Final Project-Genetic Algorithm _ TSP

Chenyang Zhao _ Jin Li

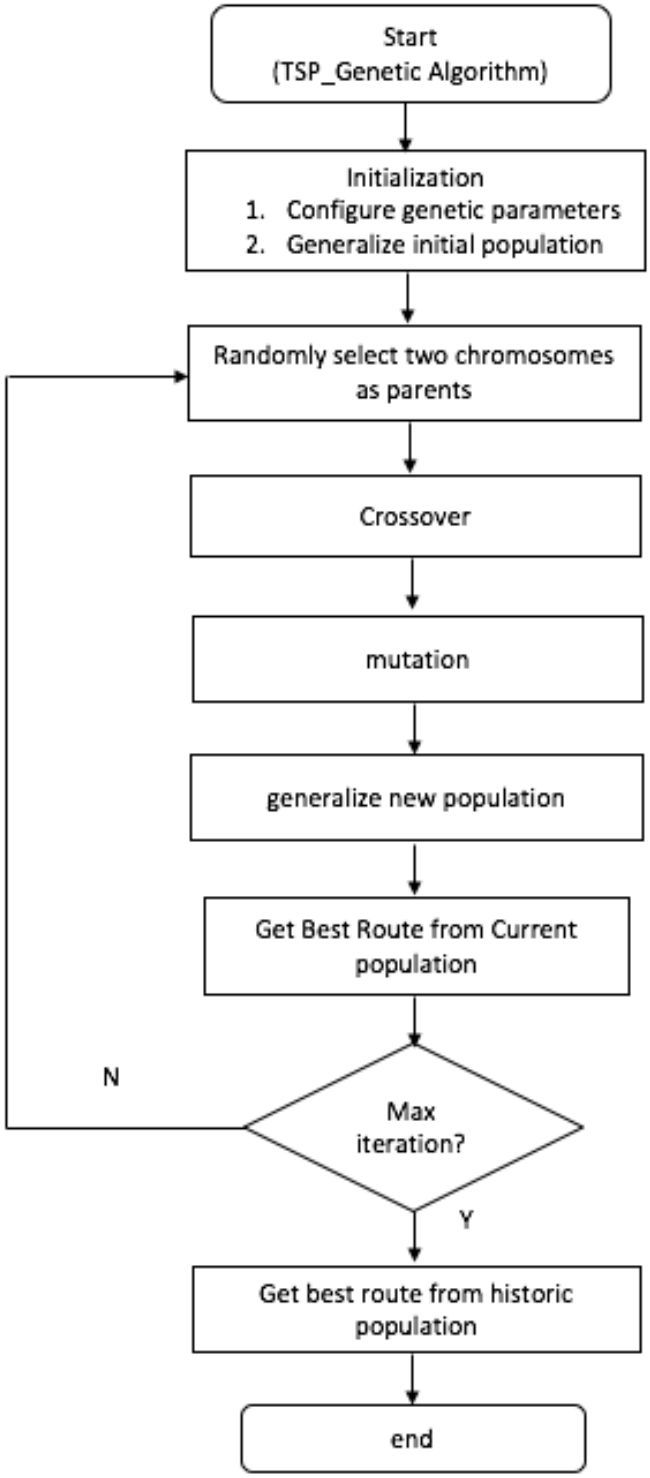
Problem Description

In this project, we solve the travelling salesman problem. We answer the following questions: “Given a list of cities and the coordinates of each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"; “which chief factors can affect the evolve generation and shortest route of travelling salesman problem?” Through creating genes(city), chromosomes, calculating fitness, evolution (crossover and mutation), acquiring shortest distance and best route, we get the results and conclusions. We discover population size, survive rate, mutation rate and crossover rate are related to the best shortest distance and evolve generation.

