Problem Set 3, Programming Part
Shortest Paths

Due: 11:55pm Thursday, November 7, 2019

---

1 Programming Assignment

The robotic duo Max and Min are in a bit of a fix. Min has a critically low battery and cannot move; they are both at the same charging station, but it is broken. If Min's battery is not recharged, naturally they will have to miss the robot opera in the evening. There are a number of charging stations throughout the city, but because Min cannot move, Min cannot get to any of these. So, the plan is for Max to visit a charging station, obtain some power, and return to Min to transfer the power obtained. However, Max can only obtain a small amount of power from each charging station, because Max lives "off the grid" and is not paying for the power. Also, Max cannot spend a lot of time in transit because of the imminent robot opera. Max therefore needs to compute the shortest paths from their current location to each of the charging stations. Fortunately, Max has a map with the pairwise distances between all pairs of charging stations. To get Min charged up for the robot opera, Max needs to compute these (single-source) shortest paths efficiently prior to venturing out.

This problem is described by the following Input and Output.

Input:	A weighted graph G of n vertices. Each edge weight corresponds to the time to travel between two locations.
Output:	For each charging station, the shortest path from the starting location to that charging station and the total distance of the trip.

A Java template has been provided containing two empty methods. The method `ShortestPaths` takes two parameters. The first parameter is an adjacency list `adj` (stored as a three-dimensional integer array) which represents a weighted graph G; the structure of `adj` is explained in the file `ShortestPaths.java`. The second parameter is an integer representing the source vertex. This function should calculate and store the single-source shortest paths from the source vertex to all other vertices. The function `PrintPaths` takes the source vertex as an argument and prints out the paths in a specific format. Your task is to write both the functions. The input graphs will be connected, and you can assume that the graphs are undirected.

You must use the provided Java template as the basis of your submission and start your implementation inside the `ShortestPaths` and `PrintPaths` functions in the template. You may not change the name, return type, or parameters of those functions. The main function in the template contains code to help you test your implementation by reading it from an input file or from the console. A sample input file is also provided (see the comments in `ShortestPaths.java` for the file format). You may modify the main function or any other function because your submission will be tested using a different main function. You can use any helper methods or any helper classes. You can use any built-in class or write your own classes and data structures. We advise you to put all the classes you write in the same file, and so no other class except the provided one should be declared as a public class.

You can also use the "IndexMinPQ" class in the provided file `IndexMinPQ.java`; this class implements an indexed priority queue, and the method "decreaseKey" will be particularly useful for decreasing the priority of an element that already belongs

to the priority queue.[1] When we test your implementation, we will have the file `IndexMinPQ.java` in the same directory, so you do not need to provide this file to us. You really are highly encouraged to use this (unless you want to implement your own indexed priority queue).

2 Examples

The input format is exactly like that of Problem Set 2. The first line denotes the number of vertices n in the graph and the subsequent n lines represent the $n \times n$ adjacency matrix. A 0 at row i and column j in the adjacency matrix means that there is no edge between vertices i and j; otherwise, that number denotes the weight of the edge between i and j. You can assume that the edge weights are between 1 and 1000.

Sample input:

```
3
0   5   6
5   0   7
6   7   0
6
0   7   9   0   0   14
7   0   10  15  0   0
9   10  0   11  0   2
0   15  11  0   6   0
0   0   0   6   0   9
14  0   2   0   9   0
```

Sample output:

```
Reading graph 1
The path from 0 to 0 is: 0 and the total distance is: 0
The path from 0 to 1 is: 0 -- > 1 and the total distance is: 5
The path from 0 to 2 is: 0 -- > 2 and the total distance is: 6
Reading graph 2
The path from 0 to 0 is: 0 and the total distance is: 0
The path from 0 to 1 is: 0 -- > 1 and the total distance is: 7
The path from 0 to 2 is: 0 -- > 2 and the total distance is: 9
The path from 0 to 3 is: 0 -- > 2 -- > 3 and the total distance is: 20
The path from 0 to 4 is: 0 -- > 2 -- > 5 -- > 4 and the total distance is: 20
The path from 0 to 5 is: 0 -- > 2 -- > 5 and the total distance is: 11
Processed 2 graphs.
Average Time (seconds): 0.00
```

[1] Note that the Java API has a PriorityQueue class, but this class does not support an operation like "decreaseKey", and hence the PriorityQueue class will not be sufficient to obtain the requisite runtime.

3 Evaluation Criteria

The programming assignment will be marked out of 40, based on a combination of automated testing (using large graphs) and human inspection.

You are advised to implement Dijkstra's algorithm. The running time of your code should be at most $O(m \log n)$, for n vertices and m edges. The mark for your submission will be based on both the asymptotic worst-case running time and the ability of the algorithm to handle inputs of different sizes.

Score	Description
0 - 15	Submission does not compile or does not conform to the provided template.
16 - 30	The implemented algorithm is substantially inaccurate on the tested inputs.
31 - 40	The implemented algorithm is $O(m \log n)$ and gives the correct answer on all tested inputs.

To be properly tested, every submission must compile correctly as submitted and must be based on the provided template. If your submission does not compile for any reason (including even trivial mistakes like typos) or was not based on the template, it will receive at most 15 out of 40. The best way to ensure your submission is correct is to download it from conneX after submitting and test it.

You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. If you do not receive such an email, your submission was not received. If you have problems with the submission process, send an email to the instructor before the due date.