U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Research applied to the collection of waste in a city

Final Report

Inteligência Artificial - 3rd degree
Mestrado Integrado em Engenharia Informática e Computação

Turma 3 - Grupo A3_2
Artur Ferreira - ei12168 - ei12168@fe.up.pt
Nuno Valente - up200204376 - up200204376@fe.up.pt

May 21, 2017

Contents

1 Objective

This project aims to determine the best route to be performed by a collection of waste trucks in a city, and it has two main objectives: minimize the distance travelled on the route taken and maximize the load waste transported. The last objective relates to minimize the number of waste trucks that are involved.

2 Description

Waste collection is a daily task in a city that must be performed as efficiently as possible, either to keep the city clean or to minimize the associated costs. In order to transport waste to the treatment stations, the city services maintain a fleet of specialized lorries which carry out collection routes, that are defined previously and carried out systematically at a given frequency.

It is intended to perform such collection more intelligently. In fact, containers scattered in various parts of the city, where the residents deposit the garbage. These containers may not be full enough to justify emptying them by the collection truck, which would make some trips unnecessary. With the technology of sensor networks developing rapidly, more effective monitoring of the level of waste accumulation in each container is already possible.

We have considered the existence of 4 types of waste: paper, plastic, glass and ordinary trash. Each truck carries only one type of waste, because we must think in recycling.

In this work, we intend to develop an application that determines the collection routes to be made by trucks, considering only the containers with sufficient residue that justifies their collection. This application should be able to suggest the best route from the central, where the trucks are stationed, to the treatment stations, where all the collected waste is deposited.

As a first step, we have considered that the collection is carried out by a single truck of limited capacity. In a second phase, we'll consider several trucks with limited capacity and when trying to optimize the route, we want to use as few trucks as possible.

3 Specification

3.1 Important concepts

In this problem we need to consider a few concepts like truck, container, place of departure, the final place and the desired route. More properly:

- The specialized truck has a limited capacity, a type of waste and a fuel diesel tank;
- The place of departure is the central where are the trucks to initialize their route;
- The final place is where, at the same time all the trucks have collected all the waste and leave their garbage in the treatment stations;
- One container is consider as a set of four individual type waste;
- The desired route is the itinerary that we're trying to determine considering the objective already mentioned.

3.2 Problem description

In a summarized way, we need to determine the best itinerary that contemplates the already referred objectives in section 1. The next subsections gather additional information necessary for the specification.

3.3 Problem restrictions

In order to make the problem more realistic, we had the intention in use real latitude and longitude coordinates of some streets of Porto where we put the containers, as we said we would do in the previous report. But, to make the problem easier to debug we use fictitious distances stored in the nodes of the graph.

Some restrictions were imposed in this problem:

- We determine the cost associated to the utilization of each truck, taking only into account the diesel fuel spent on average by a normal lorry.
- We assume a truck with infinity fuel but with a limited capacity with in their own container
- Doesn´t exist the agent truck driver, just only the truck
- Some limit values applied to some variables:

  Garbage container capacity(kg) $= 100$;

  Truck capacity(kg) $\in [0, 3000]$;

  Number of each type of truck $\in [1, 10]$;

  Minimum level of waste in a garbage container(kg) $\in [50, 100]$.

3.4 Problem representation

To represent the map of this problem, we considered a undirected-weighted graph with a list of nodes. Each node has their adjacent edges and it is used to represent, in general, the garbage container map. More generally, a node represents a point of passage: a garbage container in some street, a treatment station, the central and a desactivated garbage container.

Although it is possible to consult the entire project code in Annex A.7, here we present the fields that appear in each of the structures necessary to represent the graph:

In sense of mathematical settlement is represented by an unoriented graph G = (V, E), where V is the set of vertices and E – a set of edges of the graph. The vertices of the graph correspond to garbage containers, central or stations. The edges of the graph correspond to the settlement streets that connect the vertices.

The weights are assigned to the edge. They correspond to a total length of street, which is represented by that edge. The length of each blind street is counted twice because the rubbish disposal van has to pass it twice. The van have to pass once other streets, taking garbage, but it can pass them any number of times passing to other streets. Such transfers have been called empty journeys. As mentioned above, the optimal solution is one in which the sum made empty journeys and journeys to landfill is minimal, because the length of the path travelled by the garbage truck affect on transport costs and pollution of exhaust gases [11].

- Graph:

```
private ArrayList<Node> nodes;
private double wastePaper=0.0,
               wastePlastic=0.0,
               wasteGlass=0.0,
               wasteCommon=0.0;
private boolean original;
private HashMap<Integer, Node> containers;
```

- Node:

```
private double distance;
private String type;
private ArrayList<Edge> outEdges;
private String name;
private Node parent;
private Map<String, Double> garbageContainer;
private static int current_id = 0;
private int id;
```

- Edge:

```
private Node source;
private Node destiny;
private double distance;
```

3.5 List of requirements

In list of requirements above, in comparation with the same list in the previous report, we add a column with the field check to show what we had proposed to do when writing the previous report and what we actually did. We can obvious observe that everythinh was implemented, the mandatory and the opcional tasks.

Check	**Id**	**Priority**	**Description**
✓	R1	Mandatory	The user can chose the number of each type of truck available on the central
✓	R2	Mandatory	The user can enter the truck capacity
✓	R3	Mandatory	The user can select the number of stations to leave the garbage
✓	R4	Mandatory	The user will see the result of the implemented search algorithm in console
✓	R5	Mandatory	The application must provide the result with the data that the user chose to test
✓	R6	Opcional	The user will see the result of the implemented search algorithm in a graphical friendly user interface.
✓	R7	Opcional	Nodes and edges are loaded from a csv file to facilitate the edition of data
✓	R8	Opcional	The user might chose other algorithms to find the best itinerary

3.6 Solution

In order to finding the best solution to the problem, we have applied the algorithm A^* to a object, namely AStarNode that represents some kind of photo, that is, the actual state on the seaarch algorithm. This algorithm figures the least cost path, starting their journey at the Truck Center - start state - and ending whenever all the trash is collected and deposited at a treatment station - the goal state. To chose the best AStarNode to A^* algorithm uses a modified evaluation function, the f function and a best-first search. The evaluation function f is an estimate of the value of a AStarNode x given by the following formula:

$$f(x) = g(x) + h(x) \tag{1}$$

where $g(x)$ is the cost to get from the start state to state x and $h(x)$ is the estimated cost to get from state x to the goal state.

3.6.1 The function $g(x)$

In the equation refered in (1), the $g(x)$ represents the cost to reach the current position starting from the initial. To determine the cost we have, initially, calculate the straight line distance between to nodes but, as we already referred, to debug it was more easy to use integer distances. So, the cost is the distance in kilometers.

3.6.2 The function $h(x)$

In same way, the $h(x)$ is the heuristic function that is used to approximate distance from the current location to the goal state. This function is distinct because it is a mere estimation rather than an exact value. The more accurate the heuristic the better and faster the goal state is reach and with much more accuracy. To determine the value of $h(x)$ where x is the actual state, we have determined the difference between the remaining waste of some type to collect and the possible garbage to collect in that state x, if in that state the truck move to one container of type garbage. If in that state doesn't exist any garbage, the heuristic function only returns the value of total garbage remaining in the graph.

It is easy to understand that heuristics is admissible, since at the beginning the heuristic returns the maximum value and at the end 0, never overestimating, although it does not return a value in kilometers. We have hypothesized to perform the minimum spanning tree search on each node but we do not consider it meaningful since we would have a search within another search.

4 Development

4.1 Programming Languages, Tools, and APIs

Both the algorithm and the graphical interface were programmed in Java with support of the IntelliJ IDEA and Ecplise development environment programs. In the interface with the user we use Java Swing in the windows that allow the configuration of the application and to show some statistics. In map and solution view found, we embedded the dynamic modeling and analysis library of graphs GraphStream in Java Swing.

4.2 Application structure

* The project is divided into four folders: diagrams(diagrams), source code (src), documents (doc) and resources (res). In order to organize the project source code, the files with the code sources are subdivided into six packets:

- Graph - representation of the problem map;
- Gui - graphical interface that allows the simplified interaction of the program by the user.
- Logic - problem solving logic search algorithms.
- Tests - file that allows performing unit tests on important functions from the program.

We provide some uml diagrams to better present some key packages in this project like logicA.3 and graphA.4, and a module view of the projectA.5, all in the A.

4.3 Relevant details of implementation

We were careful to choose the data structures that could be most effective in representing and manipulating data. One example was the use of one priority queue in the A^* algorithm.

5 Experiences

In addition to the unit tests that test the important components of the program, experiments were also performed on the performance of the algorithm. We have used 3 graphs to test, with incremental number of nodes and edges, 3 heuristic functions, collectiong the following information, gathered in the table:

Nodes	Time execution(ms)	Visited nodes	Total cost(km)
6	32	121	196
12	47	74	496
18	187	540	913
30	785	1902	1155
60	94560	18221	2224

Table 1: Heuristic

Heuristic/Uniform Cost

Nodes	Time execution(ms)	Visited nodes	Total cost(km)
6	32/46	121/252	192/192
12	47/20657	74/28429	496/496

Table 2: heuristic vs uniform cost

As we can see in the previous tables, the heuristic used improves a lot the search time execution.

6 Conclusions

The application developed meets all the items of the list of requirements 3.5.

We want to increment the number of nodes and edges, maybe try and test with other algorithm to compare the solutions.

Cost minimization is important when planning routes serving the waste collection vehicles. Garbage truck must pass all the planned streets. Some of them must overcome a second time to get to the next street, now without receiving waste. This paper presents the method of determining the street, which must be passed more than one time and the sum of their length is the smallest possible, which significantly reduces costs.

7 Future enhancements

It is suggested that in future works the way of generating / importing Input data changes. Maybe we create some type of parse to get xml values, useful to download data in Open Street Map.

The next step in the planning garbage truck routes is an indication of the order of passing of the streets. In future, more variables can be added to truck, like fuel, hours of work, points to fuel the trucks and add more trucks to the fleet.

8 Resources

8.1 References

1. Slides from lectures classes
2. Stuart Russell, Peter Norvig *Artifical Intelligence A Modern Approach*, Pearson Education 3rd edition, 2010.
3. `http://www.gpsvisualizer.com/tutorials/waypoints.html`
4. `http://junit.org/junit5/`

8.2 Used software

1. `http://www.openstreetmap.org/`
2. `http://jgrapht.org/`

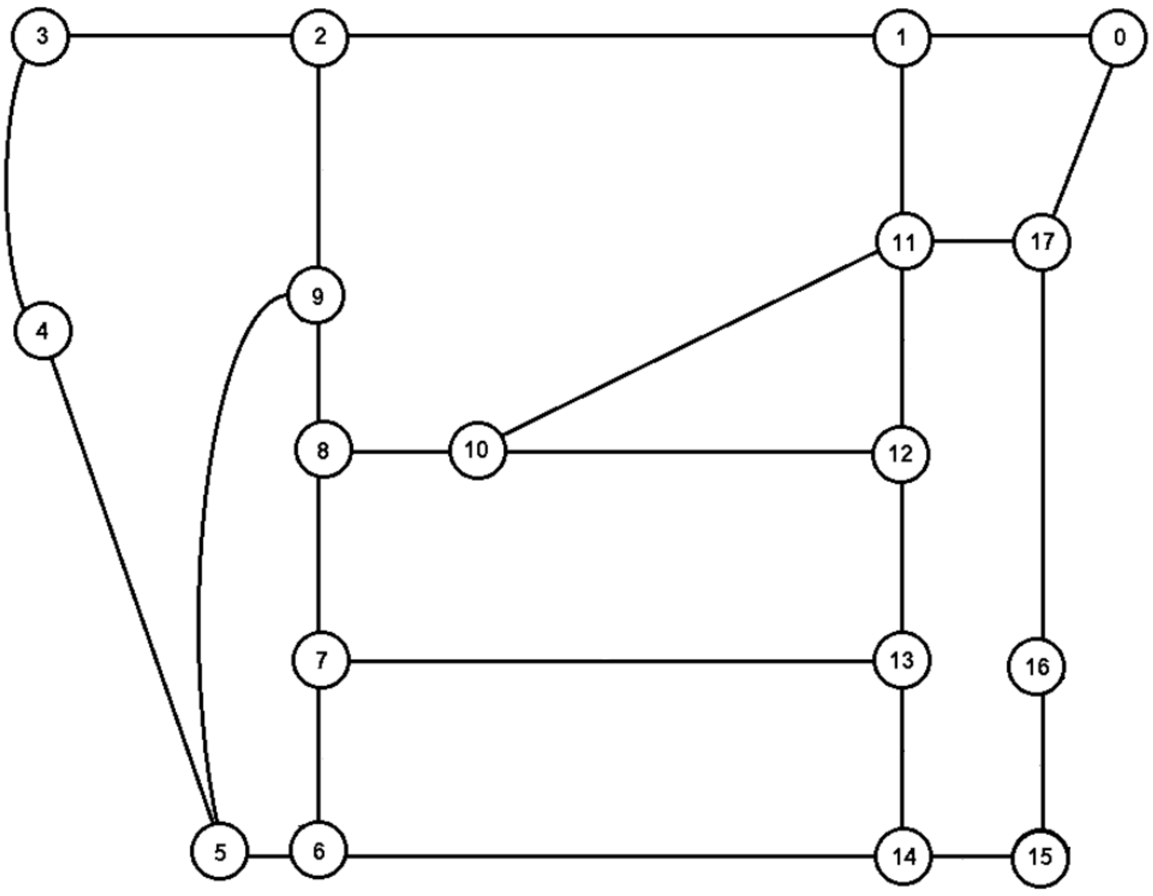
8.3 Effective work of each group member

Each element of the group, consisting of two students, initially worked on distinct parts of the project. As we were developing the project and advancing towards a final phase of it, we ended up working together because there was a fair division of the effort involved by each one. Thus, we agree on the following percentages:

- Artur Sousa Ferreira - 50%
- Nuno Miguel Rainho Valente - 50%

A Annex

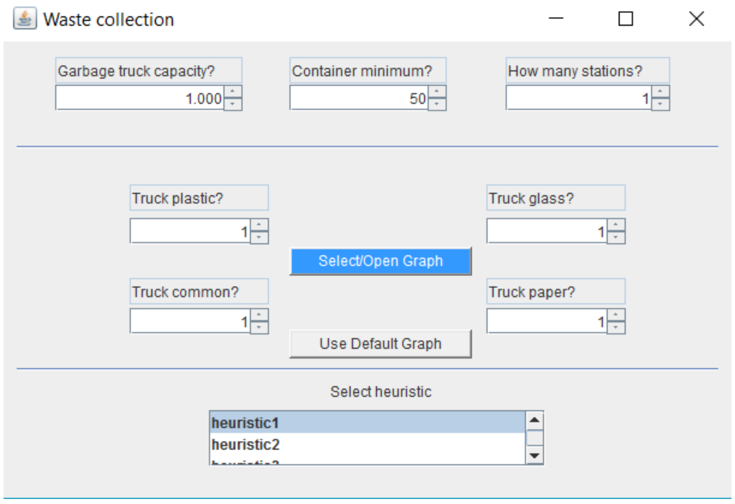
A.1 Example of graph



A.2 A Short User Manual

To run this application you can import all the source directory into your favourite IDE and run, choosing the InformationsRequest class as the entry point and the class that have the main function.

