Computer Science 431
Algorithms
The College of Saint Rose
Spring 2015

Problem Set 8: Dijkstra's Road Trip and More

Due: 4:00 PM, Thursday, April 30, 2015

You may work alone or in groups of size 2 or 3 on this assignment. Only one submission per group is needed.

Written Problems

Question 1:
Levitin Exercise 7.3.5, p. 275. You may look up the answer to the birthday paradox, then answer the question about the implication for hashing. (3 points)

Question 2:
Levitin Exercise 7.3.7, p. 275. (5 points)

Question 3:
Levitin Exercise 8.1.7 part a, p. 291. (4 points)

Read section 8.3 about optimal binary search trees. Then answer:

Question 4:
Levitin Exercises 8.3.1, p. 303, (4 points)

and

Question 5:
Levitin Exercises 8.3.5, p. 303. Justify your answer (of course). (2 points)

Question 6:
Levitin Exercise 8.4.1, p. 311 (5 points).

Read section 9.2 about Kruskal's Algorithm and union-find algorithms. Then answer

Question 7:
Levitin Exercise 9.2.1, p. 331-332, (5 points)

and

? Question 8:
Levitin Exercise 9.2.2, p. 332. (4 points)

? Question 9:
Levitin Exercise 9.3.7, p. 338. Provide pseudocode and justify its efficiency class. (6 points)

? Question 10:
Using the graph from Bailey Problem 16.7, p. 436, use Dijkstra's Algorithm to compute the shortest distance from Dover to Phoenix by filling in the tables below, using the algorithm and notation as shown in the example in the graph notes. (10 points)

The data structures and the implementation of Dijkstra's algorithm are similar to those you will be using for the programming assignment below. Vertex labels are of type `City` and edge labels are of type `TravelLink`. You may assume that `TravelLink` provides methods to retrieve road names, distances, and driving times, but those details will not be important. The priority queue will contain objects of type `ComparableAssociation<Integer,Edge<City,TravelLink>>`, where the keys are the distances or driving times, as appropriate.

Note that by using a `ComparableAssociation`, the `ComparableAssociation`'s `compareTo` method (which in turn just calls `compareTo` methods of its keys, in this case `Integers`), the generic priority queue implementations can be used. The algorithm will populate a `Map` of shortest/fastest routes. The `Map`'s keys will be of type `City` and the values will be `ComparableAssociation<Integer,Edge<City,TravelLink>>` objects, from which we can get the total distance/time to the `City` from the starting `City`.

For each case, you are to fill in the given table, representing the `Map`, in the order in which the algorithm fills in entries, and show the values in the priority queue. Do not erase values as they are removed from the priority queue, just cross them out and write a number next to them to indicate the order in which they are removed from the queue. The map and priority queue should indicate their contents at the time the city "Phoenix" is added to the map.

Fill in the table in Table 1 or create an equivalent table of your own, which is a `Map` that has `City` objects as keys and `ComparableAssociations` of the shortest distance from Dover to the last edge traversed on that shortest route as values.

It is easiest to specify edges by the labels of their endpoints rather than the edge label itself, which might not be unique.

Also, use the table in Table 2 to keep track of your priority queue. Remember, don't erase entries when you remove them from the queue, just cross them out and mark them with a number in the "Seq" column of the table entry to indicate the sequence in which the values were removed from the queue.

Implementing Dijkstra's Algorithm

Your final programming task is to develop a simplified "driving directions" system based on the

mapping data you have been working with.

You should use a variant of Dijkstra's Algorithm to compute shortest path from a given starting point (a graph vertex) to a given destination point. The general form of Dijkstra'a Algorithm computes the shortest paths from a starting vertex to all other vertices, but you will be able to stop one you find a shortest path to the specified destination rather than calculating the shortest path to all other places. You will also need to make sure that you can efficiently print/write the computed route in the proper order (starting point to destination point).

