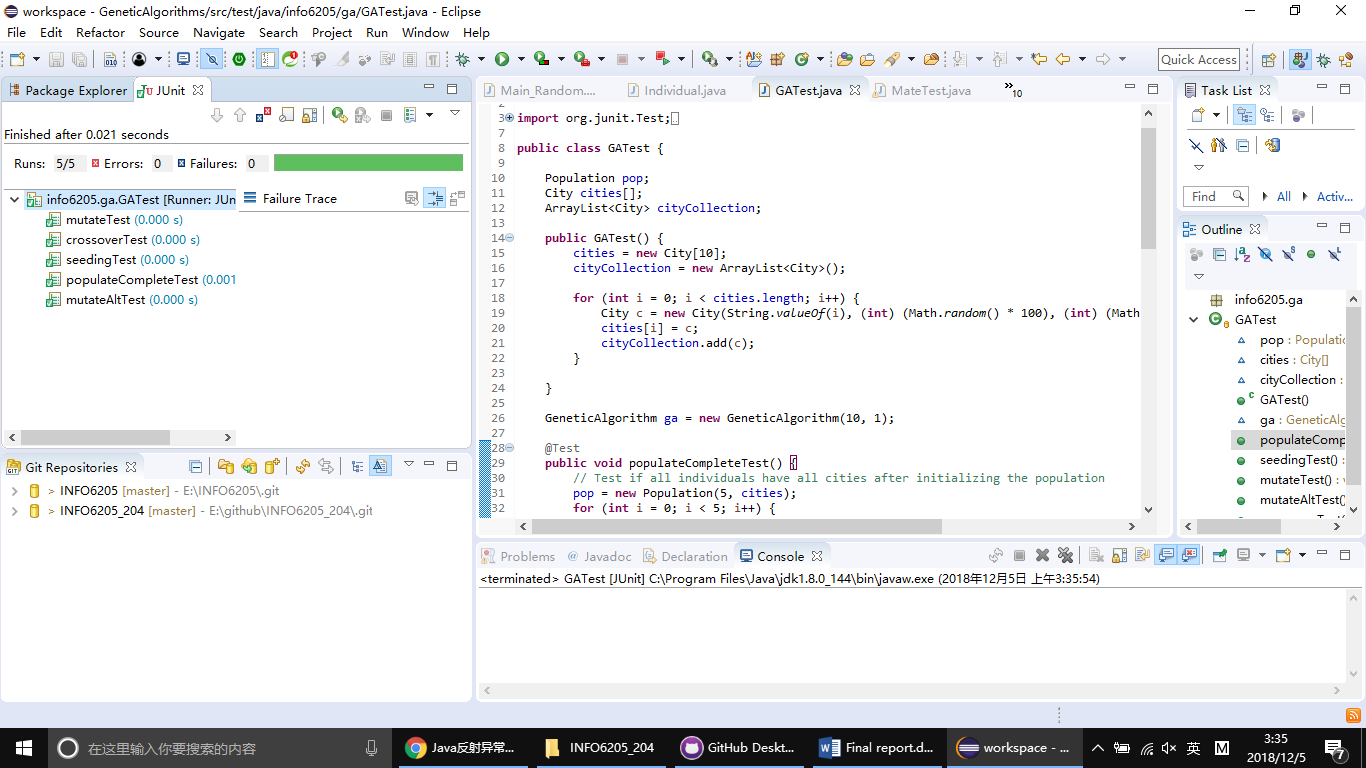
In fitness calculation, we simply use 1/totalDistance of the route as number of fitness.