After that you can edit the information to run the apllication, Following successively the following tables:


Waste collection

Garbage truck capacity? 1.000

Container minimum? 50

How many stations? 1

Truck plastic? 1

Truck glass? 1

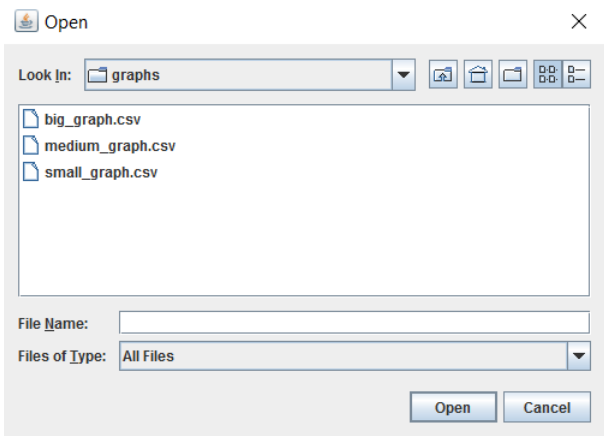
Truck common? 1

Truck paper? 1

Select heuristic

heuristic1

heuristic2


Open

Look In: graphs

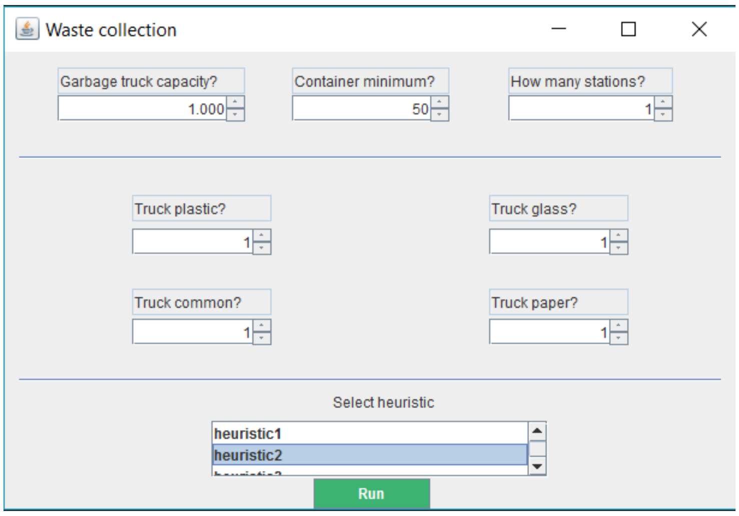
big_graph.csv

medium_graph.csv

small_graph.csv

File Name:

Files of Type: All Files


Waste collection

Garbage truck capacity? 1.000

Container minimum? 50

How many stations? 1

Truck plastic? 1

Truck glass? 1

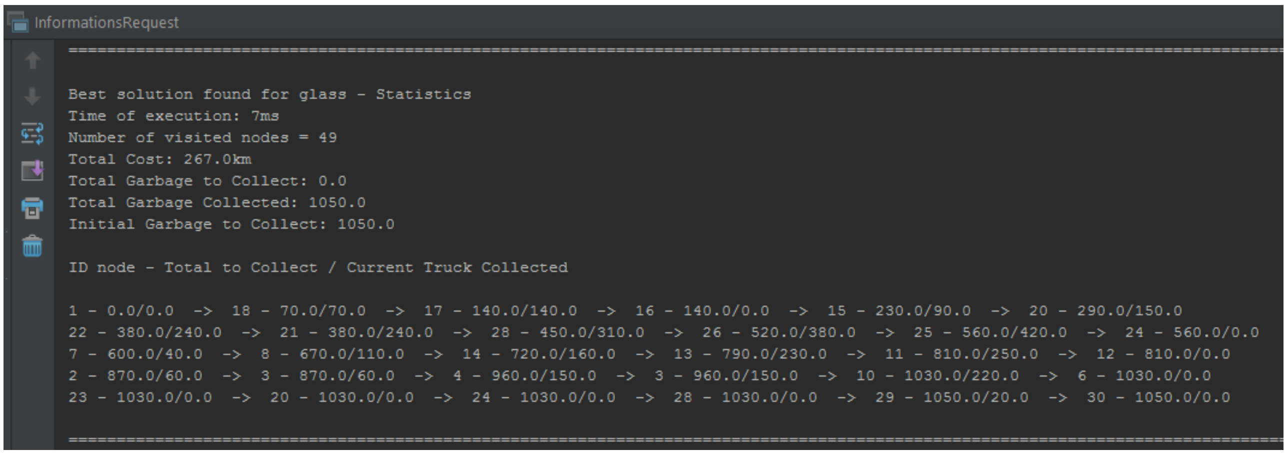
Truck common? 1

Truck paper? 1

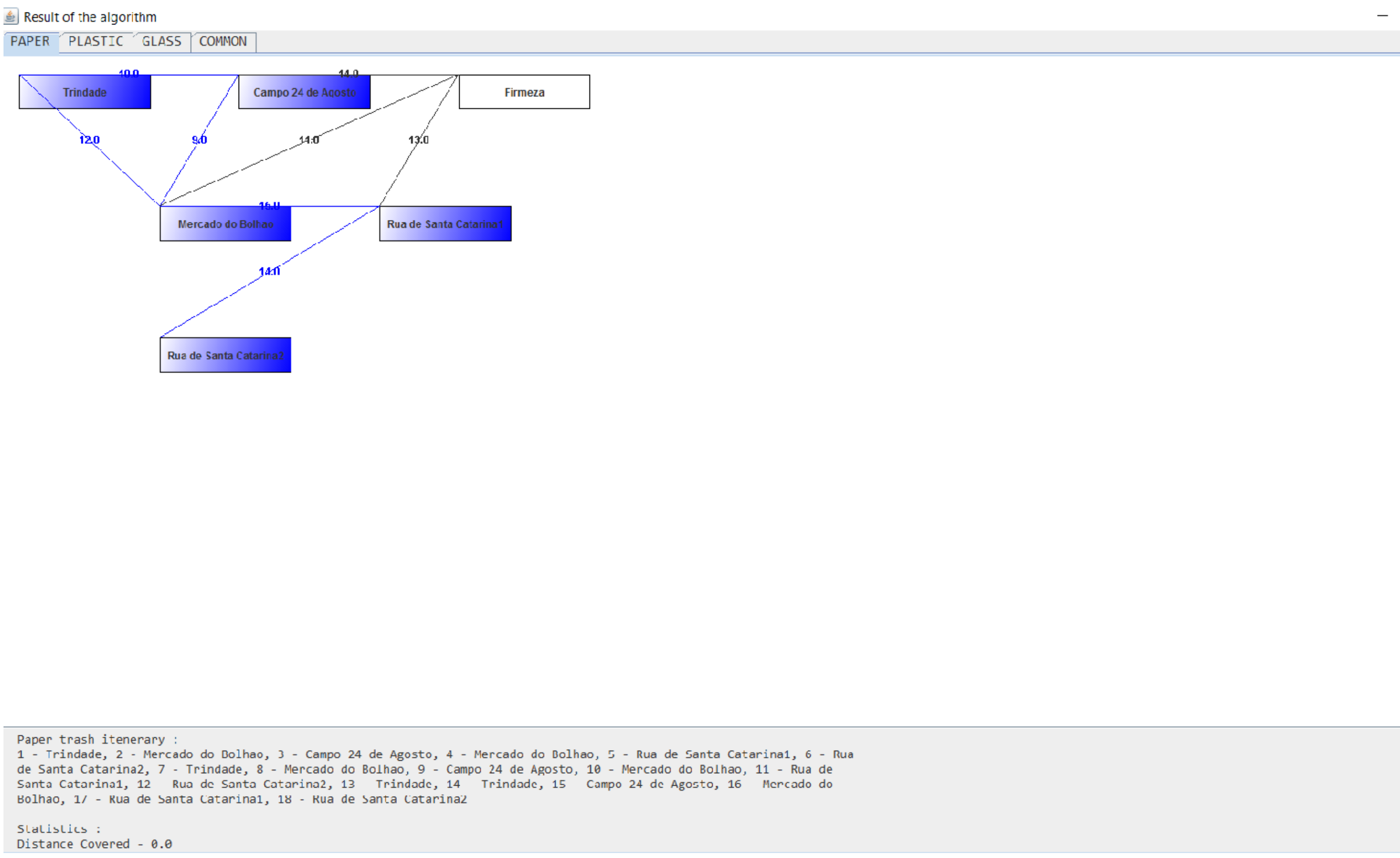
Select heuristic

heuristic1

heuristic2

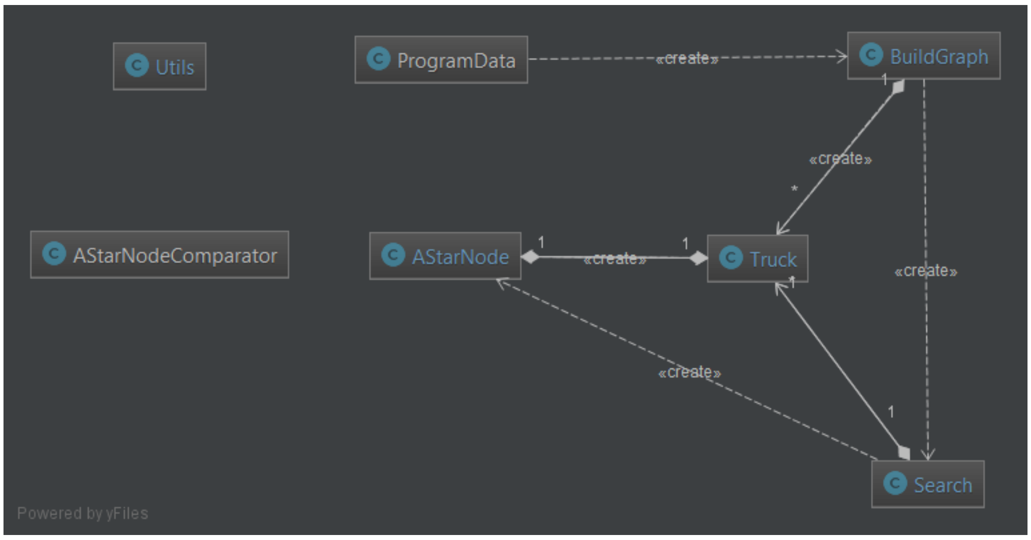


```
===========================================================================================================

Best solution found for glass - Statistics
Time of execution: 7ms
Number of visited nodes = 49
Total Cost: 267.0km
Total Garbage to Collect: 0.0
Total Garbage Collected: 1050.0
Initial Garbage to Collect: 1050.0

ID node - Total to Collect / Current Truck Collected

1 - 0.0/0.0  ->  18 - 70.0/70.0  ->  17 - 140.0/140.0  ->  16 - 140.0/0.0  ->  15 - 230.0/90.0  ->  20 - 290.0/150.0
22 - 380.0/240.0  ->  21 - 380.0/240.0  ->  28 - 450.0/310.0  ->  26 - 520.0/380.0  ->  25 - 560.0/420.0  ->  24 - 560.0/0.0
7 - 600.0/40.0  ->  8 - 670.0/110.0  ->  14 - 720.0/160.0  ->  13 - 790.0/230.0  ->  11 - 810.0/250.0  ->  12 - 810.0/0.0
2 - 870.0/60.0  ->  3 - 870.0/60.0  ->  4 - 960.0/150.0  ->  3 - 960.0/150.0  ->  10 - 1030.0/220.0  ->  6 - 1030.0/0.0
23 - 1030.0/0.0  ->  20 - 1030.0/0.0  ->  24 - 1030.0/0.0  ->  28 - 1030.0/0.0  ->  29 - 1050.0/20.0  ->  30 - 1050.0/0.0

===========================================================================================================
```

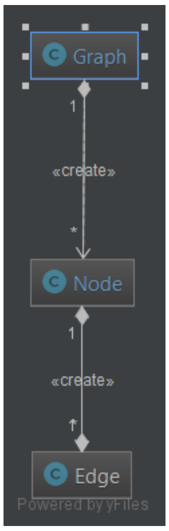


```
Paper trash itenerary :
1 - Trindade, 2 - Mercado do Bolhao, 3 - Campo 24 de Agosto, 4 - Mercado do Bolhao, 5 - Rua de Santa Catarina1, 6 - Rua
de Santa Catarina2, 7 - Trindade, 8 - Mercado do Bolhao, 9 - Campo 24 de Agosto, 10 - Mercado do Bolhao, 11 - Rua de
Santa Catarina1, 12   Rua de Santa Catarina2, 13   Trindade, 14   Trindade, 15   Campo 24 de Agosto, 16   Mercado do
Bolhao, 17 - Rua de Santa Catarina1, 18 - Rua de Santa Catarina2

Statistics :
Distance Covered - 0.0
```

