SWE 510 – Data Structures and Algorithms
Assignment 1 Report – Migros Delivery

Ali Şer Gök
ID: 2022719075
Date: 18.11.2022

Algorithm Explanation

main()

In the main() block, I saved the coordinates of the input01, input02, input03 and input04 inputs and the location names that will enter the permutation() method as an Array. I replaced the index of Migros with the 1st element in both the coordinates and the location names. Because I don't want to sort the number equivalent of Migros in the permutation() method. Migros will be our first and last stop. According to the number entered by the user, I assigned the "inputCoordinates[][]" and "inputLocations[]" arrays, which I prepared separately for each input file, to the permutation() method. Besides that, I made a list for each of them named "inputLocationss<Integer>". Because I need the correct index of each location to make use of the coordinates. Thanks to this list, even though the permutations of the locations are taken, we can find the real old index of the element in the "i" index and find its real coordinates.

permutation()

The array elements are shuffled until "startIndex" and "lastIndex" are synchronized. When the first and last indexes are synced, a unique array is created. If the first and last indexes are not equal, the mixing process continues. The permutation() method is called again. This cycle continues until all possibilities are completed. When a unique Array occurs, we put it in a "for" loop. In order to find the actual coordinates of the locations in each array, we need to access the real indexes from the "ArrayList<Integer> inputLocationss" list. For this, I first assigned the "inputLocationss" list to the "memoryList" list in the permutation() method (when it went inside the main() method to find the index, I got a Heap error every time. That's why I did this).

Then, with the expression " locElement = loc[i]", we find out what the element in each "i" index of the loc[] array is and assign it to an integer value named locElement.

collector()

I'm throwing each totalDistance and unique array (loc[]) that we found in the permutation() method into this collector() method. In this method, I also throw the routeList and tDL lists that I created in the main() method, because I will save each total and distance and unique sequence in these lists. Then I will find the minimum distance from these lists and the shortest path with the help of the index corresponding to this minimum distance.

Program Outputs

The route cannot be shown due to memory overflow but the Algorithm is correct. Route can be calculated only for input04.txt as there is no memory overflow. But if you still want to test the algorithm, you can activate other "case" statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()" method.

Input01 Output

```
1
Shortest route length:	3.8166618881078236

The route cannot be shown due to memory overflow but the Algorithm is correct.
Route can be calculated only for input04.txt as there is no memory overflow.
But if you still want to test the algorithm, you can activate other "case"
statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()"
method.
```

Input02 Output

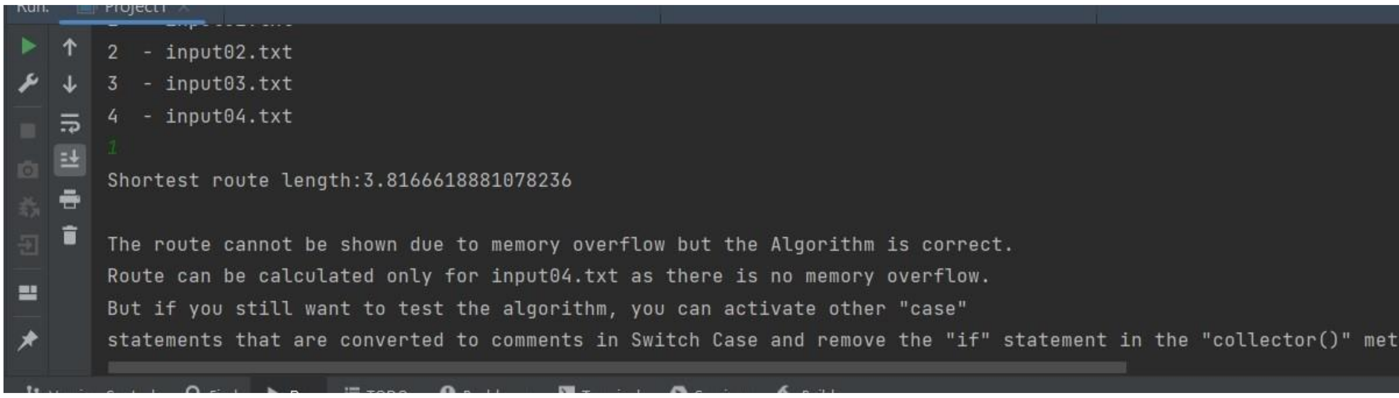
```
2
Shortest route length:	3.682071282486089

The route cannot be shown due to memory overflow but the Algorithm is correct.
Route can be calculated only for input04.txt as there is no memory overflow.
But if you still want to test the algorithm, you can activate other "case"
statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()"
method.
```

