**Project Report**

**Team 522**

**Travelling Salesman Problem**

We developed genetic algorithm for solving Travelling Salesman Problem by implementing 5 classes:

1. City
2. Driver
3. Genotype
4. Population
5. TravelPath

City class mainly focus on defining the Co-ordinates of the cities and names of the cities, the way DNA was translated to real point of locations.

We are initializing genotype length, phenotype length and defining a size of population in the Driver class. This is a main class which is used to run the whole TSP.

In Genotype class we are using two methods, generatePhenotype and compareTo. In generatePhenotype, we are generating the phenotypes based on the size of the genotypes. In compareTo method we are comparing multiple Genotypes to calculate the fitness score.

In Population class, we are initializing the TSP in the constructor. We are using initPopulation method to initialize the 3 values (populationSize, genotypeLength, phenoTypeLength), regenerationAndCulling method to generating the new generation of population by using crossover method.

In TravelPath class we are calculating the distance and fitness score by using the TotalDistance and FitnessScore methods. DistanceBetweenCities is calculating distance between two cities by using distance formula **sqrt((x2-x1)^2 + (y2-y1) ^2)).**

**Implementation:**

Mainly, we are using the Population class with a Constructor, and initPopulation, sortPopulation, regenerationAndCulling and crossover methods. We also have some sub methods like genoTypeComparator and getRandomGenotypeIndex.

• We are Initializing the initial population randomly by InitPopulation method. We are creating genotypes which has a 4-bit string representation of binary number of each city.

• SortPopulation method is used to sort the population based on the fitness score generated for each phenotype.

• Regeneration method is used to generate new population or generation using crossover method by choosing random parents from the population. The population is constant throughout the process.

In the Genotype Class, we are using two different constructors to initialize a phenotype. The class contains generatePhenotype method and compareTo method.

• We are generating the Phenotype by converting the 4-bit binary code into index and get the city from the baseOrder ArrayList.

• We are using CompareTo method to compare two genotypes.

In TravelPath Class, we are having TotalDistance, fitnessScore, distanceBetweenCities methods.

• We calculate the fitnessScore in FitnessScore method based on the total distance calculated in totalDistance method. FitnessScore is obtained by using the follow formula.

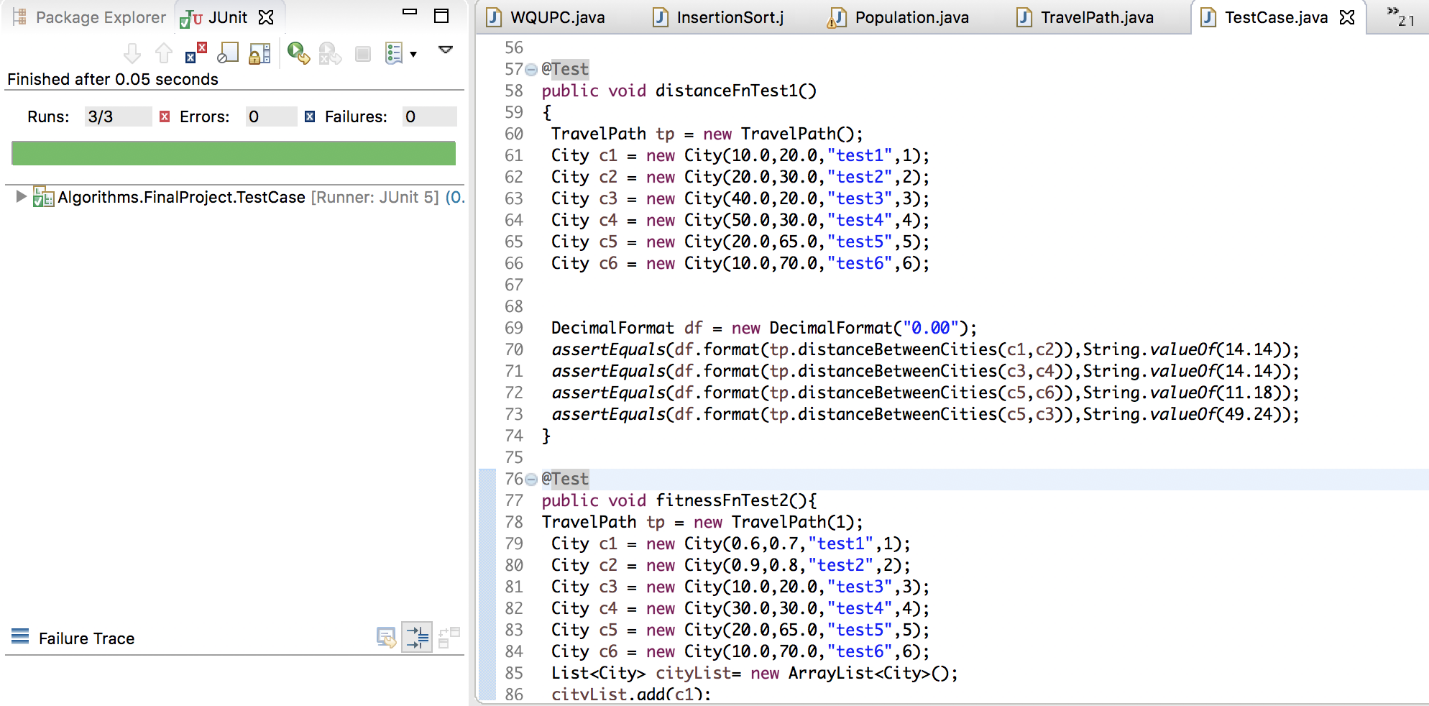
FitnessScore = (1/TotalDistance).