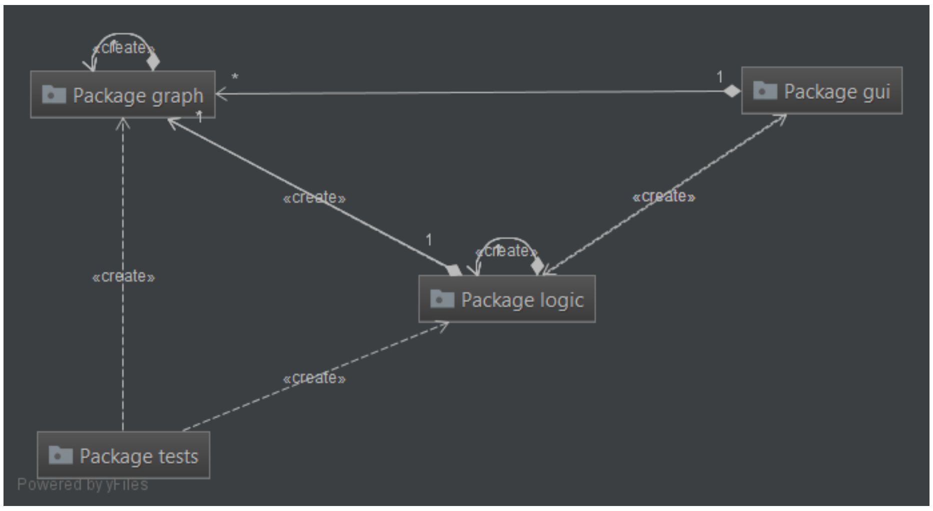
A.3 Logic package diagram



A.4 Graph package diagram



A.5 Module View UML



A.6 Graph examples used

```
[nodes];;;glass;paper;plastic;common
1;central;Trindade;0;0;0;0
2;true;Campo 24 de Agosto;60;60;50;100
3;false;Firmeza;0;0;0;0
4;true;Mercado do Bolhao;90;70;40;20
5;true;Rua de Santa Catarina1;0;0;0;0
6;station;Rua de Santa Catarina2;0;0;0;0
7;true;Rua de Santa Catarina3;40;80;40;80
8;true;Rua de Santa Catarina4;70;80;70;60
9;true;Rua de Santa Catarina5;0;90;70;0
10;true;Rua de Santa Catarina6;70;10;50;10
11;true;Rua de Santa Catarina7;20;40;20;80
12;station;Rua de Santa Catarina8;0;0;0;0
13;true;Rua de Santa Catarina9;70;80;90;40
14;true;Rua de Santa Catarina10;50;90;10;70
15;true;Rua de Santa Catarina11;90;30;90;60
16;station;Rua de Santa Catarina12;0;0;0;0
17;true;Rua de Santa Catarina13;70;40;70;80
18;true;Rua de Santa Catarina14;70;60;70;80
;;;;;;
[edges];;;;;;
1;2;10;;;;
1;18;12;;;;
2;3;14;;;;
2;12;9;;;;
3;4;11;;;;
3;10;13;;;;
4;5;15;;;;
5;6;14;;;;
6;7;10;;;;
6;10;12;;;;
10;9;14;;;;
```

```
9;8;9;;;;
9;11;11;;;;
8;7;13;;;;
8;14;15;;;;
7;15;18;;;;
11;12;10;;;;
11;13;12;;;;
15;16;14;;;;
13;14;9;;;;
12;13;11;;;;
12;18;13;;;;
18;17;15;;;;
16;17;9;;;;
14;15;6;;;;

[nodes];;;glass;paper;plastic;common
1;central;Trindade;0;0;0;0
2;true;Campo 24 de Agosto;60;60;50;100
3;false;Firmeza;0;0;0;0
4;true;Mercado do Bolhao;90;70;40;20
5;true;Rua de Santa Catarina1;0;0;0;0
6;station;Rua de Santa Catarina2;0;0;0;0
7;true;Rua de Santa Catarina3;40;80;40;80
8;true;Rua de Santa Catarina4;70;80;70;60
9;true;Rua de Santa Catarina5;0;90;70;0
10;true;Rua de Santa Catarina6;70;10;50;10
11;true;Rua de Santa Catarina7;20;40;20;80
12;station;Rua de Santa Catarina8;0;0;0;0
13;true;Rua de Santa Catarina9;70;80;90;40
14;true;Rua de Santa Catarina10;50;90;10;70
15;true;Rua de Santa Catarina11;90;30;90;60
16;station;Rua de Santa Catarina12;0;0;0;0
17;true;Rua de Santa Catarina13;70;40;70;80
18;true;Rua de Santa Catarina14;70;60;70;80
19;false;Rua de Santa Catarina15;0;0;0;0
20;true;Rua de Santa Catarina16;60;60;50;100
21;false;Rua de Santa Catarina17;0;0;0;0
22;true;Rua de Santa Catarina18;90;70;40;20
23;true;Rua de Santa Catarina19;0;0;0;0
24;station;Rua de Santa Catarina20;0;0;0;0
25;true;Rua de Santa Catarina21;40;80;40;80
26;true;Rua de Santa Catarina22;70;80;70;60
27;true;Rua de Santa Catarina23;0;90;70;0
28;true;Rua de Santa Catarina24;70;10;50;10
29;true;Rua de Santa Catarina25;20;40;20;80
30;station;Rua de Santa Catarina26;0;0;0;0
;;;;;;
[edges];;;;;;
1;2;10;;;;
1;18;12;;;;
2;3;14;;;;
2;12;9;;;;
3;4;11;;;;
```

```
3;10;13;;;;
4;5;15;;;;
5;6;14;;;;
6;7;10;;;;
6;10;12;;;;
10;9;14;;;;
9;8;9;;;;
9;11;11;;;;
8;7;13;;;;
8;14;15;;;;
7;15;18;;;;
11;12;10;;;;
11;13;12;;;;
15;16;14;;;;
13;14;9;;;;
12;13;11;;;;
12;18;13;;;;
18;17;15;;;;
16;17;9;;;;
14;15;6;;;;
19;6;11;;;;
19;22;10;;;;
20;23;5;;;;
20;15;10;;;;
20;24;6;;;;
20;21;5;;;;
20;22;5;;;;
21;22;5;;;;
21;28;7;;;;
21;27;7;;;;
23;7;5;;;;
23;6;5;;;;
24;7;13;;;;
24;16;10;;;;
24;25;5;;;;
24;28;3;;;;
25;15;10;;;;
25;26;7;;;;
26;28;5;;;;
26;27;4;;;;
26;30;4;;;;
27;30;4;;;;
28;29;7;;;;
29;30;5;;;;

[nodes];;;glass;paper;plastic;common
1;central;Trindade;0;0;0;0
2;true;Campo 24 de Agosto;60;60;50;100
3;false;Firmeza;0;0;0;0
4;true;Mercado do Bolhao;90;70;40;20
5;false;Rua de Santa Catarina1;0;0;0;0
6;station;Rua de Santa Catarina2;0;0;0;0
7;true;Rua de Santa Catarina3;40;80;40;80
```

```
8;true;Rua de Santa Catarina4;70;80;70;60
9;true;Rua de Santa Catarina5;0;90;70;0
10;true;Rua de Santa Catarina6;70;10;50;10
11;true;Rua de Santa Catarina7;20;40;20;80
12;station;Rua de Santa Catarina8;0;0;0;0
13;true;Rua de Santa Catarina9;70;80;90;40
14;true;Rua de Santa Catarina10;50;90;10;70
15;true;Rua de Santa Catarina11;90;30;90;60
16;station;Rua de Santa Catarina12;0;0;0;0
17;true;Rua de Santa Catarina13;70;40;70;80
18;true;Rua de Santa Catarina14;70;60;70;80
19;false;Rua de Santa Catarina15;0;0;0;0
20;true;Rua de Santa Catarina16;60;60;50;100
21;false;Rua de Santa Catarina17;0;0;0;0
22;true;Rua de Santa Catarina18;90;70;40;20
23;false;Rua de Santa Catarina19;0;0;0;0
24;station;Rua de Santa Catarina20;0;0;0;0
25;true;Rua de Santa Catarina21;40;80;40;80
26;true;Rua de Santa Catarina22;70;80;70;60
27;true;Rua de Santa Catarina23;0;90;70;0
28;true;Rua de Santa Catarina24;70;10;50;10
29;true;Rua de Santa Catarina25;20;40;20;80
30;station;Rua de Santa Catarina26;0;0;0;0
31;true;Rua de Santa Catarina27;20;40;20;80
32;station;Rua de Santa Catarina28;0;0;0;0
33;true;Rua de Santa Catarina29;70;80;90;40
34;true;Rua de Santa Catarina30;50;90;10;70
35;true;Rua de Santa Catarina31;90;30;90;60
36;station;Rua de Santa Catarina32;0;0;0;0
37;true;Rua de Santa Catarina33;70;40;70;80
38;true;Rua de Santa Catarina34;70;60;70;80
39;false;Rua de Santa Catarina35;0;0;0;0
40;true;Rua de Santa Catarina36;60;60;50;100
41;false;Rua de Santa Catarina37;0;0;0;0
42;true;Rua de Santa Catarina38;90;70;40;20
43;false;Rua de Santa Catarina39;0;0;0;0
44;station;Rua de Santa Catarina40;0;0;0;0
45;true;Rua de Santa Catarina41;40;80;40;80
46;true;Rua de Santa Catarina42;70;80;70;60
47;true;Rua de Santa Catarina43;0;90;70;0
48;true;Rua de Santa Catarina44;70;10;50;10
49;true;Rua de Santa Catarina45;20;40;20;80
50;station;Rua de Santa Catarina46;0;0;0;0
51;false;Rua de Santa Catarina47;0;0;0;0
52;station;Rua de Santa Catarina48;0;0;0;0
53;true;Rua de Santa Catarina49;40;80;40;80
54;true;Rua de Santa Catarina50;70;80;70;60
55;true;Rua de Santa Catarina51;0;90;70;0
56;true;Rua de Santa Catarina52;70;10;50;10
57;true;Rua de Santa Catarina53;20;40;20;80
58;station;Rua de Santa Catarina54;0;0;0;0
59;true;Rua de Santa Catarina55;70;80;90;40
60;true;Rua de Santa Catarina56;50;90;10;70
```

```
;;;;;;
[edges];;;;;;
1;2;10;;;;
1;18;12;;;;
2;3;14;;;;
2;12;9;;;;
3;4;11;;;;
3;10;13;;;;
4;5;15;;;;
5;6;14;;;;
6;7;10;;;;
6;10;12;;;;
10;9;14;;;;
9;8;9;;;;
9;11;11;;;;
8;7;13;;;;
8;14;15;;;;
7;15;18;;;;
11;12;10;;;;
11;13;12;;;;
15;16;14;;;;
13;14;9;;;;
12;13;11;;;;
12;18;13;;;;
18;17;15;;;;
16;17;9;;;;
14;15;6;;;;
19;6;11;;;;
19;22;10;;;;
20;23;5;;;;
20;15;10;;;;
20;24;6;;;;
20;21;5;;;;
20;22;5;;;;
21;22;5;;;;
21;28;7;;;;
21;27;7;;;;
23;7;5;;;;
23;6;5;;;;
24;7;13;;;;
24;16;10;;;;
24;25;5;;;;
24;28;3;;;;
25;15;10;;;;
25;26;7;;;;
26;28;5;;;;
26;27;4;;;;
26;30;4;;;;
27;30;4;;;;
28;29;7;;;;
29;30;5;;;;
30;22;7;;;;
31;21;8;;;;
```

```
31;29;6;;;;
32;25;10;;;;
32;28;12;;;;
32;26;4;;;;
32;30;2;;;;
33;6;20;;;;
33;24;10;;;;
33;29;5;;;;
34;33;2;;;;
34;30;3;;;;
34;16;10;;;;
35;27;19;;;;
35;39;19;;;;
35;42;13;;;;
36;41;14;;;;
36;43;17;;;;
36;6;11;;;;
37;14;14;;;;
37;2;12;;;;
37;9;13;;;;
38;37;20;;;;
38;21;11;;;;
38;39;19;;;;
39;28;11;;;;
39;7;6;;;;
39;27;18;;;;
40;43;16;;;;
40;35;12;;;;
40;30;12;;;;
41;1;16;;;;
41;6;14;;;;
42;21;8;;;;
42;31;10;;;;
42;25;14;;;;
43;28;18;;;;
43;39;19;;;;
43;50;5;;;;
44;6;18;;;;
44;18;18;;;;
44;1;15;;;;
45;18;15;;;;
45;39;10;;;;
46;9;18;;;;
46;39;16;;;;
47;49;3;;;;
47;49;11;;;;
48;10;8;;;;
48;36;3;;;;
49;18;11;;;;
49;30;19;;;;
49;27;16;;;;
50;38;15;;;;
50;5;12;;;;
```

```
50;3;14;;;;
51;2;3;;;;
51;35;8;;;;
51;37;9;;;;
52;28;2;;;;
52;44;9;;;;
52;39;7;;;;
53;45;8;;;;
53;29;10;;;;
54;33;9;;;;
54;54;8;;;;
54;30;4;;;;
55;27;11;;;;
55;44;5;;;;
55;55;4;;;;
56;60;4;;;;
56;39;7;;;;
57;54;10;;;;
57;35;4;;;;
57;40;4;;;;
58;25;5;;;;
58;22;3;;;;
58;38;10;;;;
59;40;7;;;;
59;13;5;;;;
60;42;4;;;;
60;57;9;;;;
60;30;10;;;;
```