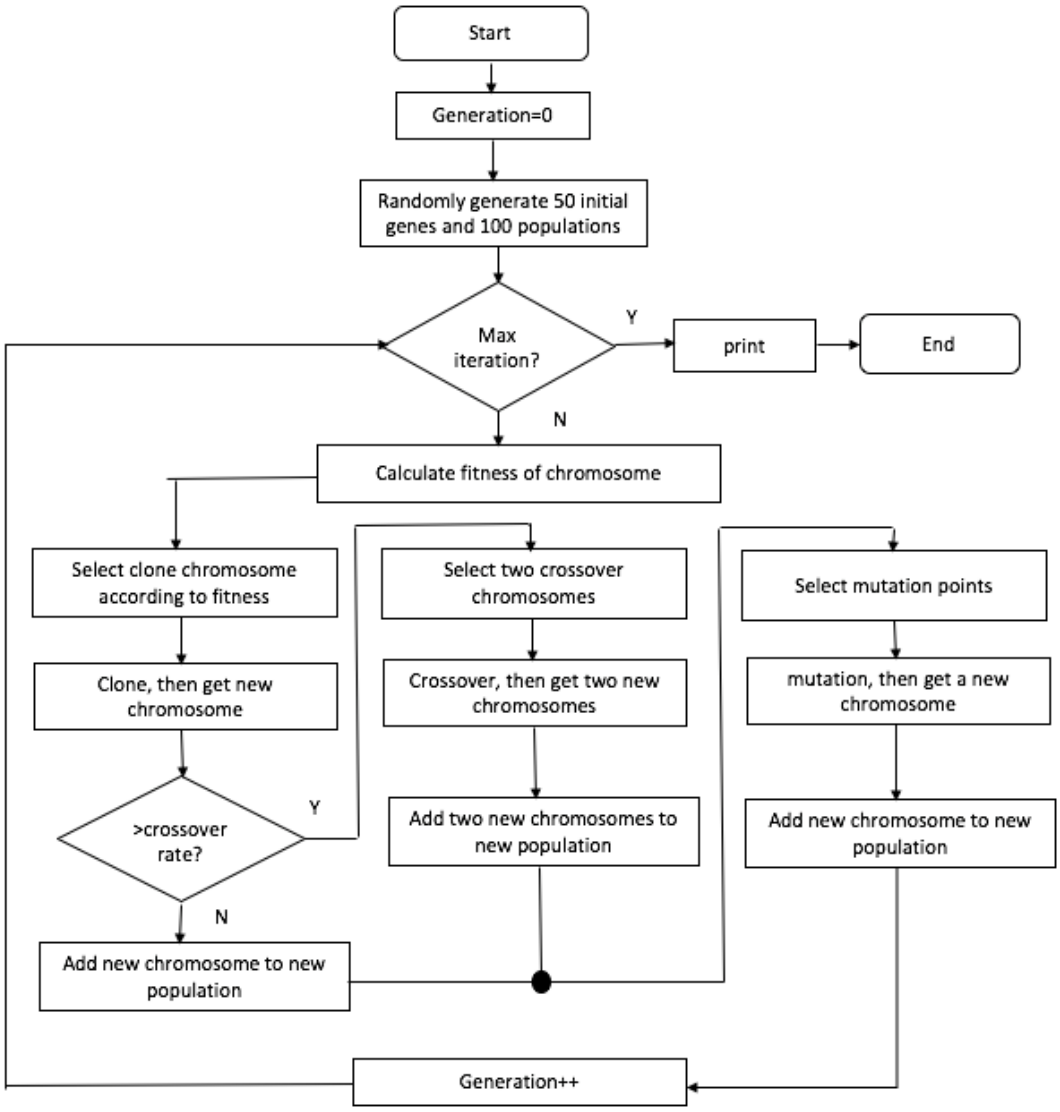
We use 4 classes: Chromosome, gene, population and Genetic Algorithm. In Gene class, we define the id and coordinates of each genes. In Chromosome class, we get the genes(cities) route ArrayList. The route is a list of different genes(cities) and we define each route as a chromosome. For one chromosome or a pair of chromosomes, we do the clone, crossover and mutation. In Population class, we get a chromosomes ArrayList and then evolution. We calculate the distance of each chromosome and acquire the shortest distance of currently generation through priority queue sorting. Finally, we get the shortest distance and best route among every generations. In Genetic Algorithm class, we initialize the parameters, define the terminal condition. When the shortest distance is stable during ‘x’ generation, we end the program and get ‘evolve generation’ which equals to current generation – x. Then, run the logging function and the program.

