COMP0005 Algorithms
Academic Year 2020-21

Individual Coursework: The London Railway Network

Released: March 22nd, 2021
Submission deadline: April 27th, 2021 @ 4pm UK time

Background.

The London Railway Network consists of approximately 30 lines (spanning the Tube, DLR, TfL Rail and Tram) and 650 stations, covering the entire metropolitan area of London. A complete list of stations is provided in file `londonstations.csv`. Each line after the header row represents one station, and follows this format:

```
Station name, Latitude, Longitude
```

Information about train lines is provided in file `londonrailwaylines.csv`. Each line after the header row represents adjacent stations on the same line, and follows this format:

```
Tube Line, From Station, To Station
```

Note that adjacent stations are connected in both directions, but only one record appears in the csv file (e.g., on the Bakerloo line, Oxford Circus stations and Piccadilly Circus stations are adjacent, but only the record `Bakerloo, Oxford Circus, Piccadilly Circus` appears in the csv file, while the record `Bakerloo, Piccadilly Circus, Oxford Circus` does not – i.e., bi-directional connection is implicitly assumed).

Task.

Your task is to design and implement data structures and algorithms to efficiently support the following API:

- loadStationsAndLines(): reads the provided files `londonstations.csv` and `londonrailwaylines.csv`, and initialises appropriate data structures to represent both stations and railway lines information;
- minStops(from, to): given in input two station names, returns the minimum number of stops that need to be travelled through, to go from one to the other;
- minDistance(from, to): given in input two station names, returns the minimum distance (in miles) that needs to be travelled to go from one to the other. To compute the distance between adjacent stations, compute the Euclidean distance between their latitude/longitude coordinates;
- newRailwayLine(stationSet): given in input stationSet (i.e., an unordered list of station names), computes how to connect them pairwise, so to minimise the sum of the distance (in miles) between adjacent stations, then returns an ordered list of station names connected pairwise. In other words, imagine

creating a new railway line that connects all stations in stationSet, so to use the least amount (in miles) of train tracks overall; then return the ordered list of station names from one end of the line to the other. No branching of the line is allowed.

Constraints. For this coursework, you are expected to implement your own algorithms and data structures. You are NOT allowed to use (import) python libraries, with the exception of `csv` (to read csv files), `math` (to compute Euclidean distances from pairs of latitude/longitude coordinates) and `timeit` (to experimentally measure the computational cost of your solution, as explained below). If in doubt about what you can and cannot use, ask in Moodle "Ask a question" forum.

Hints.

Computing a new railway line following the constraints described above is similar to solving the famous Travelling Salesman Problem (TSP) problem. There is a simple, brute-force algorithm that solves the problem. However, due to its high computational cost, it will only scale to at most ~20 stations.

If you relax the assumption of computing an optimal (i.e., minimum overall train track length) solution, you can adopt heuristics that would give you a much faster solution. In particular, stations can be represented as 2D points in a Euclidean metric space; the simpler Euclidean traveling salesmen problem can then be tackled.

Submission instructions.

This is an individual coursework, and you should NOT be collaborating with your previous group members (or any other person) to develop a solution. Using the Moodle course website, submit a single zip archive containing two files:

- A Jupyter notebook `LondonRailwayNetwork.ipynb` containing your implementation of all data structures and algorithms required for this coursework. This notebook MUST follow the structure of the skeleton code provided. Do not submit the data files (`londonstations.csv` and `londonrailwaylines.csv`); you may assume these are available in the same folder as your Jupyter notebook.

- A PDF document `report.pdf` of maximum 4 pages (font style: Ariel or Times New Roman, font size: 12pt), where you describe the following key points:
  - The data structures you have designed and implemented to represent railway stations and lines.
  - The algorithms you have designed and implemented to efficiently support the requested API.
  - The computational complexity achieved, both theoretically (as order-of-growth) and experimentally. Use the `timeit` python library to measure experimentally the execution time of each API operation.

Submission deadline: April 27th 2021, 4pm UK time.

Assessment.

This coursework represents 60% of the final mark for COMP0005 Algorithms.

Submissions will be evaluated in terms of the following marking criteria:

- Correctness and readability of your code (e.g., is your implementation of the above API computing correct answers? Is your code readable and well documented?) [20 points]
- Efficiency of your code (e.g., have appropriate data structures and algorithms been chosen? Have efficient heuristics been implemented?) [50 points]
- Depth of your analysis (e.g., is the theoretical analysis of the computational cost of your algorithms correct? Is the experimental analysis of your code comprehensive?) [30 points]

Marks for each criterion will be assigned following the UCL CS Grade Descriptor (criteria 4, 5 and 6).

Academic Integrity.

UCL has a very clear position about academic integrity – what it is, why it is important, and what happens if you breach it. Make sure you have read UCL position on this matter before attempting this coursework.

References.

You can read more about the Travelling-Salesman Problem and its Euclidean variant in Section 35.2 “The Travelling-Salesman Problem” of the textbook *Introduction to Algorithms – Third edition*, by T. H. Cormen et al.

Useful Wikipedia links: Travelling Salesman Problem (TSP) https://en.wikipedia.org/wiki/Travelling_salesman_problem and its Euclidean variant (ETSP): https://en.wikipedia.org/wiki/Travelling_salesman_problem#Euclidean