A.7 Source Code

A.7.1 Package graph

```java
package graph;

import java.util.ArrayList;
import java.util.HashMap;

import logic.Utils;

public class Graph {
    private ArrayList<Node> nodes;
    private double wastePaper=0.0,
            wastePlastic=0.0,
            wasteGlass=0.0,
            wasteCommon=0.0;
    private boolean original;
    private HashMap<Integer, Node> containers;

    public Graph() {
        this.original = true;
        this.nodes = new ArrayList<Node>();
        for (int i = 0; i < this.nodes.size(); i++) {
            if (this.nodes.get(i).getType().equals(Utils.TRUE_GARBAGE)) {
                this.wastePaper += this.nodes.get(i).
                    getGarbageContainerByType(Utils.PAPER);
                this.wastePlastic += this.nodes.get(i).
                    getGarbageContainerByType(Utils.PLASTIC);
                this.wasteGlass += this.nodes.get(i).
                    getGarbageContainerByType(Utils.GLASS);
                this.wasteCommon += this.nodes.get(i).
                    getGarbageContainerByType(Utils.COMMON);
            }
        }
        this.containers = new HashMap<Integer, Node>();
    }

    public Graph(Graph newG) {
        this.original = false;
        this.nodes = new ArrayList<Node>(newG.getNumNodes());
        for(Node n : newG.getNodes())
        this.nodes.add(new Node(n));

        for (int i = 0; i < newG.getNodes().size(); i++) {
            if (newG.getNodes().get(i).getType().equals(Utils.TRUE_GARBAGE)
                ) {
                this.wastePaper += newG.getNodes().get(i).
                    getGarbageContainerByType(Utils.PAPER);
                this.wastePlastic += newG.getNodes().get(i).
                    getGarbageContainerByType(Utils.PLASTIC);
                this.wasteGlass += newG.getNodes().get(i).
                    getGarbageContainerByType(Utils.GLASS);
                this.wasteCommon += newG.getNodes().get(i).
                    getGarbageContainerByType(Utils.COMMON);
            }
        }

        this.containers = newG.getGraphContainers();
    }

    public ArrayList<Node> getNodes() {
        return this.nodes;
    }

    public Node findNode(int nodeId) {
        for (int i = 0; i < this.nodes.size(); i++) {
            if (this.nodes.get(i).getId() == nodeId) {
                return this.nodes.get(i);
            }
        }
        return null;
    }

    public void setWasteByType(String wasteType, double collected){
        for (int i = 0; i < this.nodes.size(); i++) {
            if (this.nodes.get(i).getType().equals(Utils.TRUE_GARBAGE)) {
                if(this.nodes.get(i).getGarbageContainerByType(wasteType)
                    >= collected)
                 this.nodes.get(i).setGarbageContainer(wasteType,
                     collected);
            }
        }
    }

    public double getTotalWaste(){return wastePaper+wastePlastic+wasteGlass
        +wasteCommon;}

    public double getTotalGarbageByTypeWaste(String wasteType){
        double total=0.0;
        for(Node n : this.nodes){
            if(n.getType() == Utils.TRUE_GARBAGE)
                total += n.getGarbageContainerByType(wasteType);
        }
        return total;
    }

    public double getTotalGarbageByTypeWasteWithMinimumLevelInContainers(
        String wasteType){
        double total=0.0;
        for(Node n : this.nodes){
            if(n.getType() == Utils.TRUE_GARBAGE)
                if(n.getGarbageContainerByType(wasteType) > Utils.
                    MinimumGarbageCapacity)
                    total += n.getGarbageContainerByType(wasteType);
        }
        return total;
    }

    public boolean addNode(Node node) {
        if (this.nodes.contains(node))
            return false;
        this.nodes.add(node);
        return true;
    }

    public boolean removeNode(Node node) {
        if (this.nodes.remove(node)) {
            for (int i = 0; i < this.nodes.size(); i++) {
                this.nodes.get(i).removeEdgeTo(node);
            }
            return true;
        }
        return false;
    }

    public boolean addEdge(Node source, Node destiny, double distance) {
        if (this.nodes.contains(source) && this.nodes.contains(destiny)) {
            source.addEdge(destiny, distance);
            return true;
        }
        return false;
    }

    public boolean removeEdge(Node source, Node destiny) {
        if (this.nodes.contains(source) && this.nodes.contains(destiny)) {
            return source.removeEdgeTo(destiny);
        }
        return false;
    }

    public int getNumNodes() {
        return this.nodes.size();
    }

    public int getNumEdges() {
        int count = 0;
        for (int i = 0; i < this.nodes.size(); i++) {
            count += this.nodes.get(i).getOutEdges().size();
        }
        return count;
    }

    public boolean findEdge(Node a, Node b) {
        for (int i = 0; i < this.nodes.size(); i++) {
            for(int j = 0 ; j < this.nodes.get(i).getOutEdges().size() ; j
                ++){
                if((this.nodes.get(i).getOutEdges().get(j).getSource().
                    getName() == a.getName()) && (nodes.get(i).getOutEdges()
                    .get(j).getDestiny().getName() == b.getName()))
                    return true;
            }
        }
        return false;
    }

    public double calcDistance(Node a, Node b) {
        for (int i = 0; i < this.nodes.size(); i++) {
            for(int j = 0 ; j < this.nodes.get(i).getOutEdges().size() ; j
                ++){
```

```
                if((this.nodes.get(i).getOutEdges().get(j).getSource().
                    getName() == a.getName()) && (this.nodes.get(i).
                    getOutEdges().get(j).getDestiny().getName() == b.getName
                    ()))
                     return this.nodes.get(i).getOutEdges().get(j).
                        getDistance();
            }
        }
        return 0.0;
    }

    @Override
    public String toString() {
        StringBuilder strb = new StringBuilder();
        for (int i = 0; i < this.nodes.size(); i++) {
            strb.append(this.nodes.get(i).toString() + "\n");
        }
        String str = strb.toString();
        return str;
    }

    public void setGraphContainers(HashMap<Integer, Node> containers){
        this.containers = containers;
    }

    public HashMap<Integer, Node> getGraphContainers(){
        return this.containers;
    }

    public double getTotalInContainers(String typeofWaste){
        double waste = 0.0;
        for (HashMap.Entry<Integer, Node> entry : this.containers.entrySet
            ()) {
            Node node = entry.getValue();
            waste += node.getGarbageContainerByType(typeofWaste);
        }
        return waste;
    }

    public int getNumberOfEmptyContainers(String typeofWaste){
        int nr = 0;
        for (HashMap.Entry<Integer, Node> entry : this.containers.entrySet
            ()) {
            Node node = entry.getValue();

            if(node.getGarbageContainerByType(typeofWaste) == 0.0)
                nr++;
        }
        return nr;
    }
/*
    @Override
    public boolean equals(Object obj) {
        if (obj == null)
            return false;

        if (this == obj)
            return true;

        if (this.getClass() != obj.getClass())
            return false;

        Graph graph = (Graph) obj;

        for(int i = 0 ; i < this.getNumNodes() ; i++){
            if(this.getNodes().get(i).getType().equals(Utils.TRUE_GARBAGE))
                if (this.getNodes().get(i).getGarbageContainerByType(Utils.
                    PAPER) != graph.getNodes().get(i).
                    getGarbageContainerByType(Utils.PAPER)){
                     return false;
                }
        }
        return true;
    }*/
}
```

```
package graph;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import logic.Utils;

public class Node {

    private double distance;
    private String type;
    private ArrayList<Edge> outEdges;
    private String name;
    private Node parent;
    private Map<String, Double> garbageContainer;
    private static int current_id = 0;
    private int id;

    public Node(Node node) {
        if (node != null) {
            this.type = node.type;
            this.name = node.name;
            this.outEdges = new ArrayList<Edge>();
            this.outEdges = node.getOutEdges();
            if(this.type == Utils.TRUE_GARBAGE){
                this.garbageContainer = new HashMap<String, Double>();
                this.garbageContainer.put("glass", node.
                    getGarbageContainerByType(Utils.GLASS));
                this.garbageContainer.put("paper", node.
                    getGarbageContainerByType(Utils.PAPER));
                this.garbageContainer.put("plastic", node.
                    getGarbageContainerByType(Utils.PLASTIC));
                this.garbageContainer.put("common", node.
                    getGarbageContainerByType(Utils.COMMON));
            }

            this.id = node.id;
        }
    }

    public Node(int id, String type, String nameStreet, double glass,
       double paper, double plastic, double common) {
        this.name = nameStreet;
        this.outEdges = new ArrayList<Edge>();

        if (type.equals("central"))
            this.type = Utils.CENTRAL;
        else if (type.equals("station"))
            this.type = Utils.STATION;
        else if (type.equals("false"))
            this.type = Utils.FALSE_GARBAGE;
        else{
            this.garbageContainer = new HashMap<String, Double>();
            this.garbageContainer.put("glass", glass);
            this.garbageContainer.put("paper", paper);
            this.garbageContainer.put("plastic", plastic);
            this.garbageContainer.put("common", common);
            this.type = Utils.TRUE_GARBAGE;
        }
        ++current_id;
        this.id = current_id;
    }

    // for test class only
    public Node(int id, String name, double distance) {
        this.id = id;
        this.name = name;
        this.setDistance(distance);
        this.outEdges = new ArrayList<Edge>();
    }

    public ArrayList<Edge> cloneList(ArrayList<Edge> list) {
        ArrayList<Edge> clone = new ArrayList<Edge>(list.size());
        for (Edge item : list)
            clone.add(new Edge(item));
        return clone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public void setGarbageStation(boolean garbageStation) {
        if (garbageStation)
            this.type = Utils.TRUE_GARBAGE;
        else
            this.type = Utils.FALSE_GARBAGE;
    }

    public ArrayList<Edge> getOutEdges() {
        return this.outEdges;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null)
            return false;

        if (this == obj)
            return true;

        if (this.getClass() != obj.getClass())
            return false;

        Node node = (Node) obj;

        if (this.id == node.id)
            return true;
        else
            return false;
    }

    public boolean removeEdgeTo(Node node) {
        for (int i = 0; i < outEdges.size(); i++) {
            if (outEdges.get(i).getDestiny().equals(node)) {
                outEdges.remove(i);
                return true;
            }
        }
        return false;
    }

    public void addEdge(Node destiny, double distance) {
        Edge edge = new Edge(this, destiny, distance);
        this.outEdges.add(edge);
    }

    public String getName() {
        return this.name;
    }
    public String getType() {
        return this.type;
    }

    @Override
    public String toString() {

        StringBuilder strN = new StringBuilder();
        strN.append(this.getId() + " " + this.getName() + "\n");

        for (Edge temp : this.outEdges) {
            strN.append("\t" + temp.toString() + "\n");
        }

        return strN.toString();
    }

    public void setOutEdges(ArrayList<Edge> outEdges2) {
        this.outEdges = outEdges2;
    }

    public Node getParent() {
        return this.parent;
    }

    public void setParent(Node current) {
        this.parent = current;
    }

    public void setGarbageContainer(String typeGarbage, double collected){
        double temp = garbageContainer.get(typeGarbage);
        garbageContainer.replace(typeGarbage, temp - collected);
    }

    public Map<String, Double> getGarbageContainer(){
        return this.garbageContainer;
    }

    public double getGarbageContainerByType(String wasteType){
        return this.garbageContainer.get(wasteType);
    }

    public double getDistance() {
        return this.distance;
    }

    public void setDistance(double distance) {
        this.distance = distance;
    }
}
```

```
package graph;

public class Edge {

    private Node source;
    private Node destiny;
    private double distance;

    public Edge(Node source, Node destiny, double distance) {
        this.setSource(source);
        this.setDestiny(destiny);
        this.distance = distance;
    }

    public Edge(Edge item) {
        this.source = item.getSource();
        this.destiny = item.getDestiny();
        this.distance = item.getDistance();
    }
```

```java

    public boolean lessDistance(Edge edge) {
        if (edge != null)
            return this.distance < edge.distance;
        return false;
    }

    public boolean greaterDistance(Edge edge) {
        if (edge != null)
            return this.distance > edge.distance;
        return false;
    }

    public Node getSource() {
        return source;
    }

    public void setSource(Node source) {
        this.source = source;
    }

    public Node getDestiny() {
        return destiny;
    }

    public void setDestiny(Node destiny) {
        this.destiny = destiny;
    }

    public double getDistance() {
        return distance;
    }

    public void setDistance(double distance) {
        this.distance = distance;
    }

    @Override
    public boolean equals(Object obj) {

        if (obj == null)
            return false;

        if (this == obj)
            return true;

        if (this.getClass() != obj.getClass())
            return false;

        Edge edge = (Edge) obj;

        return this.distance == edge.distance && this.source.equals(edge.
           source) && this.destiny.equals(edge.destiny);
    }

    @Override
    public String toString() {
        String strE = (this.getSource().getName() + " ---> " + this.
           getDestiny().getName() + " - " + this.getDistance()
                + "km");
        return strE;
    }
}
```