Workflow Diagrams

1. Overall flow diagram



2. Detail flow diagram



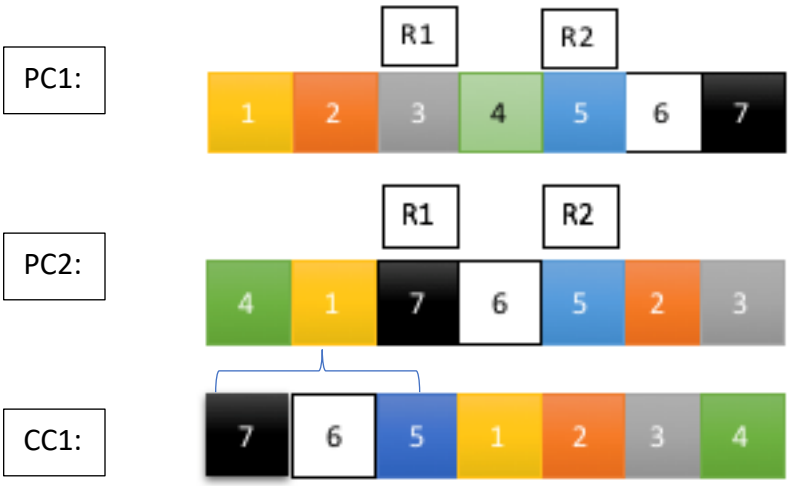
Method:

1. Genes and Chromosomes

   Firstly, we create genotype. Genotype is one gene(city) and its coordinate. We create the object gene and specify the gene size. We randomly generate the different id for each gene from 0 to gene size -1 and different coordinates. Then, we define the expression. The expression is that we map all of the genes to a route. Finally, we get the phenotype. Phenotype is a list of all mapped different genes and it is also a route of all created cities. Furthermore, for phenotype, we traverse all genes and map them into a route. In this route, every gene is different. In phenotype, we get the route ArrayList.

We regard a route as a chromosome which describes the route or the order of different genes.

2. Fitness function
   We calculate the distance between two adjacent genes. Adding all distances together, we can get the distance of current chromosome. We use the distance of chromosome as the fitness function.
3. Sort
   We create a chromosome ArrayList called population. The ArrayList has every chromosome. Using the priority queue sorting algorithm, we sort the chromosomes of current generation by the fitness(distance). According to the survive rate, we get (survive rate* gene size) chromosomes and use them as parents.
4. Evolution
   We randomly select two parent chromosomes and clone them. According to the crossover rate, if the random number is bigger than crossover rate, we add the cloned chromosomes into next generation's population ArrayList. If the random number is smaller than crossover rate, we do the crossover.
5. Crossover
   We use 2 parent chromosomes. For example, the route of parent chromosome 1(pc1) is 1-2-3-4-5-6-7. The route of parent chromosome 2(pc2) is 4-1-7-6-5-2-3. Through crossover, we get 2 children chromosomes (cc1 and cc2). Firstly, we get 2 random number, r1 and r2(r1<r2). In this example, r1 = 2 and r2 = 4. We define flag = r2-r1+1. For cc1, the genes from 0 to flag – 1 is the genes of pc2 from r1 to r2. Then, in pc1, we remove the same genes with the first 'flag' number added genes of cc1. Furthermore, we add the remaining genes to cc1 from flag to gene size -1 orderly. For cc2, the genes from 0 to flag – 1 is the genes of pc1 from r1 to r2. Then, in pc2, we remove the same genes with the first 'flag' number added genes of cc2. Furthermore, we add the remaining genes to cc2 from flag to gene size -1 orderly. In this way, we can get two children chromosomes. The route of cc1 is 7-6-5-1-2-3-4. The route of cc2 is 3-4-5-1-7-6-2.



