COMP 251 Assignment 2

Prepared by Prof. Michael Langer

Posted: Tues. Jan. 28, 2014
Due: Sun, Feb. 9, 2014 at 23:59.

General Instructions

- The T.A.s handing this assignment are Mohammed Smaoui and Mingzhou Yang. Their office hours will be posted on the public web page. Office hours will be in Trottier 3110.

- Use the mycourses discussion boards for clarification questions. Prof. Langer will monitor the discussion boards up until Fri. Feb. 7, 5 pm.

- Do not change the package name **(a2posted)**, as this slows down the grading.

- The starter code includes a test class that you may use to examine if your methods are correct. We reserve the right to test your code using a more elaborate set of test cases than the ones provided. If you identify situations that are not covered by the given test cases, then everyone would appreciate it if you would share these on the discussion board. (But do *not* hand in your tester class. If you do hand it in, the grader will ignore it.)

- Follow the other general instructions as Assignment 1. For example, do not change code in the submitted file except where you are allowed. Failure to follow the instructions may result in a penalty.

- **Late assignment policy:** Late assignments will be accepted up to only 3 days late and will be penalized by 20 points per day. If you submit one minute late, this is equivalent to submitting 23 hours and 59 minutes late, etc.

Introduction

In this assignment, you will implement two versions of Dijkstra's algorithm using the **IndexedHeap** class given in the Assignment 1 solutions. The first is the one discussed in class. Recall that Dijkstra's algorithm uses a priority queue to keep track of the shortest cost path to each vertex. For the version presented in class (which is the most common version), the size of the priority queue is at most the number of vertices in the graph. The algorithm also keeps track of the parent of each vertex, which allows one to reconstruct the shortest path to each vertex after the algorithm terminates.

The second version of Dijkstra's algorithm is new for you. It also uses a priority queue, but now it's a priority queue containing edges rather than vertices. Whenever a vertex u moves from V \ S to S, the algorithm adds all edges of the form (u,v) into the priority queue. The priority of the edge (u,v) is defined to be the total distance of the shortest path to u (which is known, because we are moving u from V \ S to S) plus the cost of the edge (u,v). You will need to think through for yourself why this algorithm is indeed correct.

Note that this algorithm could potentially move each edge into the priority queue, and so the size of the priority queue is O(| E |) rather than O(| V |). This larger size could be a significant disadvantage if the graph is dense, that is, if |E| is close to V^2, and that's why the first version of Dijkstra is the typical one.

Your task:

Fill in the missing code in the following two methods which are defined in the class **Dijkstra.**:

- **dijkstraVertices( )**
- **dijkstraEdges( )**

The assignment is out of 100 points – 50 for each method.

Notes:

- You are given several other classes in addition to the **Dijkstra** class. There is a **Graph** class that contains enough methods for you access the vertices and edges. There is a **GraphReader** class which allows you to read in graphs that are specified in a text file in a simple format. Example graph files are provided but you should test other examples as well. We encourage you to share these example graph files with your classmates. There is also a **TestDijkstra** class which you can use for reading in the graph and running Dijkstra's algorithm.

- The **changePriority()** method of the **IndexedHeap** class is not needed by **dijkstraEdges( )** since once an (edge, priority) pair is added to the priority queue, the priority doesn't change. But we can (and will) use this implementation of the priority queue, even though we don't use the **changePriority()** method.

- The algorithms given in class used pseudocode that was based on arrays e.g. parent[ ], d[ ] for shortestDistance. The code you are given uses hash maps instead of arrays. This is a more general approach since the vertex names can be anything, rather than having to be a fixed set of integers {1,..,n}. If you are not experienced with hashmaps other than A1, then this will require extra effort on your part. But it is worth it. Hashmaps often make life easier in the long run. (Those of you who are python

programmers and use dictionaries heavily know what I mean!) If you would like to see a basic implementation of a hashmap, see the code next to lecture 32 in my COMP 250 lecture notes.

What to Submit

- A single file called **Dijkstra.java** with your name and student ID at the top, including the two required methods. Please read the General Instructions carefully to avoid problems with your submission.

Have fun, get started early, and good luck!