A.7.2 Package gui

```java
package gui;

import java.awt.EventQueue;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import javax.swing.DefaultListModel;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JScrollPane;
import javax.swing.JSeparator;
import javax.swing.JSpinner;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.SpinnerModel;
import javax.swing.SpinnerNumberModel;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import logic.ProgramData;
import logic.Utils;
import java.awt.Button;
import java.awt.Label;
import java.awt.Color;
import java.awt.SystemColor;

public class InformationsRequest {

    private JFrame frame;
    private JTextField txtHowManyTrucks;
    private JTextField truckPlastic1;
    private JTextField truckGlass1;
    private JTextField truckCommon1;
    private JTextField stationsNumber;
    private JTextField garbageTruckCapacity;
    private JTextField containerMinimum;
    private SpinnerModel spinnerModel;
    private String heuristic;

    protected int truckPlastic = 1;
    protected int truckPaper = 1;
    protected int truckGlass = 1;
    protected int truckCommon = 1;
    protected int numberOfStations = 1;
    protected int truckCapacity = 1000;
    protected int minimumLevelContainer = 50;

    protected String current;
    protected File file;
    private JButton btnSubmit;
    private Button button;
    private Button defaultButton;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    InformationsRequest window = new InformationsRequest();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public InformationsRequest() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 626, 427);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);
        frame.setTitle("Waste collection");
        Utils.MinimumGarbageCapacity = minimumLevelContainer;

        /* Paper */
        txtHowManyTrucks = new JTextField();
        txtHowManyTrucks.setEditable(false);
        txtHowManyTrucks.setText("Truck paper?");
        txtHowManyTrucks.setBounds(403, 197, 116, 22);
        frame.getContentPane().add(txtHowManyTrucks);
        txtHowManyTrucks.setColumns(10);

        spinnerModel = new SpinnerNumberModel(truckPaper, // initial value
                1, // min
                10, // max
                1);// step
        JSpinner truckPaperSpinner = new JSpinner(spinnerModel);

        truckPaperSpinner.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                String current = ((JSpinner) e.getSource()).getValue().
                    toString();
                truckPaper = Integer.parseInt(current);
            }
        });
        truckPaperSpinner.setBounds(403, 222, 116, 22);
        frame.getContentPane().add(truckPaperSpinner);

        /* Glass */
        truckGlass1 = new JTextField();
        truckGlass1.setEditable(false);
        truckGlass1.setText("Truck glass?");
        truckGlass1.setColumns(10);
        truckGlass1.setBounds(403, 119, 116, 22);
        frame.getContentPane().add(truckGlass1);

        spinnerModel = new SpinnerNumberModel(truckGlass, // initial value
                1, // min
                10, // max
                1);// step
        JSpinner truckGlassSpinner = new JSpinner(spinnerModel);
        truckGlassSpinner.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                String current = ((JSpinner) e.getSource()).getValue().
                    toString();
                truckGlass = Integer.parseInt(current);
            }
        });
        truckGlassSpinner.setBounds(403, 147, 116, 22);
        frame.getContentPane().add(truckGlassSpinner);

        /* Common */
        truckCommon1 = new JTextField();
        truckCommon1.setEditable(false);
        truckCommon1.setText("Truck common?");
        truckCommon1.setColumns(10);
        truckCommon1.setBounds(106, 197, 116, 22);
        frame.getContentPane().add(truckCommon1);

        spinnerModel = new SpinnerNumberModel(truckCommon, // initial value
                1, // min
                10, // max
                1);// step
        JSpinner truckCommonSpinner = new JSpinner(spinnerModel);
        truckCommonSpinner.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                String current = ((JSpinner) e.getSource()).getValue().
                    toString();
                truckCommon = Integer.parseInt(current);
            }
        });
        truckCommonSpinner.setBounds(106, 222, 116, 22);
        frame.getContentPane().add(truckCommonSpinner);

        /* Plastic */
        truckPlastic1 = new JTextField();
        truckPlastic1.setEditable(false);
        truckPlastic1.setText("Truck plastic?");
        truckPlastic1.setColumns(10);
        truckPlastic1.setBounds(106, 119, 116, 22);
        frame.getContentPane().add(truckPlastic1);

        spinnerModel = new SpinnerNumberModel(truckPlastic, // initial
            value
                1, // min
                10, // max
                1);// step
        JSpinner truckPlasticSpinner = new JSpinner(spinnerModel);
        truckPlasticSpinner.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                String current = ((JSpinner) e.getSource()).getValue().
                    toString();
                truckPlastic = Integer.parseInt(current);

            }
        });
        truckPlasticSpinner.setBounds(106, 147, 116, 22);
        frame.getContentPane().add(truckPlasticSpinner);

        /* Number of stations */
        stationsNumber = new JTextField();
        stationsNumber.setEditable(false);
        stationsNumber.setText("How many stations?");
        stationsNumber.setColumns(10);
        stationsNumber.setBounds(419, 13, 137, 22);
        frame.getContentPane().add(stationsNumber);

        spinnerModel = new SpinnerNumberModel(numberOfStations, // initial
            value
                1, // min
                3, // max
                1);// step
        JSpinner stationsNumberSpinnner = new JSpinner(spinnerModel);
        stationsNumberSpinnner.addChangeListener(new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                String current = ((JSpinner) e.getSource()).getValue().
                    toString();
                numberOfStations = Integer.parseInt(current);
            }
        });
        stationsNumberSpinnner.setBounds(419, 36, 137, 22);
        frame.getContentPane().add(stationsNumberSpinnner);

        /* Truck capacity */
        garbageTruckCapacity = new JTextField();
        garbageTruckCapacity.setEditable(false);
        garbageTruckCapacity.setText("Garbage truck capacity?");
        garbageTruckCapacity.setColumns(10);
        garbageTruckCapacity.setBounds(44, 13, 156, 22);
        frame.getContentPane().add(garbageTruckCapacity);

        JSpinner garbageTruckCapacitySpinner = new JSpinner();
        spinnerModel = new SpinnerNumberModel(truckCapacity, // initial
            value
                0, // min
                3000, // max
                100);// step
        garbageTruckCapacitySpinner = new JSpinner(spinnerModel);
        garbageTruckCapacitySpinner.addChangeListener(new ChangeListener()
            {
             @Override
             public void stateChanged(ChangeEvent e) {
                 String current = ((JSpinner) e.getSource()).getValue().
                     toString();
                 truckCapacity = Integer.parseInt(current);
             }
        });
        garbageTruckCapacitySpinner.setBounds(44, 36, 156, 22);
        frame.getContentPane().add(garbageTruckCapacitySpinner);

        /* Container minimum level */

        containerMinimum = new JTextField();
        containerMinimum.setEditable(false);
        containerMinimum.setText("Container minimum?");
        containerMinimum.setColumns(10);
        containerMinimum.setBounds(239, 13, 131, 22);
        frame.getContentPane().add(containerMinimum);

        spinnerModel = new SpinnerNumberModel(minimumLevelContainer, //
            initial
                // value
                0, // min
                Utils.garbageCapacity, // max
                10);// step
        JSpinner containerMinimumSpinner = new JSpinner(spinnerModel);
        containerMinimumSpinner.addChangeListener(new ChangeListener() {
             @Override
             public void stateChanged(ChangeEvent e) {
                 String current = ((JSpinner) e.getSource()).getValue().
                     toString();
                 minimumLevelContainer = Integer.parseInt(current);
                 Utils.MinimumGarbageCapacity = minimumLevelContainer;
             }
        });
        containerMinimumSpinner.setBounds(239, 36, 131, 22);
        frame.getContentPane().add(containerMinimumSpinner);

        JSeparator separator = new JSeparator();
        separator.setBounds(12, 87, 584, 2);
        frame.getContentPane().add(separator);

        JSeparator separator_1 = new JSeparator();
        separator_1.setBounds(12, 272, 584, 2);
        frame.getContentPane().add(separator_1);

        /* Select heuristic */

        Label label = new Label("Select heuristic");
        label.setAlignment(Label.CENTER);
        label.setBounds(239, 280, 152, 24);
        frame.getContentPane().add(label);

        DefaultListModel<String> listModel = new DefaultListModel<String>()
            ;
        listModel.addElement(Utils.HEURISTIC1);
        listModel.addElement(Utils.HEURISTIC2);
        listModel.addElement(Utils.HEURISTIC3);
        JScrollPane scrollPane = new JScrollPane();
        scrollPane.setBounds(172, 307, 280, 47);
        frame.getContentPane().add(scrollPane);

        JList<String> list = new JList<String>(listModel);
        scrollPane.setViewportView(list);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        list.setSelectedIndex(0);
        heuristic = list.getSelectedValue();
        list.addListSelectionListener(new ListSelectionListener() {

            @Override
            public void valueChanged(ListSelectionEvent e) {
                heuristic = list.getSelectedValue();
            }
        });
        list.setVisibleRowCount(2);

        /* select file */

        final JFileChooser  fileDialog = new JFileChooser(Utils.graphFile);
        button = new Button("Select/Open Graph");
        button.setForeground(Color.WHITE);
        button.setBackground(SystemColor.textHighlight);
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int returnVal = fileDialog.showOpenDialog(frame);
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    file = fileDialog.getSelectedFile();
                    btnSubmit.setVisible(true);
                    defaultButton.setVisible(false);
                    button.setVisible(false);
                    button.setBackground(frame.getBackground());
                    button.setForeground(Color.BLACK);
                }
                else{
                    btnSubmit.setVisible(false);
                }
            }
        });
        button.setBounds(239, 171, 152, 24);
        frame.getContentPane().add(button);

        /* Button submit */

        btnSubmit = new JButton("Run");
        btnSubmit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
            }
        });
        btnSubmit.setForeground(Color.WHITE);
        btnSubmit.setBackground(new Color(60, 179, 113));
        btnSubmit.setVisible(false);
        btnSubmit.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent arg0) {
                frame.dispose();
                new ProgramData(truckCapacity, numberOfStations,
                    minimumLevelContainer, truckPlastic, truckPaper,
                        truckGlass, truckCommon, heuristic, file);
```

```java
                }
            });
            btnSubmit.setBounds(257, 355, 97, 25);
            frame.getContentPane().add(btnSubmit);

            /* Default graph */

            defaultButton = new Button("Use Default Graph");
            defaultButton.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent arg0) {
                    file = new File(Utils.defaultFileGraph);
                    button.setVisible(false);
                    btnSubmit.setVisible(true);
                    defaultButton.setVisible(false);
                }
            });
            defaultButton.setBounds(239, 240, 152, 24);
            frame.getContentPane().add(defaultButton);


        }
}
```

```java
package gui;

import graph.Edge;
import graph.Graph;
import graph.Node;
import logic.ProgramData;
import java.awt.Toolkit;

import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;

import org.jgraph.JGraph;
import java.awt.Color;
import java.awt.geom.Rectangle2D;
import java.util.List;
import javax.swing.BorderFactory;

import org.jgraph.graph.DefaultEdge;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.DefaultGraphModel;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphModel;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JTabbedPane;
import javax.swing.JPanel;
import java.awt.Font;

public class Result extends JFrame {

    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public JFrame frmResult;
    private JLabel statistics;

    protected String current;
    private int screenWidth;
    private int screenHeight;
    private double paper_distanceCovered;
    private double plastic_distanceCovered;
    private double glass_distanceCovered;
    private double common_distanceCovered;
    private List<Node> itineraryPaper;
    private List<Node> itineraryPlastic;
    private List<Node> itineraryGlass;
    private List<Node> itineraryCommon;
    private JScrollPane graphResultPaper;
    private JScrollPane graphResultGlass;
    private JScrollPane graphResultPlastic;
    private JScrollPane graphResultCommon;

    /**
     * Create the application.
     *
     * @param itinerary
     * @wbp.parser.constructor
     */
    public Result(Graph graph, List<Node> itinerary, double distanceCovered
        ) {

        this.paper_distanceCovered = distanceCovered;
        this.plastic_distanceCovered = distanceCovered;
        this.glass_distanceCovered = distanceCovered;
        this.common_distanceCovered = distanceCovered;

        this.itineraryPaper = itinerary;
        this.itineraryPlastic = itinerary;
        this.itineraryGlass = itinerary;
        this.itineraryCommon = itinerary;

        this.screenWidth = (int)(Toolkit.getDefaultToolkit().getScreenSize
            ().getWidth());
        this.screenHeight = (int)(Toolkit.getDefaultToolkit().getScreenSize
            ().getHeight());

        initialize();
    }

    public Result(Graph graph, List<Node> itineraryPaper, double
        paper_distanceCovered, List<Node> itineraryPlastic, double
        plastic_distanceCovered, List<Node> itineraryGlass, double
        glass_distanceCovered, List<Node> itineraryCommon, double
        common_distanceCovered) {

        this.paper_distanceCovered = paper_distanceCovered;
        this.plastic_distanceCovered = plastic_distanceCovered;
        this.glass_distanceCovered = glass_distanceCovered;
        this.common_distanceCovered = common_distanceCovered;

        this.itineraryPaper = itineraryPaper;
        this.itineraryPlastic = itineraryPlastic;
        this.itineraryGlass = itineraryGlass;
        this.itineraryCommon = itineraryCommon;

        this.screenWidth = (int)(Toolkit.getDefaultToolkit().getScreenSize
            ().getWidth());
        this.screenHeight = (int)(Toolkit.getDefaultToolkit().getScreenSize
            ().getHeight());

        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {

        /* ALL WINDOWS FRAME */
        frmResult = new JFrame();
        frmResult.setBounds(this.screenWidth/64, this.screenHeight/64, (int
            )(this.screenWidth* 0.9), (int)(this.screenHeight* 0.9));
        frmResult.setResizable(false);
```

```
frmResult.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frmResult.getContentPane().setLayout(null);
frmResult.setTitle("Result of the algorithm");

/* TAB */
JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
tabbedPane.setBorder(null);
tabbedPane.setBounds(0, 0, (int)(this.screenWidth* 0.895), (int)(
   this.screenHeight* 0.87));
tabbedPane.setFont(new Font("Consolas", Font.PLAIN, 16));

/* TAB PAPER*/
JPanel panelPaper = new JPanel(null);
tabbedPane.addTab("PAPER", null, panelPaper, null);

// Graph Display
GraphModel modelPaper = new DefaultGraphModel();

DefaultGraphCell[] cellsPaper = Graph(this.itineraryPaper);

JGraph graphPaper = new JGraph(modelPaper);
graphPaper.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent arg0) {
        graphPaper.refresh();
        graphPaper.revalidate();
        graphPaper.repaint();
    }
});

// Insert the cells via the cache, so they get selected
graphPaper.getGraphLayoutCache().insert(cellsPaper);

graphResultPaper = new JScrollPane(graphPaper);
graphResultPaper.setHorizontalScrollBarPolicy(JScrollPane.
   HORIZONTAL_SCROLLBAR_AS_NEEDED);
graphResultPaper.setVerticalScrollBarPolicy(JScrollPane.
   VERTICAL_SCROLLBAR_AS_NEEDED);
graphResultPaper.setBounds(0, 0, (int)(this.screenWidth* 0.89), (
   int)(this.screenHeight* 0.71));
panelPaper.add(graphResultPaper);

statistics = new JLabel();
statistics.setFont(new Font("Consolas", Font.PLAIN, 14));
statistics.setText("<html>Paper trash itenerary : <br>" +
   printItinerary(this.itineraryPaper) + "<br><br>Statistics :" + "
   <br>Distance Covered - " + String.valueOf(this.
   paper_distanceCovered) + "</html>");
statistics.setBounds((int)(this.screenWidth* 0.01), (int)(this.
   screenHeight* 0.7), (int)(this.screenWidth* 0.5), (int)(this.
   screenHeight* 0.15));
panelPaper.add(statistics);

/* TAB PLASTIC*/
JPanel panelPlastic = new JPanel(null);
tabbedPane.addTab("PLASTIC", null, panelPlastic, null);

// Graph Display
GraphModel modelPlastic = new DefaultGraphModel();

DefaultGraphCell[] cellsPlastic = Graph(this.itineraryPlastic);

JGraph graphPlastic = new JGraph(modelPlastic);
graphPlastic.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent arg0) {
        graphPlastic.refresh();
        graphPlastic.revalidate();
        graphPlastic.repaint();
    }
});
```

```java
// Insert the cells via the cache, so they get selected
graphPlastic.getGraphLayoutCache().insert(cellsPlastic);

graphResultPlastic = new JScrollPane(graphPlastic);
graphResultPlastic.setHorizontalScrollBarPolicy(JScrollPane.
    HORIZONTAL_SCROLLBAR_AS_NEEDED);
graphResultPlastic.setVerticalScrollBarPolicy(JScrollPane.
    VERTICAL_SCROLLBAR_AS_NEEDED);
graphResultPlastic.setBounds(0, 0, (int)(this.screenWidth* 0.89), (
    int)(this.screenHeight* 0.71));
panelPlastic.add(graphResultPlastic);

statistics = new JLabel();
statistics.setFont(new Font("Consolas", Font.PLAIN, 14));
statistics.setText("<html>Plastic trash itenerary : <br>" +
    printItinerary(this.itineraryPlastic) + "<br><br>Statistics :" +
     "<br>Distance Covered - " + String.valueOf(this.
    plastic_distanceCovered) + "</html>");
statistics.setBounds((int)(this.screenWidth* 0.01), (int)(this.
    screenHeight* 0.7), (int)(this.screenWidth* 0.5), (int)(this.
    screenHeight* 0.15));
panelPlastic.add(statistics);

/* TAB GLASS*/
JPanel panelGlass = new JPanel(null);
tabbedPane.addTab("GLASS", null, panelGlass, null);

// Graph Display
GraphModel modelGlass = new DefaultGraphModel();

DefaultGraphCell[] cellsGlass = Graph(this.itineraryGlass);

JGraph graphGlass = new JGraph(modelGlass);
graphGlass.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent arg0) {
        graphGlass.refresh();
        graphGlass.revalidate();
        graphGlass.repaint();
    }
});

// Insert the cells via the cache, so they get selected
graphGlass.getGraphLayoutCache().insert(cellsGlass);

graphResultGlass = new JScrollPane(graphGlass);
graphResultGlass.setHorizontalScrollBarPolicy(JScrollPane.
    HORIZONTAL_SCROLLBAR_AS_NEEDED);
graphResultGlass.setVerticalScrollBarPolicy(JScrollPane.
    VERTICAL_SCROLLBAR_AS_NEEDED);
graphResultGlass.setBounds(0, 0, (int)(this.screenWidth* 0.89), (
    int)(this.screenHeight* 0.71));
panelGlass.add(graphResultGlass);

statistics = new JLabel();
statistics.setFont(new Font("Consolas", Font.PLAIN, 14));
statistics.setText("<html>Glass trash itenerary : <br>" +
    printItinerary(this.itineraryGlass) + "<br><br>Statistics :" + "
    <br>Distance Covered - " + String.valueOf(this.
    glass_distanceCovered) + "</html>");
statistics.setBounds((int)(this.screenWidth* 0.01), (int)(this.
    screenHeight* 0.7), (int)(this.screenWidth* 0.5), (int)(this.
    screenHeight* 0.15));
panelGlass.add(statistics);

/* TAB COMMON*/
JPanel panelCommon = new JPanel(null);
tabbedPane.addTab("COMMON", null, panelCommon, null);
```

```
        // Graph Display
        GraphModel modelCommon = new DefaultGraphModel();

        DefaultGraphCell[] cellsCommon = Graph(this.itineraryCommon);

        JGraph graphCommon = new JGraph(modelCommon);
        graphCommon.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseReleased(MouseEvent arg0) {
                graphCommon.refresh();
                graphCommon.revalidate();
                graphCommon.repaint();
            }
        });

        // Insert the cells via the cache, so they get selected
        graphCommon.getGraphLayoutCache().insert(cellsCommon);

        graphResultCommon = new JScrollPane(graphCommon);
        graphResultCommon.setHorizontalScrollBarPolicy(JScrollPane.
            HORIZONTAL_SCROLLBAR_AS_NEEDED);
        graphResultCommon.setVerticalScrollBarPolicy(JScrollPane.
            VERTICAL_SCROLLBAR_AS_NEEDED);
        graphResultCommon.setBounds(0, 0, (int)(this.screenWidth* 0.89), (
            int)(this.screenHeight* 0.71));
        panelCommon.add(graphResultCommon);

        statistics = new JLabel();
        statistics.setFont(new Font("Consolas", Font.PLAIN, 14));
        statistics.setText("<html>Common trash itenerary : <br>" +
            printItinerary(this.itineraryCommon) + "<br><br>Statistics :" +
            "<br>Distance Covered - " + String.valueOf(this.
            common_distanceCovered) + "</html>");
        statistics.setBounds((int)(this.screenWidth* 0.01), (int)(this.
            screenHeight* 0.7), (int)(this.screenWidth* 0.5), (int)(this.
            screenHeight* 0.15));
        panelCommon.add(statistics);
        frmResult.getContentPane().add(tabbedPane);

    }

    private DefaultGraphCell[] Graph(List<Node> itinerary) {
        // Count elements
        int counter = 0;
        for (Node node : ProgramData.graph.getNodes()) {
            counter++;
            counter += node.getOutEdges().size();
        }

        // Insert all three cells in one call, so we need an array to store
             them
        DefaultGraphCell[] cells = new DefaultGraphCell[counter];

        // Nodes and edges
        int i = 0, x = 20, j = 0, k = 0, y = 20;

        for (Node node : ProgramData.graph.getNodes()) {
            if (itinerary.contains(node))
                cells[i] = createVertex(node.getName(), x, y, 150, 40,
                    Color.BLUE, false);
            else
                cells[i] = createVertex(node.getName(), x, y, 150, 40, null
                    , false);

            x += 250;
            if (i % 2.0 == 0.0 && i != 0) {
                y += 150;
                x = 180;
            }
            i++;
        }
```

```java
        for (Node node : ProgramData.graph.getNodes()) {
            k++;
            for (Edge edgeo : node.getOutEdges()) {
                DefaultEdge edge = new DefaultEdge(edgeo.getDistance());

                edge.setSource(cells[k - 1]);

                j = 0;
                Node target = null;
                for (Node nodeaux : ProgramData.graph.getNodes()) {
                    if (nodeaux.equals(edgeo.getDestiny())) {
                        edge.setTarget(cells[j]);
                        target = nodeaux;
                    }
                    j++;
                }

                if (itinerary.contains(node) && itinerary.contains(target))
                    GraphConstants.setLineColor(edge.getAttributes(), Color
                        .BLUE);

                GraphConstants.setEndFill(edge.getAttributes(), true);
                cells[i] = edge;
                i++;
            }
        }
        return cells;
    }

    public static DefaultGraphCell createVertex(String name, double x,
       double y, double w, double h, Color bg,
            boolean raised) {

        // Create vertex with the given name
        DefaultGraphCell cell = new DefaultGraphCell(name);

        // Set bounds
        GraphConstants.setBounds(cell.getAttributes(), new Rectangle2D.
           Double(x, y, w, h));

        // Set fill color
        if (bg != null) {
            GraphConstants.setGradientColor(cell.getAttributes(), bg);
            GraphConstants.setOpaque(cell.getAttributes(), true);
        }

        // Set raised border
        if (raised) {
            GraphConstants.setBorder(cell.getAttributes(), BorderFactory.
                createRaisedBevelBorder());
        } else // Set black border
        {
            GraphConstants.setBorderColor(cell.getAttributes(), Color.black
                );
        }
        // Add a Floating Port
        cell.addPort();

        return cell;
    }

    public String printItinerary(List<Node> itinerary) {

        String string = "";
        int i = 1;
        for (Node node : itinerary) {
            if (i == (itinerary.size()))
                string += i + " - " + (node.getName());
            else
                string += i + " - " + (node.getName() + ", ");
            i++;
        }
        return string;

    }
}
```