6. Mutation
   According to the mutation rate, we do the mutation for a chromosome. We randomly select two genes (ids/ cities) and then change their positions.
   Before mutation: 1-2-3-4-5-6-7

   After mutation: 1-5-3-4-2-6-7

7. Get best distance
   After calculating the fitness function, we can get the shortest distance of each generation. Then, we can get the shortest distance among all generations in order to get the historic best route.
8. Logging function
   We use log to document the shortest distance and best route of each generation into a log file.

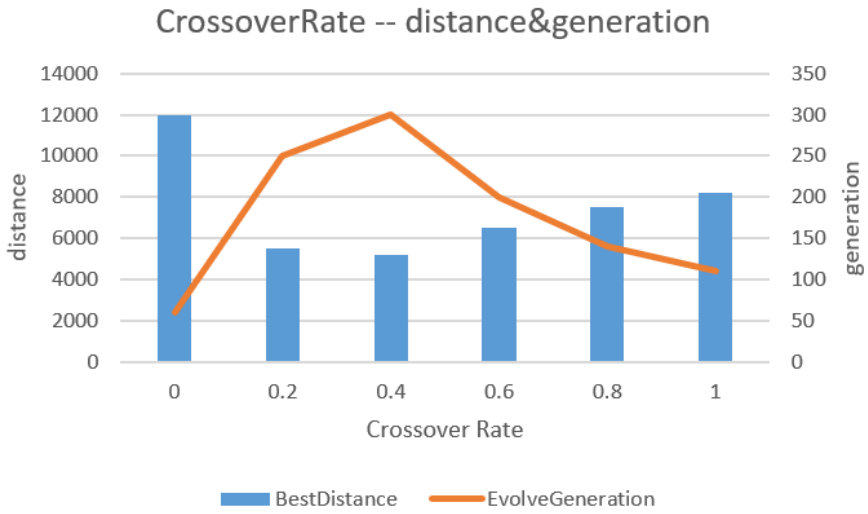
Results & Conclusion

We can get the shortest possible distance and best route of both current generation and all generations on console and in log file. The screenshots of program running, logging file record and unit test run are all included in the bellow appendix.

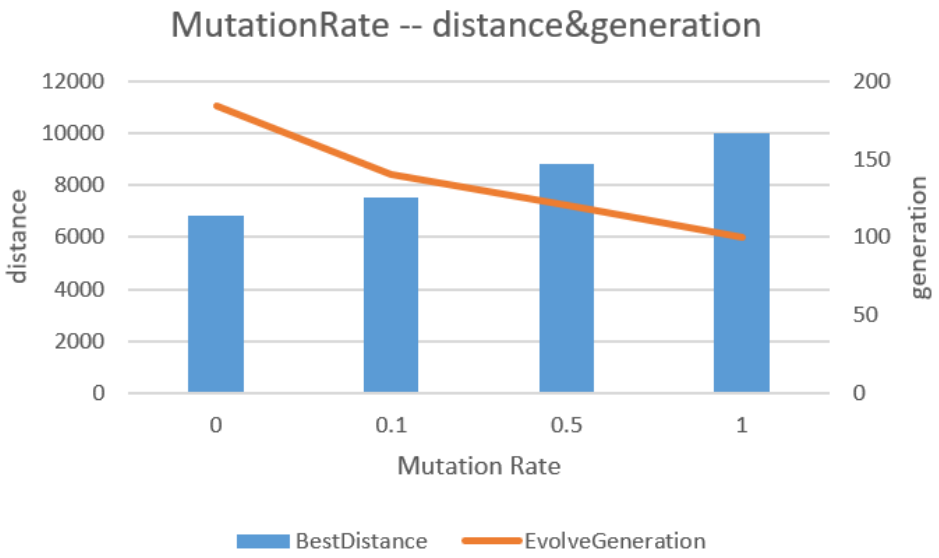
Furthermore, we explore which chief factors can affect the astringency and stability of travelling salesman problem? In other words, which factors can affect the best shortest distance and evolve generation. We discover population size, survive rate, mutation rate and crossover rate are related to the best shortest distance and evolve generation.

Default Configuration: Population Size =500. GeneSize =30. stableGeneration=50. Crossover Rate=0.8. Mutation Rate=0.1. Survive Rate=0.5. For each test, we run the code 5 times, and calculate the average value of best distance and evolution generation.