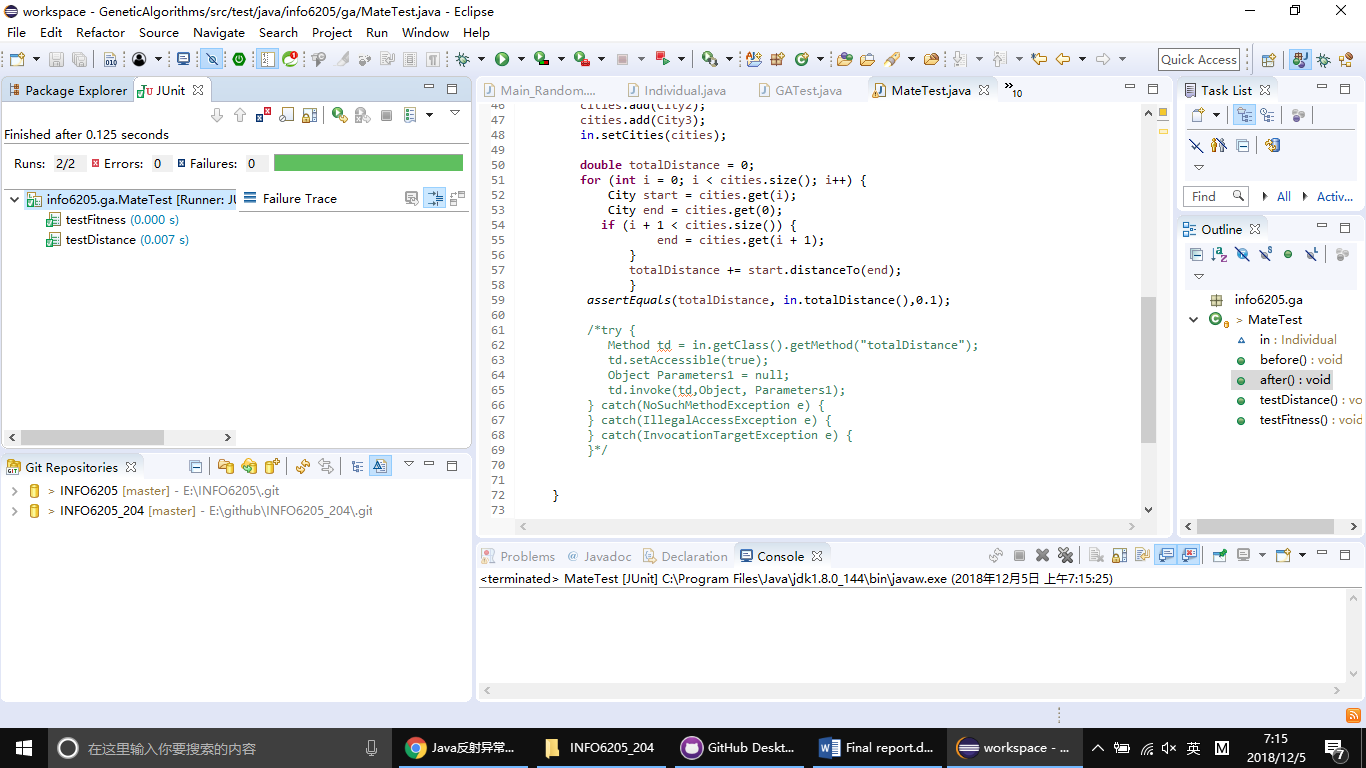
Then, select the best half of the population to survive and use the survivor pool to fill the rest of the population.

After that, we print out the shortest distance as result.

# JUNIT TEST CASES

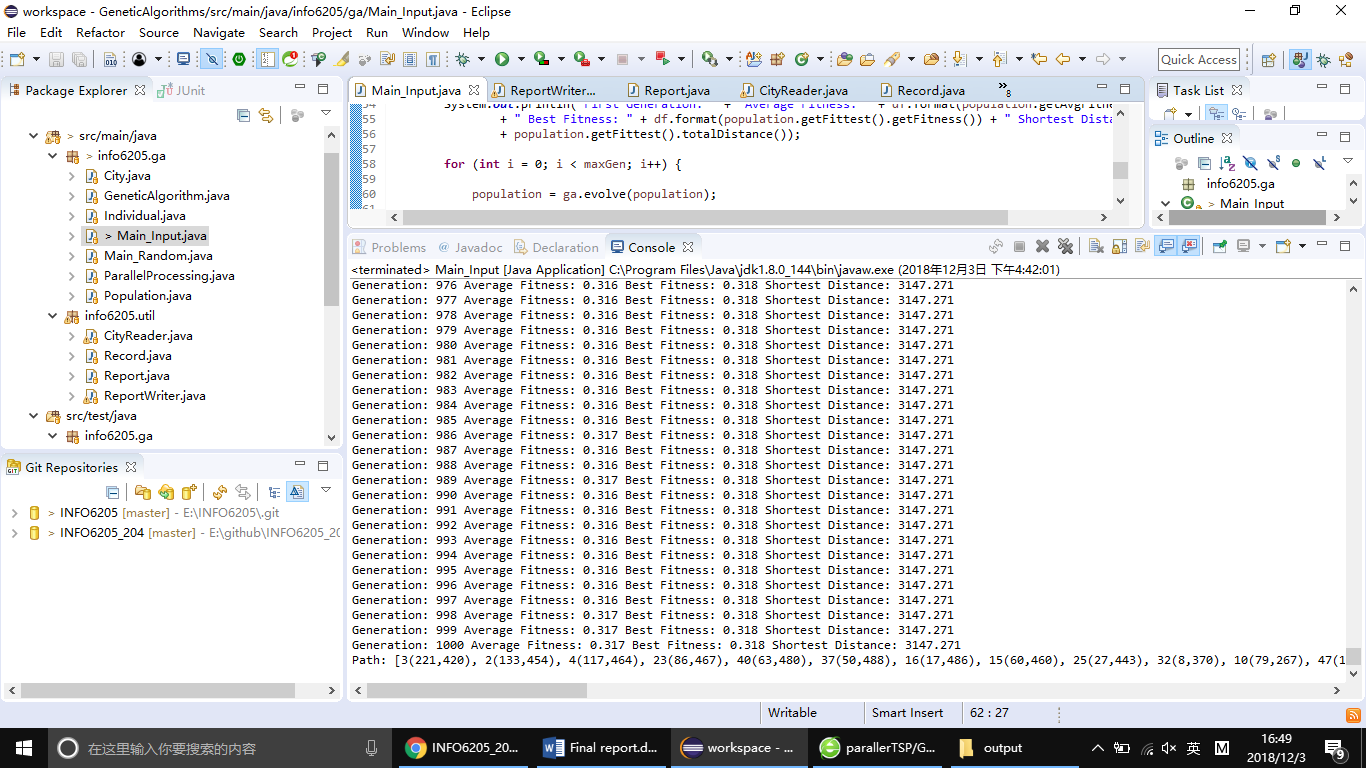
All the test case successfully run.