A.7.3 Package logic

```
package logic;

import graph.Graph;
import graph.Node;

import java.util.HashMap;

public class AStarNode {

    private Graph graph;
    private double g;
    private double h;
    private Node node;
    private AStarNode parent;
    private Truck truck;

    public AStarNode(Graph graph, Node node, Truck truck) {
        this.graph = new Graph(graph);

        this.truck = new Truck(truck);
        for (Node e : this.graph.getNodes())
            if (e.getId() == node.getId())
                this.node = e;

        this.truck.truckCollect(this.node);
        HashMap<Integer, Node> temp = this.graph.getGraphContainers();
        temp.replace(this.node.getId(), this.node);
        this.graph.setGraphContainers(temp);
    }

    public void setTruck(Truck t) {
        this.truck = t;
    }

    public void setGraph(Graph g) {
        this.graph = g;
    }

    public Graph getGraph() {
        return graph;
    }

    public Node getNode() {
        return this.node;
    }

    public double getG() {
        return g;
    }

    public void setG(double g) {
        this.g = g;
    }

    public double getH() {
        return h;
    }

    public void setH(double h) {
        this.h = h;
    }

    public int compareTo(AStarNode obj) {
        return Double.compare(this.getG() + this.getH(), obj.getG() + obj.
            getH());
    }

    public AStarNode getParent() {
        return this.parent;
    }

    public void setParent(AStarNode parent2) {
        this.parent = parent2;

    }

    @Override
    public String toString() {
        String str = this.getNode().getId() + " - G: " + this.getG() + " -
            H: " + this.getH() + "    - Colected/Total - " + this.getTruck().
            allWasteSinceStart + "/" + Search.graph.
            getTotalGarbageByTypeWaste(this.getTruck().getType());
        return str;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj == null)
            return false;

        AStarNode aStarNode = (AStarNode) obj;

        if (this.g != aStarNode.g) {
            return false;

        }

        if (this.getTruck().getTotalGarbage() != aStarNode.getTruck().
            getTotalGarbage())
            return false;

        if (this.getTruck().getTotalGarbageSinceInit() != aStarNode.
            getTruck().getTotalGarbageSinceInit())
            return false;

        if (this.getGraph().getTotalGarbageByTypeWaste(this.getTruck().
            getType()) != aStarNode.getGraph().getTotalGarbageByTypeWaste(
            this.getTruck().getType()))
            return false;

        if(!this.getNode().equals(aStarNode.getNode()))
            return  false;

            /*

            return false;
*/
/*
        System.out.println("G : \n" + this.g + " " + aStarNode.g);
        System.out.println("Total Garbage \n" + this.getTruck().
            getTotalGarbageSinceInit() + " " + aStarNode.getTruck().
            getTotalGarbageSinceInit());
        System.out.println("Garbage Since Start : \n" + this.getTruck().
            getTotalGarbageSinceInit() + " " + aStarNode.getTruck().
            getTotalGarbageSinceInit());
*/


        return true;
    }
    public boolean hasFinish(String typeofWaste) {
        if (this.getTruck().allWasteSinceStart == Search.graph.
            getTotalGarbageByTypeWasteWithMinimumLevelInContainers(
            typeofWaste))
             if (this.getTruck().getTotalGarbage() == 0.0)
                 return true;
        return false;
    }

    public Truck getTruck() {
        return truck;
    }
}
```

```
package logic;

import java.util.Comparator;

public class AStarNodeComparator implements Comparator<AStarNode> {

    public int compare(AStarNode arg0, AStarNode arg1) {
            return arg0.compareTo(arg1);

    }
}
```

```
package logic;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import graph.Graph;
import graph.Node;

public class BuildGraph {

    protected HashMap<Integer, Node> containers;
    protected ArrayList<Node> garbageStations;
    protected Map<String, Integer> typeTruck;
    protected Node central;
    protected Node station;
    protected Map<String, ArrayList<Truck>> trucks;

    public BuildGraph(Graph graph, int truckPlastic, int truckPaper, int
       truckGlass, int truckCommon,
            int numberOfStations, double truckCapacity, double
               minimumLevelContainer, String heuristic) {

        setContainersAndStations(graph);
        setMapTrucks(truckPlastic, truckPaper, truckGlass, truckCommon);
        setTrucks(truckCapacity);
        new Search(graph, containers, garbageStations, typeTruck, central,
            station, trucks, heuristic);


    }

    private Map<String, Integer> setMapTrucks(int truckPlastic, int
       truckPaper, int truckGlass, int truckCommon) {
        typeTruck = new HashMap<String, Integer>();
        typeTruck.put(Utils.GLASS, truckGlass);
        typeTruck.put(Utils.PAPER, truckPaper);
        typeTruck.put(Utils.PLASTIC, truckPlastic);
        typeTruck.put(Utils.COMMON, truckCommon);
        return typeTruck;
    }

    public ArrayList<Node> getGarbageStation() {
        return this.garbageStations;
    }

    public void setContainersAndStations(Graph graph) {
        this.containers = new HashMap<Integer, Node>();
        this.garbageStations = new ArrayList<Node>();

        for (Node temp : graph.getNodes()) {
            if (temp.getType().equals(Utils.STATION)) {
                station = temp;
                station.setOutEdges(temp.getOutEdges());
                addGarbageStation(temp);
            } else if (temp.getType().equals(Utils.TRUE_GARBAGE)) {
                addGarbageContainer(temp);
            } else if (temp.getType().equals(Utils.CENTRAL)) {
                central = temp;
                central.setOutEdges(temp.getOutEdges());
            }
        }
        graph.setGraphContainers(this.containers);
    }

    public Map<String, ArrayList<Truck>> getTrucks() {
        return this.trucks;
    }

    public void setTrucks(double capacity) {
        Truck truck;
        ArrayList<Truck> truckTemp = new ArrayList<Truck>();
        int i;
        this.trucks = new HashMap<String, ArrayList<Truck>>();

        for (i = 0; i < typeTruck.get(Utils.GLASS); i++) {
            truck = new Truck(capacity, Utils.GLASS);
            truckTemp.add(truck);
        }
        if (truckTemp.size() > 0)
            this.trucks.put(Utils.GLASS, truckTemp);


        truckTemp = new ArrayList<Truck>();
        for (i = 0; i < typeTruck.get(Utils.PLASTIC); i++) {
            truck = new Truck(capacity, Utils.PLASTIC);
            truckTemp.add(truck);
        }
        if (truckTemp.size() > 0)
            this.trucks.put(Utils.PLASTIC, truckTemp);

        truckTemp = new ArrayList<Truck>();
        for (i = 0; i < typeTruck.get(Utils.PAPER); i++) {
            truck = new Truck(capacity, Utils.PAPER);
            truckTemp.add(truck);
        }
        if (truckTemp.size() > 0)
            this.trucks.put(Utils.PAPER, truckTemp);

        truckTemp = new ArrayList<Truck>();
        for (i = 0; i < typeTruck.get(Utils.COMMON); i++) {
            truck = new Truck(capacity, Utils.COMMON);
            truckTemp.add(truck);
        }
        if (truckTemp.size() > 0)
            this.trucks.put(Utils.COMMON, truckTemp);
    }

    public void addGarbageContainer(Node node) {
        this.containers.put(node.getId(), node);
    }

    public void addGarbageStation(Node node) {
        this.garbageStations.add(node);
    }
}
```

```
package logic;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;

import graph.Graph;
import graph.Node;

public class ProgramData {

    public static Graph graph;
    protected int truckPlastic;
    protected int truckPaper;
    protected int truckGlass;
    protected int truckCommon;
    protected int numberOfStations;
    protected int truckCapacity;
    protected double minimumLevelContainer;
    protected String heuristic;

    public ProgramData(int truckCapacity, int numberOfStations, double
        minimumLevelContainer, int truckPlastic,
            int truckPaper, int truckGlass, int truckCommon, String
                heuristic, File file) {

        this.truckPlastic = truckPlastic;
        this.truckPaper = truckPaper;
        this.truckGlass = truckGlass;
        this.truckCommon = truckCommon;
        this.numberOfStations = numberOfStations;
        this.truckCapacity = truckCapacity;
        this.minimumLevelContainer = minimumLevelContainer;
        this.heuristic = heuristic;

        // displayInformation();
        try {
            ProgramData.graph = loadMap(file);
        } catch (IOException e) {
            System.out.println("Unable to load csv file");
            e.printStackTrace();
        }
        // printGraph();

        new BuildGraph(ProgramData.graph, this.truckPlastic, this.
            truckPaper, this.truckGlass, this.truckCommon,
                this.numberOfStations, this.truckCapacity, this.
                    minimumLevelContainer, this.heuristic);
    }

    public void displayInformation() {
        System.out.println(this);
    }

    @Override
    public String toString() {
        int totalTrucks = this.truckCommon + this.truckGlass + this.
            truckPaper + this.truckPlastic;
        String str = "Number of plastic trucks: " + this.truckPlastic + "\n
            " + "Number of glass trucks: "
                + this.truckGlass + "\n" + "Number of paper trucks: " +
                    this.truckPaper + "\n"
                + "Number of common trucks: " + this.truckCommon + "\n" + "
                    Number of total trucks: " + totalTrucks
                + "\n" + "Minimum Level of each container: " + this.
                    minimumLevelContainer + "\n" + "Truck capacity: "
                + this.truckCapacity + "\n" + "Heuristic selected: " + this
                    .heuristic + "\n";
        return str;
    }

    public Graph loadMap(File file) throws IOException {

        Graph graph = new Graph();
        FileInputStream fstream = new FileInputStream(file);
        BufferedReader br = new BufferedReader(new InputStreamReader(
            fstream));
        String readMode = Utils.UNDEFINED;
        String fileLine;

        while ((fileLine = br.readLine()) != null) {

            // Check non empty line
            if (fileLine.length() > 0) {

                // Checks if is a node or edge line
                if (fileLine.contains(Utils.NODES)) {
                    readMode = Utils.NODES;
                    continue;
                } else if (fileLine.contains(Utils.EDGES)) {
                    readMode = Utils.EDGES;
                    continue;
                }

                switch (readMode) {

                case Utils.NODES:
                    String[] nodeValues = fileLine.split(Utils.SPLITTER);
                    if (nodeValues.length != 7) {
                        break;
                    }

                    Node node = new Node(Integer.parseInt(nodeValues[0]),
                        // id
                            nodeValues[1], // type
                            nodeValues[2], // name of street
                            Double.parseDouble(nodeValues[3]), // glass
                                garbage
                            Double.parseDouble(nodeValues[4]), // paper
                                garbage
                            Double.parseDouble(nodeValues[5]), // plastic
                                garbage
                            Double.parseDouble(nodeValues[6]) // common
                                garbage
                            );
                    graph.addNode(node);


                    break;

                case Utils.EDGES:

                    String[] edgeValues = fileLine.split(Utils.SPLITTER);

                    if (edgeValues.length != 3) {
                        break;
                    }

                    Node source = graph.findNode(Integer.parseInt(
                        edgeValues[0]));
                    Node destiny = graph.findNode(Integer.parseInt(
                        edgeValues[1]));

                    if (source == null || destiny == null) {
                        System.out.println(source);
```

```java
                        System.out.println(destiny);
                        throw new IOException();

                    }

                    double distance = Double.parseDouble(edgeValues[2]);

                    // double side because the file has only one -
                        undirected graph

                    source.addEdge(destiny, distance);
                    destiny.addEdge(source, distance);

                    break;
                default:
                    readMode = Utils.UNDEFINED;
                    break;
                }
            }
        }
        // Close the input stream
        br.close();
        return graph;
    }

    public void printGraph() {
        System.out.println(ProgramData.graph);
    }
}
```

```java
package logic;

import java.util.*;

import graph.Edge;
import graph.Graph;
import graph.Node;
import gui.Result;

public class Search {

    protected HashMap<Integer, Node> containers;
    protected ArrayList<Node> garbageStations;
    public static ArrayList<Node> itinerary = new ArrayList<Node>();
    protected Map<String, Integer> typeTruck;
    protected Node central;
    protected Node station;
    protected Map<String, ArrayList<Truck>> trucks;
    protected static Graph graph;
    private double distanceCovered;
    private Truck truck;

    public Search(Graph graph, HashMap<Integer, Node> containers, ArrayList
        <Node> garbageStations,
                  Map<String, Integer> typeTruck, Node central, Node
                      station, Map<String, ArrayList<Truck>> trucks,
                  String heuristic) {

        this.containers = containers;
        this.garbageStations = garbageStations;
        this.central = central;
        this.station = station;
        Search.graph = graph;
        this.trucks = trucks;

        switch (heuristic) {
            case Utils.HEURISTIC1:
                computeProgram(heuristic);
                break;
            case Utils.HEURISTIC2:
                computeProgram(heuristic);
                break;
            case Utils.HEURISTIC3:
                computeProgram(heuristic);
                break;
            default:
                break;
        }

        // show in gui the result

        sendSearchToResult();
    }

    private void computeProgram(String heuristic) {

        truck = this.trucks.get(Utils.PAPER).get(0);

        ArrayList<Object> paperResult = searchAStar(heuristic, Utils.PAPER)
          ;
        showresults(paperResult, Utils.PAPER);

        truck = this.trucks.get(Utils.GLASS).get(0);
        ArrayList<Object> glassResult = searchAStar(heuristic, Utils.GLASS)
          ;
        showresults(glassResult, Utils.GLASS);

        truck = this.trucks.get(Utils.PLASTIC).get(0);
        ArrayList<Object> plasticResult = searchAStar(heuristic, Utils.
          PLASTIC);
        showresults(plasticResult, Utils.PLASTIC);

        truck = this.trucks.get(Utils.COMMON).get(0);
        ArrayList<Object> commonResult = searchAStar(heuristic, Utils.
          COMMON);
        showresults(commonResult, Utils.COMMON);

        showTotalResults(paperResult, glassResult, plasticResult,
          commonResult);
    }

    public void showresults(ArrayList<Object> result, String typeofWaste) {
        if (result != null) {
            System.out.println("Best solution found for " + typeofWaste + "
                 - Statistics");
            long timePassed = (long) result.get(1);
            System.out.println("Time of execution: " + timePassed + "ms");
            AStarNode finalNode = (AStarNode) result.get(0);

            System.out.println("Number of visited nodes = " + result.get(2)
              );
            System.out.println("Total Cost: " + finalNode.getG() + "km");
            System.out.println("Total Garbage to Collect: " + finalNode.
