CSC2001F 2019 Assignment 6: Graphs

Introduction

This assignment concerns using directed graphs to develop an algorithm for a new business called UberHaul. As the name suggests, it is similar to Uber, but for moving materials, not people. Clients who need items taken by truck from one location to another can request an UberHaul driver.

- Clients are people who ask for an UberHaul driver to take goods from one node to another node.
- Edges in the graph represent routes the drivers take, modelled as a weighted directed graph.
- Nodes represent locations in the city. Some are where UberHaul trucks are stationed, some are the source or destination of deliveries, and some are just intersections where drivers can change direction to minimise cost.

Generally, the problem is to service delivery requests as cheaply as possible.

For Part One, we will assume that each UberHaul driver, when servicing a request, will start from home and return home afterwards. A driver whose home is at node number *D* and is doing an UberHaul trip to take goods from node number *Cp* to node number *Cd* will take the cheapest route from *D* to *Cp*, from *Cp* to *Cd*, and finally, from *Cd* to *D*.

If there are two or more UberHaul drivers who could service any request at the same minimum cost, then the one whose home has the lowest node number is selected.

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: Delivery simulation

Given a delivery request, the problem is to identify the UberHaul truck that can make the lowest cost trip from the driver's home to the client's pickup node, then to the client's drop-off node, then back to the driver's home.

You will write a program called '`SimulatorOne.java`' that accepts as input a file containing data on roads, locations and delivery requests. The program will then, for each delivery, calculate and output details of the lowest-cost trip to service that delivery.

You must use Dijkstra's algorithm to calculate the lowest cost path.

Input format:

<number of nodes><newline>
{<source node number> {<destination node number> <weight>}<newline>}**
<number of UberHaul drivers><newline>
{<driver's home node number>}<newline>*
<number of delivery requests><newline>
{<pick-up/source node number> <drop-off/destination node number>}<newline>*

(An asterisk after a closing brace means zero or more e.g. '{x}*' mean zero or more of item x.)

The first line gives the number of nodes, N. Nodes are assumed to be numbered 0..N-1.

The next N lines describe the outgoing edges for each node. Each line starts with the node number followed by zero or more edge descriptions, each comprising a destination node number and weight.

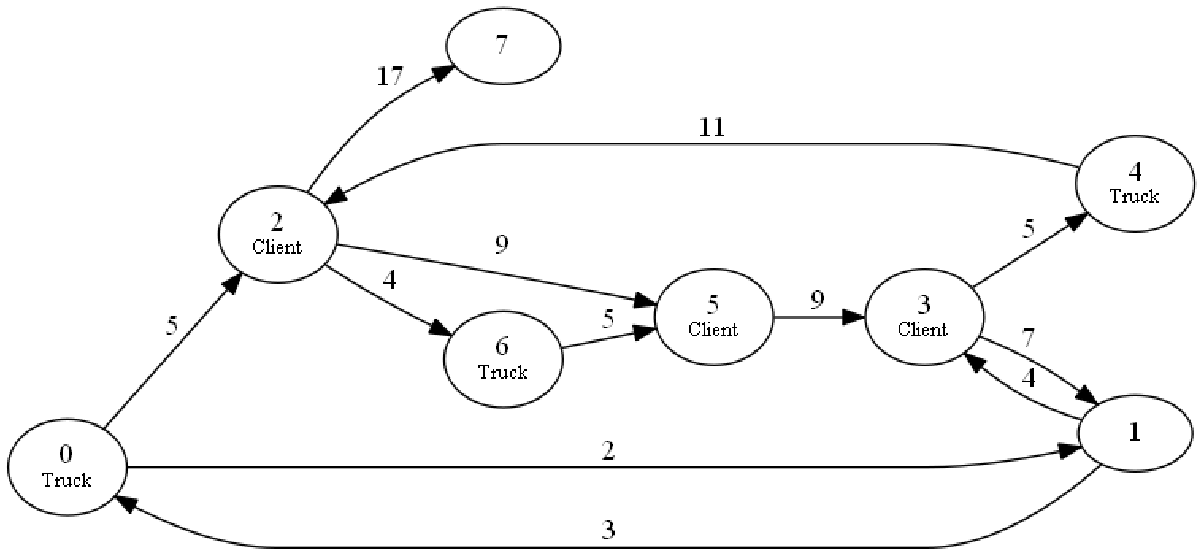
The next pair of lines give information on UberHaul drivers. The first line gives the number of drivers, D. The second line contains a list of the D nodes at which these drivers are located.

