

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**CSE2046**

PROJECT 2

Half Traveling Salesmen Problem

08.06.2023

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dept | Student Id | Name Surname |
| 1 | CSE | 150120022 | Tolga Fehmioğlu |
| 2 | CSE | 150121538 | Muhammed Enes Gökdeniz |
| 3 | CSE | 150121002 | Enes Torluoğlu |
| 4 | CSE | 150121032 | Mehmet Toprak Balıkcı |

**Classes And Functions:**

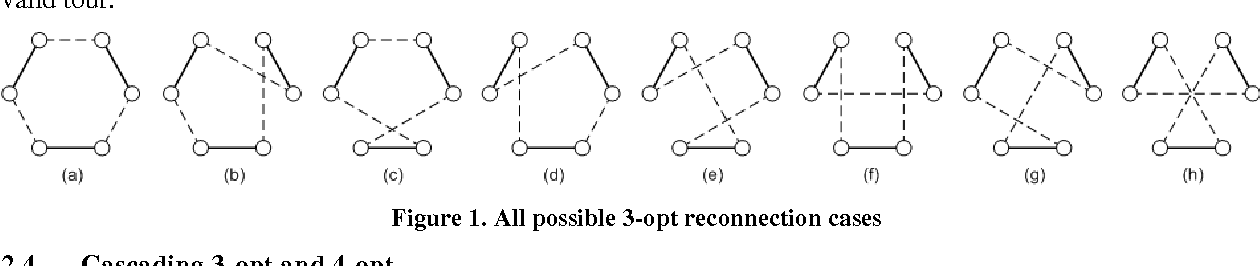
* **HalfTSP**

This is our main class.

* **main(String[] args)**
  + In this function we take input file’s name, then read the cities in the input file and call getTour() function to get initial tour and optimize() function the tour. Then it prints the optimized tour. Also, calculates the time passed.
* **write\_to\_file(int tourDistance, ArrayList<City> initialTour)**
  + This function creates output.txt file and writes the tour to this file.
* **getTourDistance(ArrayList<City> initialTour)**
  + This function calculates the given tour’s distance.
* **getTour(ArrayList<City> cities)**
  + This function returns the initial tour that is found using the nearest neighbor algorithm.
* **getNeighbourCity(City currentCity, ArrayList<City> cities, ArrayList<City> initialTour)**
  + Finds closest city to the given currentCity.
* **getFirstCity(ArrayList<City> cities)**
  + This function finds the first city to start the initial tour.
* **getMiddleCity(int averageX, int averageY, ArrayList<City> cities)**
  + This function finds the arithmetic mean of all cities’ coordinates.
* **getDistance(int averageX, int averageY, City firstCity)**
  + Finds the distance between arithmetic mean and the given city.
* **getAverageY(ArrayList<City> cities)**
  + Finds mean Y coordinate of all cities.
* **getAveragex(ArrayList<City> cities)**
  + Finds mean X coordinate of all cities.
* **readCities(String inputName)**
  + Reads all cities from input file and puts them in an array list.
* **getNumbers(String[] lineElements)**
  + Finds cities’ ids, X and Y coordinates.
* **optimize(ArrayList<City> cities)**
  + This function takes the initial tour and optimizes it using opt3() function.
* **opt3(ArrayList<City> cities, int a, int c, int e)**
  + Finds the most optimized path for the given 6 cities.
* **change(ArrayList<City> cities, int startingIndex, int caseIndex)**
  + Called from opt3 to change the route for a shorter route.
* **distance(City city1, City city2)**
  + Returns distance between two cities.
* **City**

This is the class which we use for cities. We store city id, x, and y coordinates.

**Algorithms**

We used the nearest neighbor algorithm to find the initial tour. To start the nearest neighbor algorithm, we need to choose a city to start from. With some heuristics approach we thought arithmetic mean of all coordinates would be a good position to start the nearest neighbor algorithm. After finding the initial tour we optimized this tour using opt3 algorithm. Starting from the first city we took every 6 cities and used opt3 to optimize the route among these 6 cities. There are 8 possible different routes changing 3 paths. All possible routes are shown in the image below:  


After deciding the best route possible, we change these cities places in the array. And continue this operation till the last 6 cities. The reason behind choosing only 6 continues cities and not random 6 cities is, if the route goes like A to B and there are multiple cities between A and B, and we find a better route where the direction changes and becomes B to A, it would cost a bit to change all the cities between A and B to the other direction so it becomes B to other cities to A which would be costly to implement in Java using Array Lists.