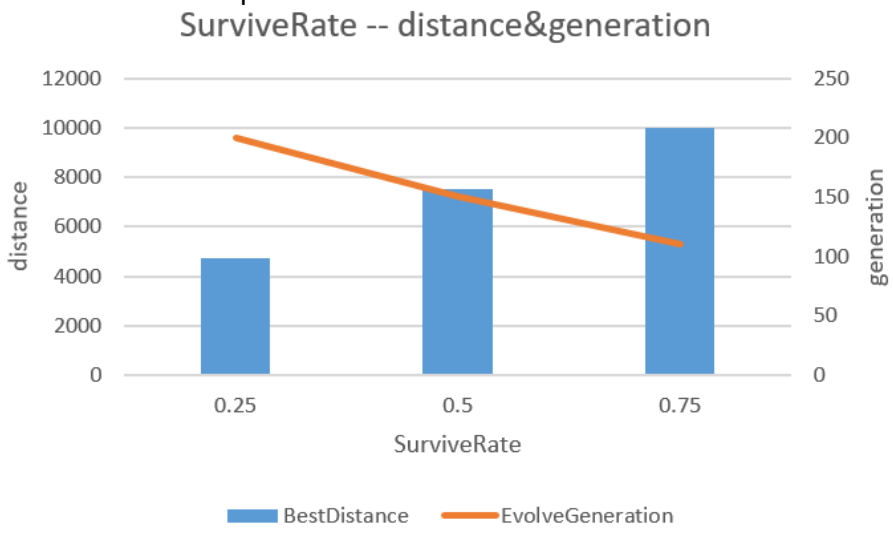
Conclusion1: Population size does not influence the best distance and evolution generation much.



Conclusion2: The best distance decreases when the crossover rate decreases from 1 to 0.4, but it will be stable when crossover rate less than 0.4. When crossover rate is 0, the evolution progress only contains mutation, the best distance will become much longer, and the evolution speed is much faster. Because the default mutation rate is only 0.1, the routes will not change much.



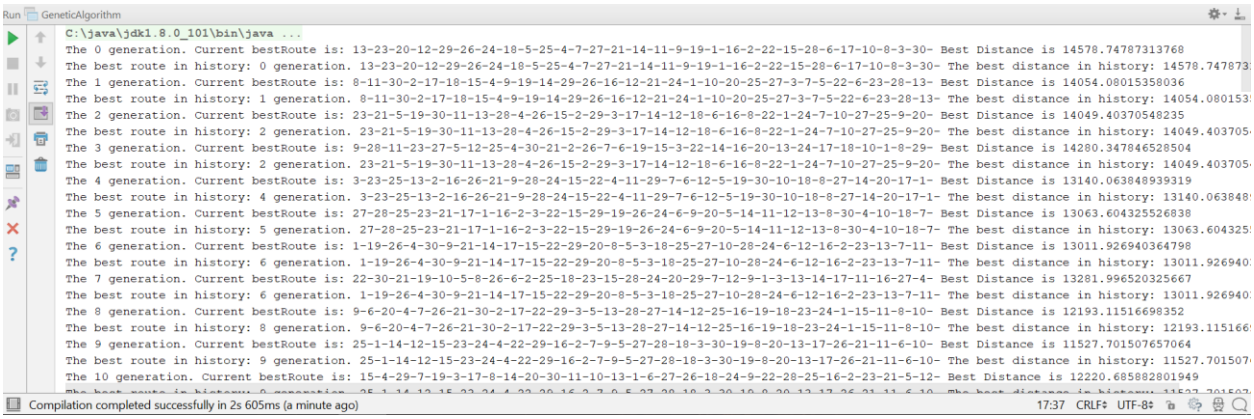
Conclusion3: The larger mutation rate, the faster evolved, the longer best distance. Thus, the crossover without mutation will produce the better result.



