ALGORITHMICS - Bachelor of Software Engineering, 2020

LAB GUIDE. SESSION 4

GOALS:

a) **Greedy algorithms**

1. Package delivery

A package delivery company knows the places where it must deliver a series of shipments and wants to optimize its resources to do it. To do this, they need to calculate the shortest possible path that allows the delivery car to leave the warehouse, go through each place of delivery (only once) and after completing all the planned places, return back to the origin.

To trace the route, you must consider: 1) we have paths between any pair of points/places; 2) the distance will be determined by positive integer numbers; and 3) the distance will be the same in both directions. Places will be identified by integer values starting from 0 (`v0`).

To solve it through a greedy algorithm, we will consider the following heuristics:

a) **Close node**. Let `(C, v)` be the road constructed until the moment that begins at place `v0` and ends at `v`. Initially `C` is empty and `v = v0`. If `C` contains all the nodes of the graph `G`, the algorithm includes the edge `(v, v0)` and ends. If not, it includes the edge `(v, w)` of minimum length between all the edges from `v` to the nodes `w` that are not yet in the path `C`.

b) **Short edge**. Another possible greedy algorithm would choose, in each iteration, the shortest edge not yet considered that meets the following two conditions:

- 1) Not to create a cycle with the edges already selected, except in the last iteration, which is where the trip is finished.
- 2) It is not the third edge that affects the same node among those already chosen.

The traveling salesman problem

The problem of the **Traveling Salesman Problem** (TSP) intends to find out the route that a seller carries out, who begins at a given origin and visits a certain set of cities, returning then to the origin, so that the total distance travelled is the minimal and each city is only visited once.

If one wishes to express the problem in a mathematical way using graph theory, the TSP would consist in finding, in a connected and weighted graph G, the Hamiltonian cycle such that the sum of the costs of the edges that make up this cycle is as small as possible.

The TSP is one of the most representative and best studied combinatorial optimization problems, but despite this, there are still no known efficient algorithms that exactly solve it. The algorithms used for resolution only provide approximations, and the optimal solution is only achievable for specific or relatively small cases of the problem.

It is classified as an NP-complete problem, which means that the computational effort that must be carried out to find an optimal solution grows exponentially with the input of the problem, which in the specific case of TSP would be the number of nodes or vertices of the network.

There is much information available on the problem on the Internet such as that shown on the Wikipedia Web: https://en.wikipedia.org/wiki/Travelling_salesman_problem and researchers from around the world are still looking for an algorithm that can solve it efficiently, trying also to include dynamic information such as traffic at any time depending on the area or the situation of roads in general, such as the information that appears in this article from a Massachusetts MIT magazine: https://www.technologyreview.es/s/8935/dentro-de-los-algoritmos-cada-vez-mas-complejos-que-llevan-los-paquetes-la-puerta-de-casa

TO DO:

A. Work to be done

- An Eclipse `session3` **package** in your course project. The content of the package should be:
  - Package `session3.salesman`: All the files that were given with the instructions for this session together with new classes/code:
    - `Salesman.java`. You are asked to create a Salesman program in Java that runs with the following syntax:

```
Traveling [HEURISTIC] [ [NUM_PLACES] [DIST_MAX] | [INPUT_FILE] ]
```

This program must be able to work with different parameters:

- `[HEURISTIC]` It is a common parameter to the two types of calls and indicates the heuristic to use to calculate the solution: `CloseNode` or `ShortEdge`.
- **Randomly generated data**. It will receive a parameter with the number of cities `NUM_PLACES` and will generate a random adjacency matrix for an undirected and connected graph, whose maximum distance between two places will be specified by the integer value `DIST_MAX`.
- **Data from a file**. It will receive the name of a text file `INPUT_FILE` as an input parameter with the following format:
  - The first line contains the number of places to go.
  - The following lines constitute the matrix of adjacencies with the costs of the edges between the pairs of places. Each line will contain as many integers as points separated by a tab. There will be as many lines as points.
  - When there is no path, a 0 will appear. This is a possible example of the content of the file:

```
7
0 12 43 99 57 32 78
12 0 10 80 93 33 11
43 10 0 60 43 20 22
99 80 60 0 50 18 31
57 93 43 50 0 31 73
32 33 20 18 31 0 22
78 11 22 31 73 22 0
```

The expected output from the program will be the sequence of places travelled and the total cost of the path/toad:

```
4 -> 5 -> 3 -> 6 -> 1 -> 2 -> 0 -> 4
Cost = 201
```

Use the unit tests (`SalesmanTest.java`) to check the correct operation of the program.

- `SalesmanTimes.java`. Class to calculate and compare the runtime and the results of the two alternatives for different sizes of the problem (with a loop increasing the size by 2 in each iteration).

- A **PDF document** using the course template. The activities of the document should be the following:
  - **Activity 1. Salesman**
    - Please indicate a brief explanation about the Salesman algorithm for each of the heuristics and their corresponding complexities.
  - **Activity 2. SalesmanTimes 1**
    - How do heuristics work regarding obtaining optimal solutions? Which one is better? Justify the answer based on the data obtained.
  - **Activity 3. SalesmanTimes 2**
    - Do the times obtained make sense if we compare them to the complexity of the algorithm? Justify it and fill in the following table:

N	t `CloseNode`	t `ShortEdge`
10	…..	…..
20	…..	…..
40	…..	…..
80	…..	…..
160	…..	…..
320	…..	…..

640	…..	…..
1280	…..	…..
…..	…..	
Until heap overflow or take more than 30 minutes		

B. Delivery method

You should **commit and push** your project with your new `session4` package in your Github repository with the following content inside it:

- All the requested source files.
- The requested PDF document called `session4.pdf` with the corresponding activities.

Important:

- Make sure your Github course project is up to date after the delivery and that the last commit is strictly before the deadline. If not, the mark for this session will be 0 without any exception.
- Your mark will be eventually included in the `Marks.xlsx` file in your repository. Please, do not modify anything in that file to avoid conflicts between versions.

Deadlines:

- Group L.I-01 (Thursday): March 18, 2020 at 11:55pm.
- Group L.I-02 (Wednesday): March 17, 2020 at 11:55pm.
- Group L.I-03 (Tuesday): March 16, 2020 at 11:55pm.
- Group L.I-04 (Monday): March 22, 2020 at 11:55pm.