Once a shortest path is computed, you will need to be able to output it in a human-readable form (for the `FindRoute` command) or in a form plottable by HDX (for the `MapRoute` command).

For example, if you load the `ny-all.gra` file, and compute a shortest path for a few nearby points: `US20@WesAve` (the "Y" intersection at Western and Madison right near campus) and `NY2/US9` (Latham Circle), your path would traverse the following points:

```
US20@WesAve, NY443/US9W@US20&US20@US9W, NY5/US9W, US9/US9W
I-90@6/US9, NY377/US9, NY378/US9, NY155/US9 and NY2/US9.
```

Your "human readable" output might look something like this:

```
Travel from US20@WesAve to NY443/US9W@US20&US20@US9W
 for 1.56 miles along US20, total 1.56
Travel from NY443/US9W@US20&US20@US9W to NY5/US9W
 for 0.37 miles along US9W, total 1.93
Travel from NY5/US9W to US9/US9W
 for 0.28 miles along US9W, total 2.21
Travel from US9/US9W to I-90@6/US9
 for 0.87 miles along US9, total 3.09
Travel from I-90@6/US9 to NY377/US9
 for 0.44 miles along US9, total 3.53
Travel from NY377/US9 to NY378/US9
 for 2.04 miles along US9, total 5.57
Travel from NY378/US9 to NY155/US9
 for 2.24 miles along US9, total 7.81
Travel from NY155/US9 to NY2/US9
 for 0.78 miles along US9, total 8.59
```

Your plottable data for the Highway Data Examiner should be in a "`.pth`" file. This file format must match the following:

```
START US20@WesAve (42.666502,-73.791776)
US20 NY443/US9W@US20&US20@US9W (42.652458,-73.767786)
US9W NY5/US9W (42.656734,-73.763301)
US9W US9/US9W (42.659938,-73.759975)
US9 I-90@6/US9 (42.669562,-73.748817)
```

```
US9 NY377/US9 (42.675873,-73.747659)
US9 NY378/US9 (42.704925,-73.754568)
US9 NY155/US9 (42.736832,-73.76225)
US9 NY2/US9 (42.748115,-73.761048)
```

Here, each line describes one "hop" along the route, consisting of the road name of the segment (*i.e.*, your edge label), the waypoint name (*i.e.*, the label in your vertex), and the coordinates of that point. The exception is the first line, where we substitute `START`, since you don't have to take any road to get to your starting point.

These files should be given a `.pth` extension. Once such a file is created, it can be visualized by directing a browser at `http://courses.teresco.org/chm/viewer/` and uploading the `.pth` file in the file selection box at the top of the page.

Printing human-readable directions is worth 20 points, and generating a `.pth` file is worth 5 points.

Submitting

Before 4:00 PM, Thursday, April 30, 2015, submit your problem set for grading. To complete the submission, upload an archive (a `.7z` or `.zip`) file containing all required files using Submission Box at `http://sb.teresco.org` under assignment "PS8".

Grading

This assignment is worth 80 points, which are distributed as follows:

Feature	Value	Score
Question 1, 7.3.5	3	
Question 2, 7.3.7	5	
Question 3, 8.1.7	4	
Question 4, 8.3.1	4	
Question 5, 8.3.5	2	
Question 6, 8.4.1	5	
Question 7, 9.2.1	5	
Question 8, 9.2.2	4	
Question 9, 9.3.7	6	
Question 10, Dijkstra's Algorithm Practice	10	
`Mapping FindRoute` command	20	
`Mapping MapRoute` command	5	
`Mapping` style, documentation, and formatting	7	
Total	80	

City	(distance,last-edge)
Dover	(0, `null`)

Table 1: `Map` of Dijkstra's algorithm results for last edge traversed to find each `City` for the first time.

(distance,last-edge)	Seq

Table 2: Table showing the progress of the priority queue in Dijkstra's algorithm.