The final pair of lines gives information on client requests. The first line gives the number of requests, R. The second line contains the request list. Each request comprising a pair of node numbers: the number of the node where the UberHaul truck must fetch items from (“pickup node”) followed by the number of the node where the truck must deliver these items to (“drop-off node”).

Example:

```
8
0   1   2   2   5
1   0   3   3   4
2   5   9   6   4   7   17
3   1   7   4   5
4   2   11
5   3   9
6   5   5
7
3
0   4   6
3
2   5   5   1   3   7
```

Representing the following weighted di-graph:



Output format

Output is ordered by service request:

client *<node P_0> <node D_0>*
<results for delivery from P_0 to D_0>
...
client *<node P_N> <node D_N>*
<results for delivery from P_N to D_N>

The results for a client consist of details of the lowest-cost circuit from UberHaul driver home to pick-up point, to drop-off point, and back to that UberHaul driver's home:

truck *<driver's home node number T_X>*
<shortest path from T_X to P_i>
pickup *<P_i>*
<shortest path from P_i to D_i>
dropoff *<D_i>*
<shortest path from D_i to T_x>

A path is represented by the sequence of nodes traversed.

If there is more than one UberHaul truck with the lowest cost route, then choose the one with the lowest node number and output that driver's route.

There are 3 "legs" or parts of each solution, namely:

<shortest path from T_X to P_i>

<shortest path from P_i to Di>

<shortest path from D_i to T_x>

If there is more than one shortest path for any leg of the trip, then output the cost of that leg instead of these paths, in the following form:

...
multiple solutions cost *<C_{min}>*.
...

If there is no path from a client's pickup node to their drop-off node, or there is no way that any UberHaul driver can get to a client's pickup node from home and also back home from that client's drop-off node (i.e. if no trip is possible), then output takes this form:

...
client *<node P_N> <node D_N>*
cannot be helped
...

Example (assuming the example input data above):

```
client 2 5
truck 0
0 2
pickup 2
multiple solutions cost 9
dropoff 5
5 3 1 0
client 5 1
truck 0
multiple solutions cost 14
pickup 5
5 3 1
dropoff 1
1 0
client 3 7
cannot be helped
```

Results for the client at node 2 (the first delivery request) are followed with the results for the client at node 5 (the second delivery request) are followed with the results for the client at node 3 (the third delivery request).

In the case of the first delivery request (pickup at 2 and drop-off at 5), using an UberHaul truck at node 0 gives the shortest trip. There are in fact two drop-off routes that the UberHaul truck can follow (2→5, and 2→6→5), hence '`multiple solutions cost 9`' is displayed.

In the case of the second delivery request (pickup at 5 and drop-off at 1), using an UberHaul truck at node 0 gives the shortest trip. Though the truck at node 3 is closer for pickup, total costs are higher. Truck 0 incurs a pickup cost of 14 (again, multiple solutions, which are 0→2→6→5 or 0→2→5), while the return home incurs a cost of 3 (1→0). Truck 3 incurs a pickup cost of 5, while the return home incurs a cost of 17 (multiple solutions, which are 1→0→2→6→5 and1→0→2→5).

In the case of the final delivery request, the client cannot be helped as there is no return from node 7. (Admittedly, a tad artificial.)

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:

- UberHaul trucks are not always stationed at home.
- Some/all UberHaul trucks are big enough to do two deliveries at a time, i.e. they can do the two pickups (in either order) before doing the two drop-offs (in either order).
- UberHaul trucks have different sizes and hence different petrol consumption rates, and this is also factored in when finding the lowest cost routes.
- 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.

Submission requirements

Part one: Automatically marked

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

Part two: 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	90
Solution to part two	Correctness	10

END