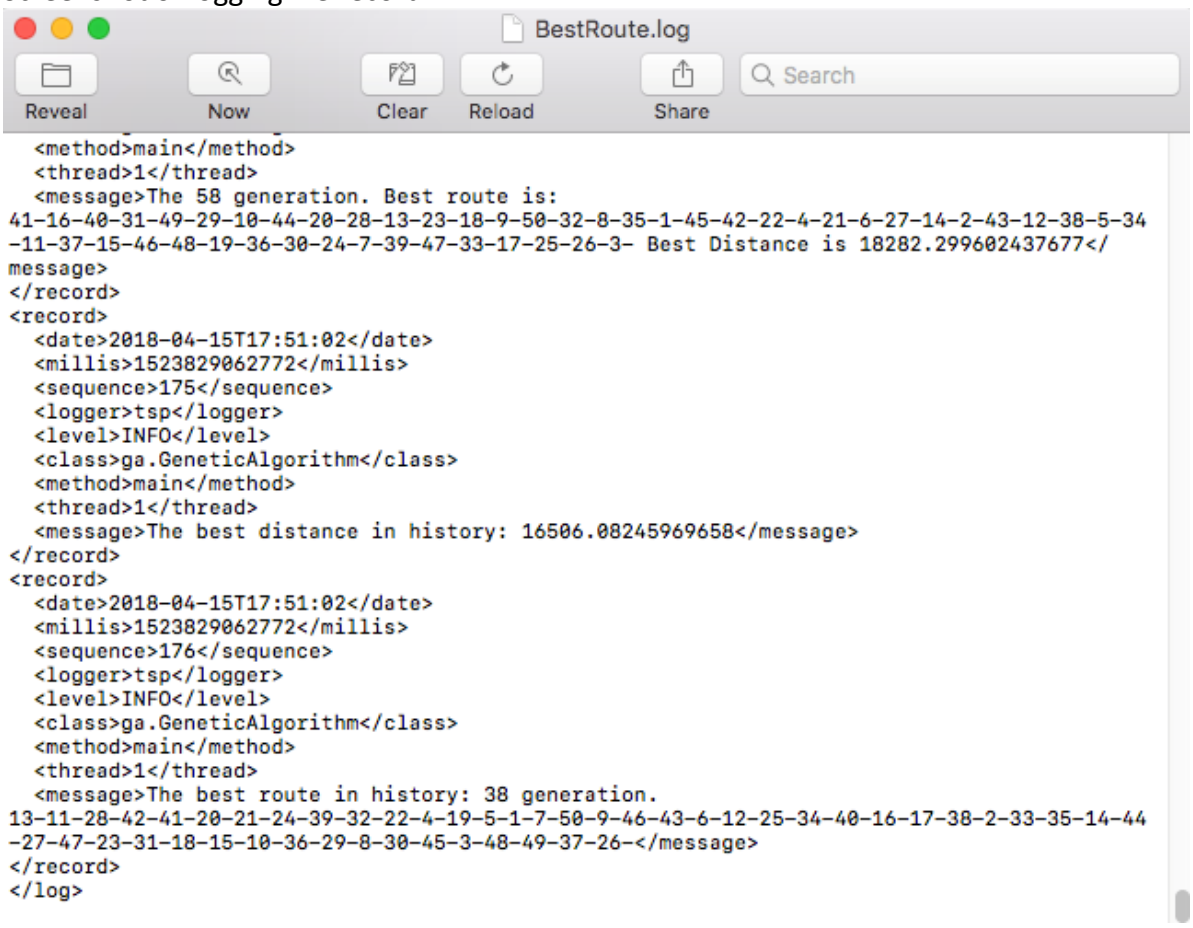
Conclusion4: The little survive rate, the little best distance, and the larger evolution generation.