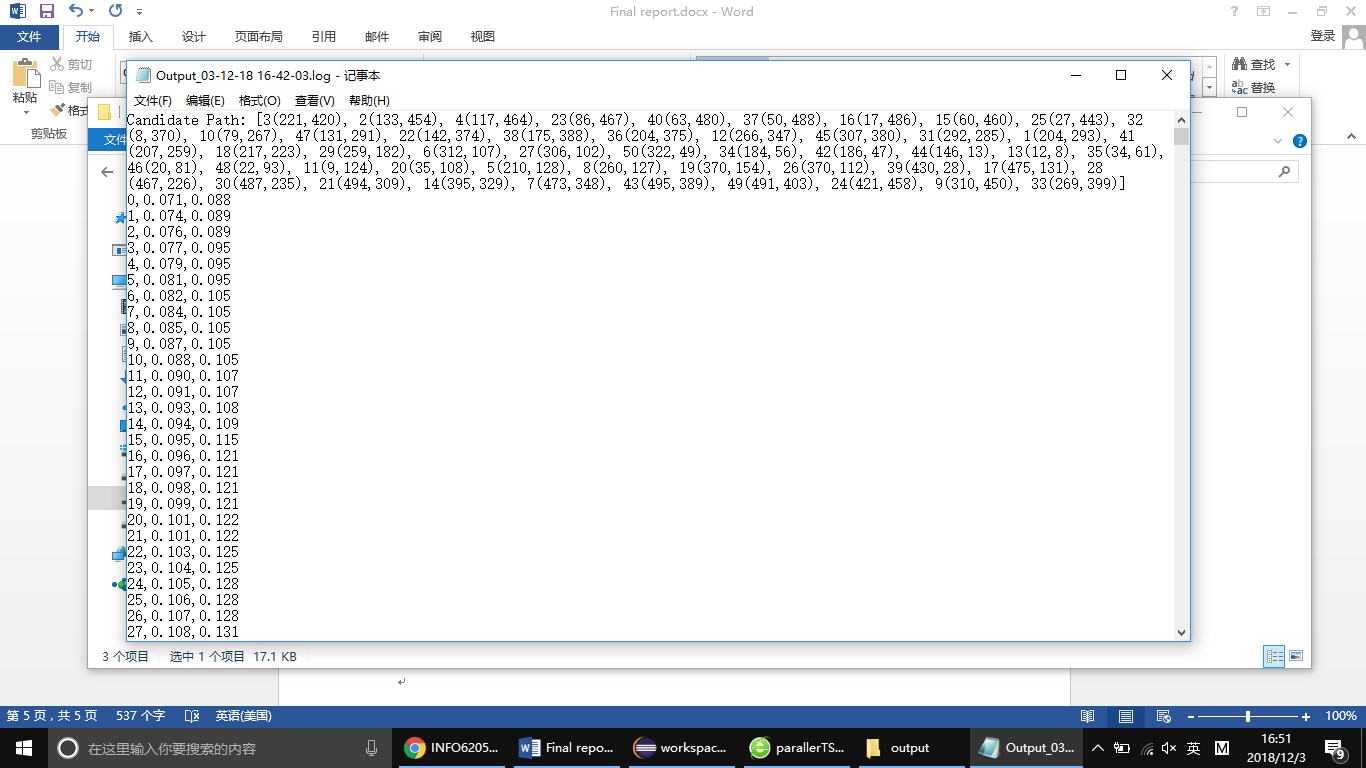
# Running results

Here is the running result for this project:



The shortest distance of each generation will be printed. We sort the distance each time when a new generation is created and the shortest distance is not always improved.

We also generate log file as follow:



# Reference

[1]. <https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications>

[2]. <https://github.com/Apress/genetic-algorithms-in-java-basics/tree/master/GA%20in%20Java>

[3]. <https://www.apress.com/us/book/9781484203293>