              getGraph().getTotalGarbageByTypeWaste(typeofWaste));
            System.out.println("Total Garbage Collected: " + finalNode.
              getTruck().allWasteSinceStart);
            System.out.println("Initial Garbage to Collect: " + graph.
              getTotalGarbageByTypeWaste(typeofWaste) + "\n");
            printResult(finalNode);
            System.out.println("
              ==================================================================
              n");
        } else {
            System.out.println("No solution found!");
        }
    }

    private void showTotalResults(ArrayList<Object> paperResult, ArrayList<
      Object> glassResult, ArrayList<Object> plasticResult, ArrayList<
      Object> commonResult) {
        System.out.println("Total Statistics");
        long timePassed = (long) paperResult.get(1) + (long) glassResult.
            get(1) + (long) plasticResult.get(1) + (long) commonResult.get
            (1);
        System.out.println("Time of execution: " + timePassed + "ms");
        AStarNode finalNodePaper = (AStarNode) paperResult.get(0);
        AStarNode finalNodeGlass = (AStarNode) glassResult.get(0);
        AStarNode finalNodePlastic = (AStarNode) plasticResult.get(0);
        AStarNode finalNodeCommon = (AStarNode) commonResult.get(0);
        int totalvisitednodes = (int) paperResult.get(2) + (int)
            glassResult.get(2) + (int) plasticResult.get(2) + (int)
            commonResult.get(2);
        System.out.println("Number of visited nodes = " + totalvisitednodes
            );
        double totalcost = finalNodePaper.getG() + finalNodeCommon.getG() +
             finalNodeGlass.getG() + finalNodePlastic.getG();
        System.out.println("Total Cost: " + totalcost + "km");
        double totalgarbagetocollect = finalNodePaper.getGraph().
            getTotalGarbageByTypeWaste(Utils.PAPER) + finalNodeCommon.
            getGraph().getTotalGarbageByTypeWaste(Utils.COMMON) +
                finalNodeGlass.getGraph().getTotalGarbageByTypeWaste(Utils.
                    GLASS) + finalNodePlastic.getGraph().
                    getTotalGarbageByTypeWaste(Utils.PLASTIC);
        System.out.println("Total Garbage to Collect: " +
            totalgarbagetocollect);
        double totaltruckcollect = finalNodePaper.getTruck().
            allWasteSinceStart + finalNodeCommon.getTruck().
            allWasteSinceStart +
                finalNodeGlass.getTruck().allWasteSinceStart +
                    finalNodePlastic.getTruck().allWasteSinceStart;
        System.out.println("Total Garbage Collected: " + totaltruckcollect)
            ;
        double totalInitialGarbage = graph.getTotalGarbageByTypeWaste(Utils
            .PAPER) +
                graph.getTotalGarbageByTypeWaste(Utils.PLASTIC) +
                graph.getTotalGarbageByTypeWaste(Utils.COMMON) +
                graph.getTotalGarbageByTypeWaste(Utils.GLASS);
        System.out.println("Initial Garbage to Collect: " +
            totalInitialGarbage + "\n");
        System.out.println("
            ==========================================================================
            n");


    }

    private void printResult(AStarNode result) {
        Stack<AStarNode> stack = new Stack<AStarNode>();
        stack.add(result);

        AStarNode parent = result.getParent();
        while (parent != null) {
            stack.push(parent);
            parent = parent.getParent();
        }
        System.out.println("ID node - Total to Collect / Current Truck
            Collected\n");
        int count = 1;
        while (stack.size() > 0) {
            itinerary.add(stack.peek().getNode());
            if (count % 6 == 0)
                System.out.println(stack.peek().getNode().getId() + " - " +
                     stack.peek().getTruck().allWasteSinceStart + "/" +
                    stack.peek().getTruck().getTotalGarbage());
            else
                System.out.print(stack.peek().getNode().getId() + " - " +
                    stack.peek().getTruck().allWasteSinceStart + "/" + stack
                    .peek().getTruck().getTotalGarbage() + "  ->  ");

            count++;
            stack.pop();
        }
        System.out.println("");
    }

    public ArrayList<Object> searchAStar(String heuristic, String
        typeofWaste) {
        ArrayList<Object> result = new ArrayList<Object>(); // to get the
            result of search
        // Number of visited nodes
        int visitedNodes = 0;

        // Initialize open and closed lists
        ArrayList<AStarNode> open = new ArrayList<AStarNode>();
        ArrayList<AStarNode> closed = new ArrayList<AStarNode>();
        PriorityQueue<AStarNode> queue = new PriorityQueue<AStarNode>(graph
            .getNumNodes(),new AStarNodeComparator());
        AStarNode initial = new AStarNode(graph, central, truck);
        initial.setG(0);
        initial.setH(heuristic_cost_estimate(initial, heuristic,
            typeofWaste));

        // Add it to the open list

        open.add(initial);
        queue.add(initial);
        long startTime = System.currentTimeMillis();
        AStarNode lowF = null;

        // Loop the open list as long as it isn't empty
        while (!open.isEmpty()) {
            // Increment number of visited nodes
            visitedNodes++;

            // Get the node with the lowest f value
            //lowF = lowestF(open);
          lowF = queue.poll();
            //System.out.println(lowF);

            // Check if it is the goal
            if (lowF.hasFinish(typeofWaste)) {
                long stopTime = System.currentTimeMillis();
                long elapsedTime = stopTime - startTime;

                result.add(lowF);
                result.add(elapsedTime);
                result.add(visitedNodes);
                return result;
            }

            // Add it to the closed list and remove it from the open list
            closed.add(lowF);
            open.remove(lowF);

            // Get the adjacent nodes
            ArrayList<AStarNode> adj = getAdjacentNodes(lowF);


            // Check each adjacent node not on the closed list
            for (int i = 0; i < adj.size(); i++) {
                if (!closed.contains(adj.get(i))) {
                    // Set this node's f value
                    adj.get(i).setH(heuristic_cost_estimate(adj.get(i),
                        heuristic, typeofWaste));

                    // Check if it is on the open list
                    if (!open.contains(adj.get(i))) {
                        // Add it if it isn't
                        open.add(adj.get(i));
                        queue.add(adj.get(i));
                    } else {
                        // Get the one on the open list
                        AStarNode temp = open.get(open.indexOf(adj.get(i)))
                           ;

                        // Check which one has the lowest g value
                        if (adj.get(i).getG() < temp.getG()) {

                            temp.setG(adj.get(i).getG());
                            temp.setParent(adj.get(i).getParent());
                        }
                    }
                }
            }
        }
        return null;
    }

    private double heuristic_cost_estimate(AStarNode aStarNode, String
       heuristic, String typeofWaste) {
        double h = 0.0;
        // Check the chosen heuristic
        if (heuristic == Utils.HEURISTIC1) {
            if (aStarNode.getNode().getType().equals(Utils.TRUE_GARBAGE))
                h = aStarNode.getGraph().getTotalGarbageByTypeWaste(
                   typeofWaste) - aStarNode.getNode().
                   getGarbageContainerByType(typeofWaste);//
                   getTotalGarbageByTypeWasteWithMinimumLevelInContainers(
                   Utils.PAPER);// - aStarNode.getNode().
                   getGarbageContainerByType(Utils.PAPER);
            else
                h = aStarNode.getGraph().getTotalGarbageByTypeWaste(
                   typeofWaste);//
                   getTotalGarbageByTypeWasteWithMinimumLevelInContainers(
                   Utils.PAPER);
        } else if (heuristic == Utils.HEURISTIC2) {
            return h;
        } else if (heuristic == Utils.HEURISTIC3) {
            System.out.println((aStarNode.getGraph().getGraphContainers().
               size() - aStarNode.getGraph().getNumberOfEmptyContainers(
               typeofWaste)) * 10);
            if (aStarNode.getNode().getType().equals(Utils.TRUE_GARBAGE))
                h = (aStarNode.getGraph().getGraphContainers().size() -
                   aStarNode.getGraph().getNumberOfEmptyContainers(
                   typeofWaste)) * 10;
            else h = aStarNode.getGraph().getTotalGarbageByTypeWaste(
               typeofWaste) * 10;
        }
        return h;
    }

    public void sendSearchToResult() {
        try {
            Result window = new Result(Search.graph, Search.itinerary,
               distanceCovered);
            window.frmResult.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private AStarNode lowestF(ArrayList<AStarNode> open) {
        AStarNode temp = null;

        for (int i = 0; i < open.size(); i++) {
            if (temp == null)
                temp = open.get(i);
            else if ((open.get(i).getH() + open.get(i).getG()) < (temp.getH
               () + temp.getG()))
                temp = open.get(i);
        }
        return temp;
    }

    private ArrayList<AStarNode> getAdjacentNodes(AStarNode curr) {
        ArrayList<AStarNode> adjacents = new ArrayList<AStarNode>();


        for (Edge e : curr.getNode().getOutEdges()) {

            Node neighbor = e.getDestiny();
            double cost = e.getDistance();
            double temp_g_scores = curr.getG() + cost;

            AStarNode aux = new AStarNode(curr.getGraph(), neighbor, curr.
                getTruck());
            aux.setG(temp_g_scores);
            aux.setParent(curr);

            // Add it
            adjacents.add(aux);
        }
        return adjacents;
    }

    Graph getGraph() {
        return Search.graph;
    }
}
```

```
package logic;


import graph.Node;

public class Truck {

    private final double capacity; // Kilograms
    private Node startingPosition; // Starting node
    private Node destinyPosition; // Goal node
    private final String type; // paper, plastic, glass, common
    private double distanceCovered; // Kilometers
    private double totalGarbage; // garbage collected
    protected static int current_id = 0;
    protected int id;
    protected AStarNode itinerary = null;
    protected double allWasteSinceStart = 0.0;

    public Truck(double capacity, String type) {
        this.capacity = capacity;
        this.type = new String(type);
        this.distanceCovered = 0.0;
        this.totalGarbage = 0.0;

        ++current_id;
        this.id = current_id;
    }

    public Truck(Truck truck) {
        this.capacity = truck.capacity;
        this.startingPosition = truck.startingPosition;
        this.destinyPosition = truck.destinyPosition;
        this.type = truck.type;
        this.distanceCovered = truck.distanceCovered;
        this.totalGarbage = truck.totalGarbage;
        this.id = truck.id;
        this.allWasteSinceStart = truck.allWasteSinceStart;

    }

    public Node getStartingPosition() {
        return startingPosition;
    }

    public double getDistanceCovered() {
        return distanceCovered;
    }

    public void setStartingPosition(Node startPos) {
        this.startingPosition = startPos;
    }

    public Node getDestinyPosition() {
        return destinyPosition;
    }

    public void setDestinyPosition(Node destinyPos) {
        this.destinyPosition = destinyPos;
    }

    public void addToDistanceCovered(double distance) {
        this.distanceCovered += distance;
    }

    public void resetDistanceCovered() {
        this.distanceCovered = 0.0;
    }

    public double getCapacity() {
        return capacity;
    }

    public boolean isFull() {
        return this.totalGarbage > this.capacity;
    }

    public String getType() {
        return type;
    }

    public int getID() {
        return id;
    }

    public void setTotalGarbage(double garbage) {
        this.totalGarbage += garbage;
    }

    public void resetTotalGarbage() {
        this.totalGarbage = 0.0;
    }

    public double getTotalGarbage() {
        return totalGarbage;
    }

    public AStarNode getItinerary(){
        return this.itinerary;
    }

    public void setItinerary(AStarNode node){
        this.itinerary = node;
    }

    @Override
    public String toString() {
        String str = this.getID() + "-" + this.getType() + ": " + this.
            distanceCovered;// + " " + this.startingPosition.getName() + "
            -> " + this.destinyPosition.getName();
        return str;
    }
    public void collectWaste(){
        truckCollect(itinerary.getNode());
        AStarNode parent = itinerary.getParent();

        while (parent != null) {
            truckCollect(parent.getNode());
            parent = parent.getParent();
        }
    }

    public double truckCollect(Node node) {
        if(node.getType().equals(Utils.TRUE_GARBAGE)){ // se for contentor
           de lixo
            double actualPaperToCollected = node.getGarbageContainerByType(
                this.type);
            if(actualPaperToCollected > Utils.MinimumGarbageCapacity){ //
                se houver papel por apanhar
                if(this.getType().equals(this.type) && (this.
                    getTotalGarbage()+actualPaperToCollected) <= this.
                    getCapacity()){
                    this.setTotalGarbage(actualPaperToCollected);
                    this.allWasteSinceStart += actualPaperToCollected;
                    node.setGarbageContainer(this.type,
                        actualPaperToCollected); // apanha o papel
                    return actualPaperToCollected;
                }
                else if(this.getType().equals(this.type) && (this.
                    getTotalGarbage()+actualPaperToCollected) > this.
                    getCapacity()){
                    double currentLoadGarbage = this.getCapacity() - this.
                        getTotalGarbage();
                    this.setTotalGarbage(currentLoadGarbage);
                    this.allWasteSinceStart += currentLoadGarbage;
                    node.setGarbageContainer(this.type, currentLoadGarbage)
                        ; // apanha o papel
                    return currentLoadGarbage;
                }
            }
        }

        else if(node.getType().equals(Utils.STATION) && this.
           getTotalGarbage() > 0.0){ // se for estacao de tratamento,
           esvazia
            this.resetTotalGarbage();
        }
        return 0.0;
    }

    public void printItinerary(){
            System.out.println(itinerary);
    }

    public double getTotalGarbageSinceInit() {
        return this.allWasteSinceStart;
    }


}
```

```
package logic;

import java.io.File;

public class Utils {

    public static final String UNDEFINED = "UNDEFINED";
    public static final String NODES = "[nodes]";
    public static final String EDGES = "[edges]";
    public static final String CENTRAL = "central";
    public static final String STATION = "station";
    public static final String TRUE_GARBAGE = "true";
    public static final String FALSE_GARBAGE = "false";
    public static final String GLASS = "glass";
    public static final String PLASTIC = "plastic";
    public static final String PAPER = "paper";
    public static final String COMMON = "common";
    public static final String SPLITTER = ";";
    public static final File graphFile = new File("./resources/graphs");
    public static final String HEURISTIC1 = "heuristic1";
    public static final String HEURISTIC2 = "uniform cost";
    public static final String HEURISTIC3 = "heuristic3";
    public static int MinimumGarbageCapacity;
    public static final int garbageCapacity = 100;
    public static final String defaultFileGraph = "./resources/graphs/
        medium_graph.csv";
}
```