Input03 Output

```
3
Shortest route length:	3.022090246134521

The route cannot be shown due to memory overflow but the Algorithm is correct.
Route can be calculated only for input04.txt as there is no memory overflow.
But if you still want to test the algorithm, you can activate other "case"
statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()"
method.
```

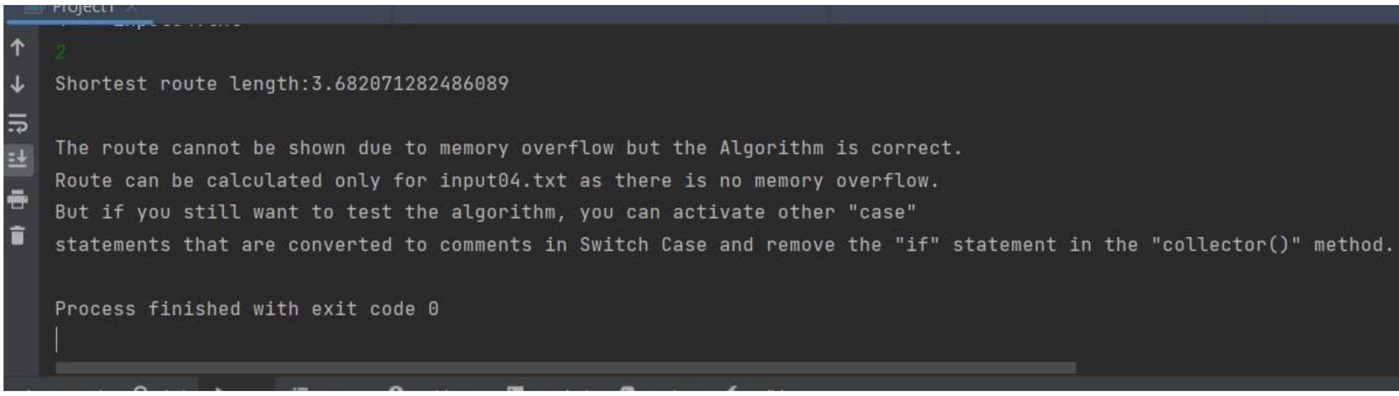
Input04 Output

```
4
Shortest route length:	3.343261097586741

Migros 7 --> 2--> 3--> 4--> 5--> 6--> 1--> 8--> 7 Migros
```