• TotalDistance method is used to calculate Distance between a pair of cities using co-ordinates of the cities.

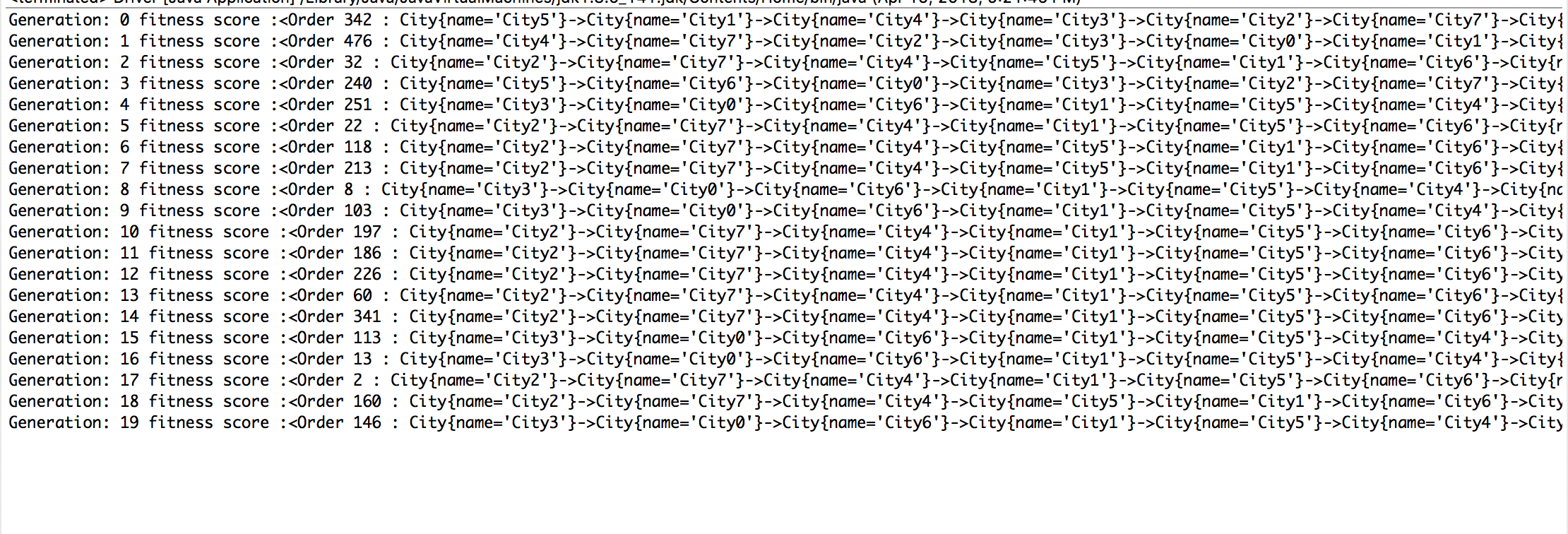
In Driver Class, we have implemented the main function which runs the whole algorithm. Along with the main function GetBaseOrder method is implemented in the driver class.

