

CSC2001F 2018 Assignment: Graphs

Introduction

This assignment concerns using directed graphs to develop a simulation of a very cheap taxi service offered by supermarket chain QnQ. This service is cost-effective for QnQ partly because taxis follow only some limited routes through the city.

- Taxis are stationed at QnQ shops.
- Clients are people who ask for a QnQ taxi to take them from a QnQ pickup spot to the nearest QnQ shop.
- Edges in the graph represent routes the taxis take, modelled as a **weighted directed graph**.
- As the fare for the taxi is a set price, clients are only taken to the nearest QnQ shop.
- To minimise costs, the nearest taxi must go to the client's pickup spot, and from there it must take the client to the nearest QnQ shop.

Part one of the assignment will be automatically marked, while part two will be manually marked.

Please note that the Automarker will generate random data, so you will not have the same situation every time you run it.

Part one: client-trip simulation

Given an incoming call, the problem is to identify the taxi that can make the lowest cost trip to the client and then to the nearest shop.

You will write a program called 'SimulatorOne.java' that accepts as input a file containing data on roads, shop locations and incoming calls. The program will then, for each call, calculate and output details of the (best) shortest trip.

You must use Dijkstra's algorithm to calculate the lowest cost path from taxi to client to shop.

Input format:

```
<number of nodes><newline>
{<source node number> {<destination node number> <weight>}*}<newline>*
<number of shops><newline>
{<shop node number>}*<newline>
<number of clients><newline>
{<client node number>}*<newline>

('{x})* mean zero or more of item x.)
```

The last line is a chronologically ordered series of calls, where each call is represented by the number of the node that the client calls a taxi from.

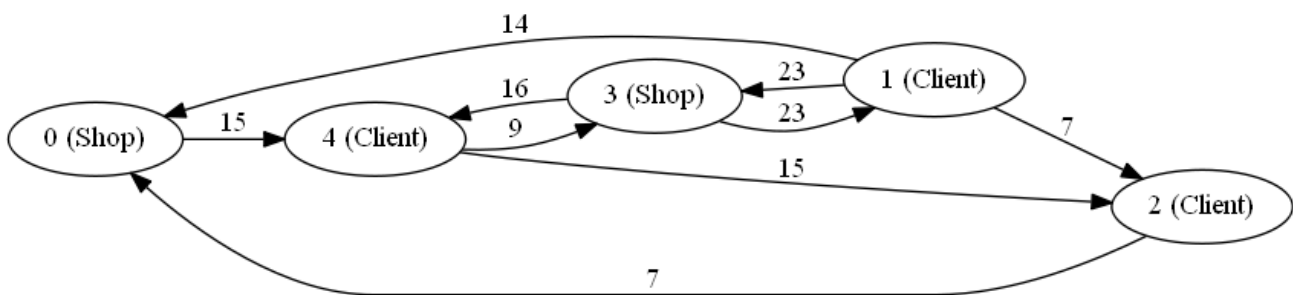
Example:

```

5
0 4 15
1 0 14 2 7 3 23
2 0 7
3 1 23 4 16
4 2 15 3 9
2
0 3
3
1 4 2

```

Representing the following weighted di-graph:



Output format:

Output is ordered by call event:

```

client <node  $C_0$ >
<results for client at node  $C_0$ >
...
client <node  $C_N$ >
<results for client at node  $C_N$ >

```

The results for a client (at a given node) consist of details of the lowest cost pick-up (shortest path from a taxi to the client), and then details of the lowest cost drop-off (shortest path from the client to a shop).

```

taxi <taxi node  $T_X$ >
<shortest path from  $T_X$  to  $C_n$ >
shop <shop node  $S_i$ >
<shortest path from  $C_n$  to  $S_i$ >

```

If there is more than one taxi with the lowest cost pick-up, then these are output in ascending order of taxi node, and if there is more than one shop with the lowest cost drop-off then these are output in ascending order of shop node:

```

taxi <taxi node  $T_X$ >
<shortest path from  $T_X$  to  $C_n$ >
...
taxi <taxi node  $T_Y$ >
<shortest path from  $T_Y$  to  $C_n$ >

```

Continued

...
shop <shop node S_j >
 <shortest path from C_n to S_j >

A path is represented by the sequence of nodes traversed.

In the case that there is more than one minimum-cost trip from a taxi at node T_i to a client at node C_j , the cost of these solutions is reported **instead of** these paths, i.e. output takes the following form:

...
taxi <taxi node T_x >
 multiple solutions cost < C_{min} >.
 ...

Similarly, of there is more than one minimum-cost trip from a client at node C_j to a shop at node S_k , output takes the form:

...
shop <shop node S_i >
 multiple solutions cost < C_{min} >.
 ...

If there is no path from any taxi to a client or from a client to a shop, the output takes this form:

...
client <node C_N >
 cannot be helped
 ...

Example (assuming input data of example on page 2 above):

```
client 1
taxi 3
3 1
shop 0
multiple solutions cost 14
client 4
taxi 0
0 4
shop 3
4 3
client 2
taxi 0
0 4 2
taxi 3
3 1 2
shop 0
2 0
```

Results for the client at node 1 (the first call event) are followed with the results for the client at node 4 (the second call event) are followed with the results for the client at node 2 (the third call event).

In the case of the first call event, using a taxi at node 3 and travelling to the shop at node 0 gives the shortest trip. There are in fact two drop-off routes that the taxi can follow (1 0 and 1 2 0).

In the case of the second call event, there is only one shortest path, and this trip is for a taxi at node 0 dropping off at the shop at node 3.

In the case of the third call event, there are two locations from which taxis can be dispatched that give rise to the same pick-up cost, so both these results are given in ascending order of taxi-node number i.e. taxi from node 0 first and then taxi from node 3.

Part two: extension

This part will NOT be marked by the Automarker.

Create a new version of your simulation called '`SimulatorTwo.java`' that incorporates some advanced features, such as the following:

- Taxis are not always stationed at shops.
- Taxis can take a client to any shop.
- Some/all taxis can carry 2 or more clients.
- The number of taxis is finite.
- Taxis and shops belong to different companies (e.g. QnQ and Shopfite), and taxis only operate between the shops of their own company.
- Client calls are interspersed with traffic reports requiring changes to the graph weightings.

These examples are under specified. We want you to exercise your creativity.

Along with your code, you should submit a '`readme.txt`' document containing a brief description of the features that you have implemented.

Development requirements

For part 2 you are required make use of these additional tools:

- javadoc, for documentation generation
- make, for automation of compilation, documentation generation, and cleaning of files

Submission requirements for part 1 – automatically marked

Submit a .zip compressed archive to the automatic marker containing the source code (.java) files for your solution. This archive will be unpacked and evaluated by the automatic marker.

Submission requirements for part 2 – manually marked

Place your solution to part 2 in a .tar.gz compressed archive called '`part-two.tar.gz`'. It should contain the following:

- Makefile
- src/
 - all source code

- bin/
 - all class files
- doc/
 - javadoc output
- readme.txt
 - description of the advanced features implemented for part two.

Place *part-two.tar.gz* in the ZIP file created for part one, and resubmit it to the automatic marker.

The automatic marker will not assess the contents of *part-two.tar.gz* but will store it for manual marking by tutors. (This submission strategy just ensures everything is in one place.)

Marking guidelines

Artefact	Aspect	Mark
Solution to part one	Correctness	80
Solution to part two	Correctness	15
Development	javadoc and make	5

END