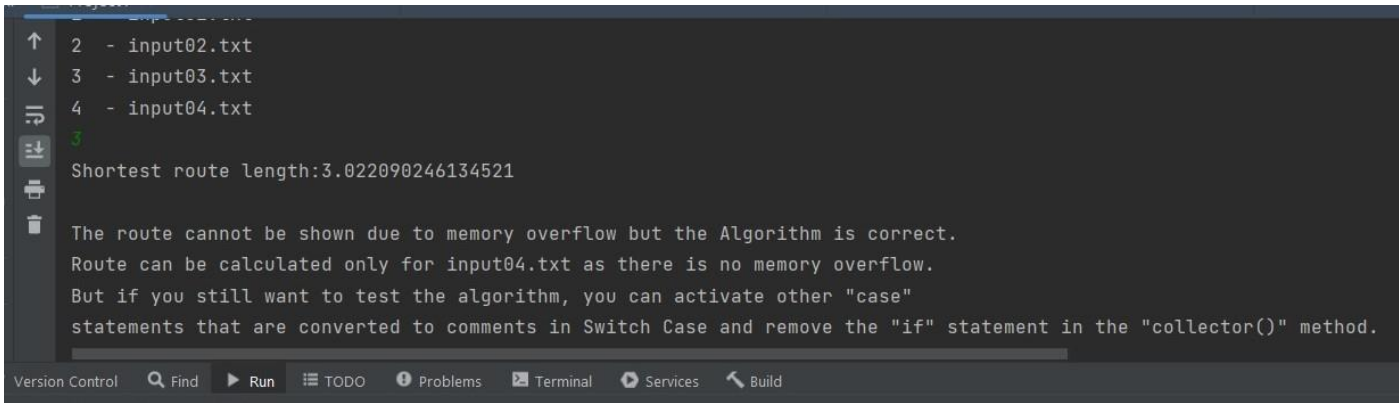
```
2  - input02.txt
3  - input03.txt
4  - input04.txt
1
Shortest route length:3.8166618881078236

The route cannot be shown due to memory overflow but the Algorithm is correct.
Route can be calculated only for input04.txt as there is no memory overflow.
But if you still want to test the algorithm, you can activate other "case"
statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()" met
```

input01



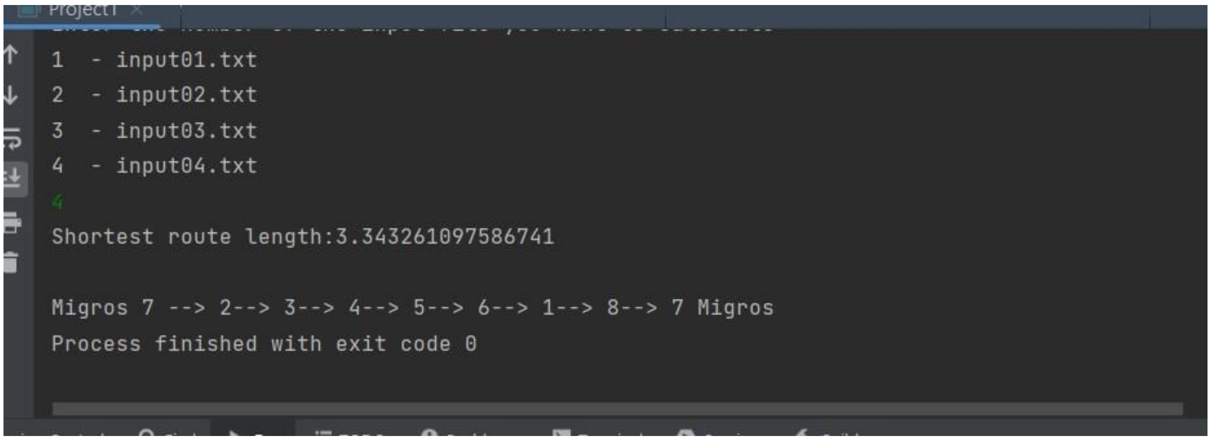
```
2
Shortest route length:3.682071282486089

The route cannot be shown due to memory overflow but the Algorithm is correct.
Route can be calculated only for input04.txt as there is no memory overflow.
But if you still want to test the algorithm, you can activate other "case"
statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()" method.

Process finished with exit code 0
```

Input02



```
2  - input02.txt
3  - input03.txt
4  - input04.txt
3
Shortest route length:3.022090246134521

The route cannot be shown due to memory overflow but the Algorithm is correct.
Route can be calculated only for input04.txt as there is no memory overflow.
But if you still want to test the algorithm, you can activate other "case"
statements that are converted to comments in Switch Case and remove the "if" statement in the "collector()" method.
```

Input03



```
1  - input01.txt
2  - input02.txt
3  - input03.txt
4  - input04.txt
4
Shortest route length:3.343261097586741

Migros 7 --> 2--> 3--> 4--> 5--> 6--> 1--> 8--> 7 Migros
Process finished with exit code 0
```

Note: The average processing time for input01, input02 and input03 is 27 000 milliseconds. input04 gives immediate results.