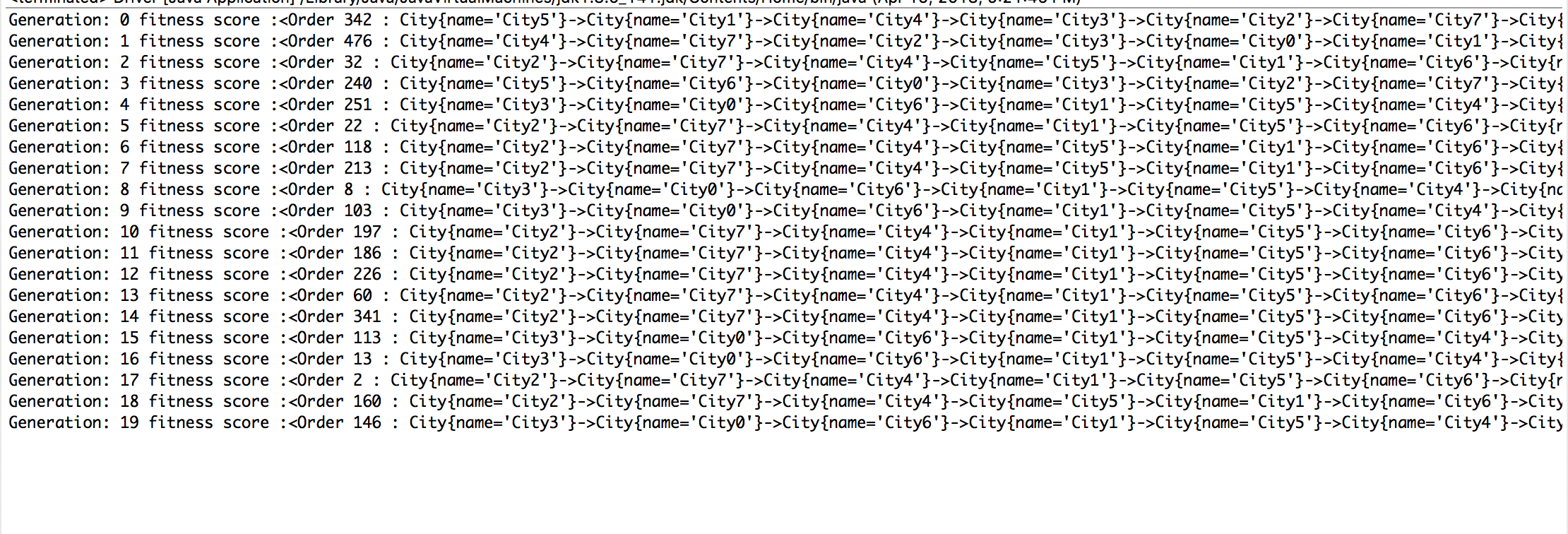
• Array of Random order of cities and along with (X,Y) coordinate is generated using GetBaseOrder.

**Test Cases:**



**Output:**





**Conclusion:**

The whole point of this implementation is to find the shortest path to reach all the cities without repetition by using the Genetic Algorithm. We have developed a method to solve this issue. The result of this method will provide an optimum solution to find the shortest path to reach all cities. According to our tests, the Genetic Algorithm will provide a result which is almost equivalent to the optimal solution. In this problem we have taken only 4 cities where the program generates a result which is close to the optimal solution. It took 10 generations to reach the optimal solution, becomes almost flat (gives the same values) after that generation (According to our logs). If we take more number of cities, our program would need more generations to provide a result close to optimal solution. If we go through our logs, we could see the result trying to reach the optimal solution, but it can not actually reach the optimal solution. After couple of generations, the best DNA will takeover all other DNA’s and remain constant (not changing anymore) and the crossover with other DNAs will not affect much on the result.

So, going back to our original problem, if we want to find the shortest route for our map of different locations we would have to evaluate many different routes! Even with modern computing power this is terribly impractical, and for even bigger problems, it's close to impossible. To find an optimal solution, it would require more effort and other factors.

So, Genetic algorithms appear to find good solutions for the traveling salesman problem, however it depends very much on the way the problem is encoded, and which crossover and mutation methods are used. Overall, it seems that genetic algorithms have proved suitable methods for solving the traveling salesman problem. Yet, genetic algorithms have not found a better solution to the traveling salesman problem than is already known, but many of the already known best solutions have been found by some genetic algorithm method.