JDEP 283 Homework 3 Fall 2008

Due Thursday, November 13, 2008

Name _____

Instructions Follow instructions *carefully.* This homework consists of two parts; a set of analytical problem from the class text, Levitin, and a programming assignment. For the analytical part, you should solve all of them and submit them individually on or before the due date. For programing part, you are allowed to work with a partner of your choice.
Clarity and legibility of presentation of your submission are as important as your answers to problems. You are very strongly advised to typeset your solution using some document processing system. If the grader cannot easily read your writings, you may not be awarded full points even if you claim your answers are correct. Staple this cover page to the front of your assignment for easier grading. For the programming part, be sure to submit all your program files via the webhandin. You should follow JDE coding standard which is available at the course homepage `http://www.cse.unl.edu/~vinod/jdef08/info.html`.

Problem	Page	Points	Score
6.4.1a	224	5	
7.2.2ab	264	5	
9.1.7ab	314	10	
9.2.1ab	321	10	
9.3.2b	327	5	
9.3.3	327	5	
9.4.1abc	332	10	
problem sub total		50	
Program			
Correctness		30	
Style/Documentation		5	
Analysis		15	
program sub total		50	
Total		100	

Programming: For this assignment, you will be implementing Dijkstra's algorithm using a min-heap as the priority queue. You are to implement the heap yourself as a separate class, called MinHeap (not use an existing implementation from a Library). Dijkstra's Algorithm should be implemented within a source code file called ShortestPath. The Heap class should have the following methods:

- MinHeap() - constructor that creates an empty heap.
- insertItem(- inserts new node at the end of the heap.
- heapify() - shifts nodes around to maintain min-heap characteristics.
- deleteItem(- deletes root node, and rearranges heap so that minimum priority key is new root.
- is empty() - determine is heap is empty.
- printHeap() - should print to STDout the content of the heap, starting at the root, and going to each level of the heap in order (if you use an array implementation, this would mean starting at the zeroth index and printing out the array in order).

The shortestPath Program should have the following methods:

- ShortestPath(G) - Constructor, takes a graph as argument and set up needed data structures to implement Dijkstra's algorithm.
- printGraph() - should print to STDout the content of the graphs from the input file.

- Dijkstra(vertex)- takes as argument a source vertex and prints to STDout the vertex name, predecessor, and shortest path length from source for each of the vertexes in the graph (including the source).

Finally, the test program should be called TestDijkstra and should take as command line argument the name of a test file which will contain the data on which your implementations will be tested. The format for the will be as follows:

```
Graph
a b c d e
b a 3
a d 7
d b 2
b c 4
c e 6
e d 4
d c 5
Dijkstra a
Dijkstra c
```

The two keywords are Graph and Dijkstra. The first line of the file will always be followed by the data for the graph. The line following the keyword Graph will contain the list of nodes. Each line following this first will contain the information for the weighted edges. You are to assume this is a directed graph. So line 'b a 3' is translated as the edge from b to a with weight 3, but does not imply that the edge a to b exists. After the graph is read in, it should be printed out to STDout. The keyword Dijkstra is followed by a vertex that is to be the source for which the shortest distance must be calculated. So, 'Dijkstra a' is translated: perform Dijkstra's algorithm on the preceding graph using 'a' as the source vertex. The results should be printed to STDout.
You can assume that the input file will not include a call to Dijkstra's before a graph is read in. However, your program must be able to catch an error (without hanging or terminating) if you are asked to perform Dijkstra's algorithm with a source node that is not in the graph.