A.7.4 Package tests

```
package tests;

import graph.Graph;
import graph.Node;
import logic.Truck;
import org.junit.Test;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestApp {

    /*@Test
    public void testStraightLineDistance() {
        assertEquals(Search.straightLineDistance(38.898556, -77.037852,
            38.897147, -77.043934), 0.549, 0.001);

        assertEquals(Search.straightLineDistance(41.0961655, -8.7248637,
            39.4078969, -0.4315509), 728.321, 0.001);
    }*/

    @Test
    public void testTruckIDs() {
        Truck truck1 = new Truck(1000, "paper");
        Truck truck2 = new Truck(1000, "common");
        Truck truck3 = new Truck(1000, "paper");
        Truck truck4 = new Truck(1000, "glass");
        assertEquals(1, truck1.getID());
        assertEquals(2, truck2.getID());
        assertEquals(3, truck3.getID());
        assertEquals(4, truck4.getID());
    }

    @Test
    public void testDistanceBetweenNodes() {
        // Node node1 = new Node
    }

    @Test // go from n1 to n13
    public void testAlgotithmAstar() {

        Node n1 = new Node(1, "Arad", 366);
        Node n2 = new Node(2, "Zerind", 374);
        Node n3 = new Node(3, "Oradea", 380);
        Node n4 = new Node(4, "Sibiu", 253);
        Node n5 = new Node(5, "Fagaras", 178);
        Node n6 = new Node(6, "Rimnicu Vilcea", 193);
        Node n7 = new Node(7, "Pitesti", 98);
        Node n8 = new Node(8, "Timisoara", 329);
        Node n9 = new Node(9, "Lugoj", 244);
        Node n10 = new Node(10, "Mehadia", 241);
        Node n11 = new Node(11, "Drobeta", 242);
        Node n12 = new Node(12, "Craiova", 160);
        Node n13 = new Node(13, "Bucharest", 0);
        Node n14 = new Node(14, "Giurgiu", 77);

        // Arad
        n1.addEdge(n2, 75);
        n1.addEdge(n4, 140);
        n1.addEdge(n8, 118);

        // Zerind
        n2.addEdge(n1, 75);
        n2.addEdge(n3, 71);

        // Oradea
        n3.addEdge(n2, 71);
        n3.addEdge(n4, 151);

        // Sibiu
        n4.addEdge(n1, 140);
        n4.addEdge(n5, 99);
        n4.addEdge(n3, 151);
        n4.addEdge(n6, 80);

        // Fagaras
        n5.addEdge(n4, 99);
        n5.addEdge(n13, 211);

        // Rimnicu Vilcea
        n6.addEdge(n4, 80);
        n6.addEdge(n7, 97);
        n6.addEdge(n12, 146);

        // Pitesti
        n7.addEdge(n6, 97);
        n7.addEdge(n13, 101);
        n7.addEdge(n12, 138);

        // Timisoara
        n8.addEdge(n1, 118);
        n8.addEdge(n9, 111);

        // Lugoj
        n9.addEdge(n8, 111);
        n9.addEdge(n10, 70);

        // Mehadia
        n10.addEdge(n9, 70);
        n10.addEdge(n11, 75);

        // Drobeta
        n11.addEdge(n10, 75);
        n11.addEdge(n12, 120);

        // Craiova
        n12.addEdge(n11, 120);
        n12.addEdge(n6, 146);
        n12.addEdge(n7, 138);

        // Bucharest
        n13.addEdge(n7, 101);
        n13.addEdge(n14, 90);
        n13.addEdge(n5, 211);

        // Giurgiu
        n14.addEdge(n13, 90);
        assertArrayEquals(new int[] { 3, 2 }, new int[] { n1.getOutEdges().
            size(), n2.getOutEdges().size() });

        Graph graph = new Graph();
        graph.addNode(n1);
        graph.addNode(n2);
        graph.addNode(n3);
        graph.addNode(n4);
        graph.addNode(n5);
        graph.addNode(n6);
        graph.addNode(n7);
        graph.addNode(n8);
        graph.addNode(n9);
        graph.addNode(n10);
        graph.addNode(n11);
        graph.addNode(n12);
        graph.addNode(n13);
        graph.addNode(n14);

        assertEquals(14, graph.getNodes().size());



    }
};
```