Appendix:

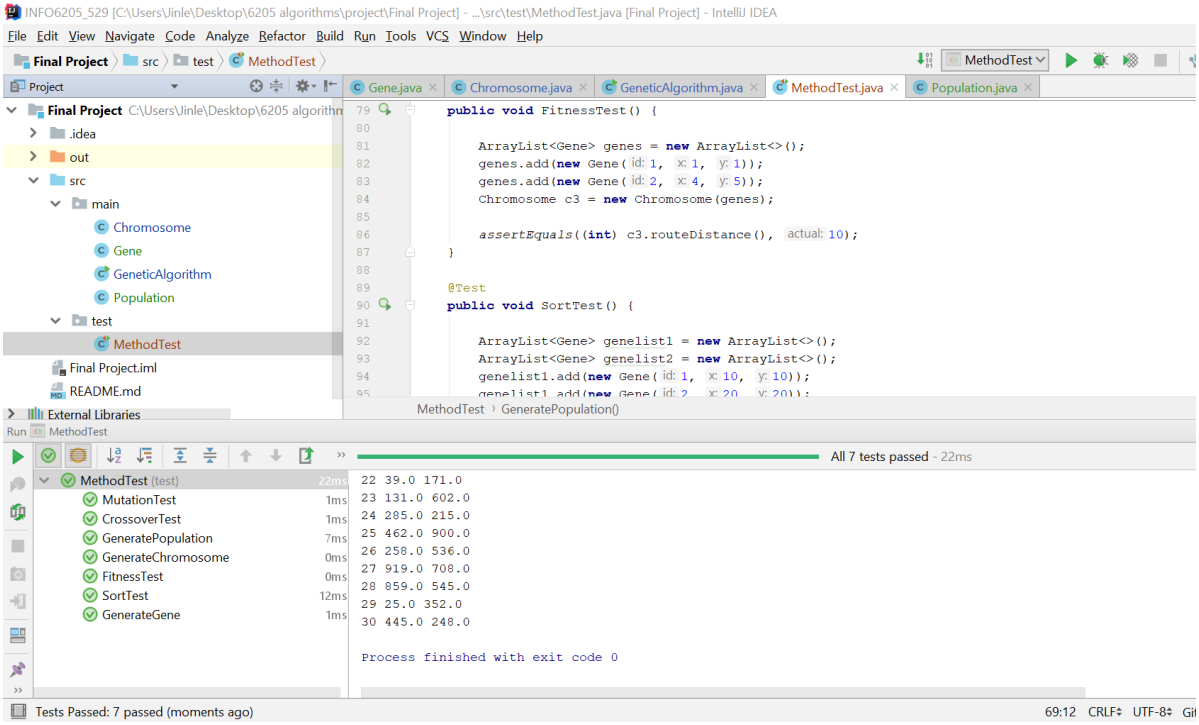
Screenshot of program run:



Screenshot of logging file record:



Screenshot of unit test passed:


INFO6205_529 [C:\Users\Jinle\Desktop\6205 algorithms\project\Final Project] - ...\src\test\MethodTest.java [Final Project] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Final Project src test MethodTest
MethodTest
Project
Gene.java Chromosome.java GeneticAlgorithm.java MethodTest.java Population.java
Final Project C:\Users\Jinle\Desktop\6205 algorithm
.idea
out
src
main
Chromosome
Gene
GeneticAlgorithm
Population
test
MethodTest
Final Project.iml
README.md
External Libraries
public void FitnessTest() {
ArrayList<Gene> genes = new ArrayList<>();
genes.add(new Gene(id: 1, x: 1, y: 1));
genes.add(new Gene(id: 2, x: 4, y: 5));
Chromosome c3 = new Chromosome(genes);
assertEquals((int) c3.routeDistance(), actual: 10);
}
@Test
public void SortTest() {
ArrayList<Gene> genelist1 = new ArrayList<>();
ArrayList<Gene> genelist2 = new ArrayList<>();
genelist1.add(new Gene(id: 1, x: 10, y: 10));
MethodTest › GeneratePopulation()
Run MethodTest
All 7 tests passed - 22ms
MethodTest (test) 22ms
MutationTest 1ms
CrossoverTest 1ms
GeneratePopulation 7ms
GenerateChromosome 0ms
FitnessTest 0ms
SortTest 12ms
GenerateGene 1ms
22 39.0 171.0
23 131.0 602.0
24 285.0 215.0
25 462.0 900.0
26 258.0 536.0
27 919.0 708.0
28 859.0 545.0
29 25.0 352.0
30 445.0 248.0
Process finished with exit code 0
Tests Passed: 7 passed (moments ago)
69:12 CRLF UTF-8 Git