**Performance Analysis**

|  |  |  |  |
| --- | --- | --- | --- |
| Input File | Number of Cities | Distance Using Only Nearest Neighbor Algorithm | Distance After Optimizing with opt3 |
| test-input-1.txt | 280 | 1505 | 1495 |
| test-input-2.txt | 1002 | 141271 | 139046 |
| test-input-3.txt | 33810 | 35227320 | 35002143 |
| test-input-4.txt | 2924 | 5421 | 5354 |

The third input file, 33810 cities, lasts between 15 and 20 minutes while the other 3 inputs are just milliseconds not even a second. Since these values may change for every different computer, here are time complexity for our algorithms: Our nearest neighbor algorithm’s time complexity is O(n^2), and 3-opt algorithm’s time complexity is O(n^3).

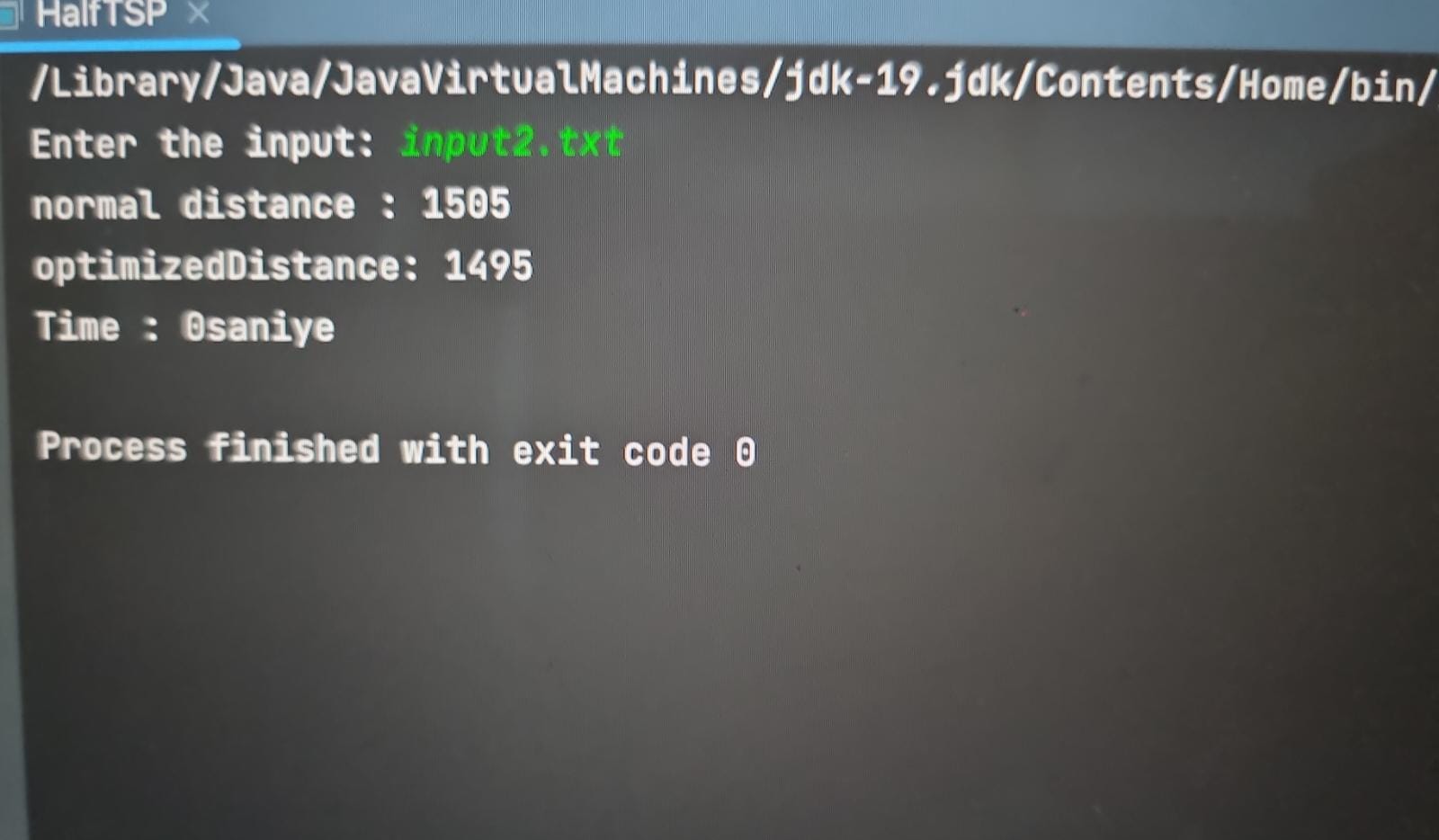
**Division of Labor**

**Tolga Fehmioğlu:** input + neigbor algorithm in general

**Muhammed Enes Gökdeniz :** the decision of applying opt3

**Enes Torluoğlu**: 3 opt algorithm.

**Mehmet Toprak Balıkcı**: Nearest neighbor algorithm+ decision of starting first city.

**HOW THE CODE WORKS**