CSCI 338—Algorithm Design and Analysis
Programming Assignment 4: Dijkstra's algorithm

Due: Wednesday, March 15, 2017

This is an individual assignment.

This assignment is to implement Dijkstra's algorithm using an adjacency-matrix graph representation on a weighted undirected graph.

Language

You may use Java, C, C++, or Haskell for this assignment. If you wish to use a different language, you must first get permission from the instructor.

Building random weighted graphs

Create a new version of your graph-building code that supports graphs with integer weights. Your code should only produce edges with positive integer weights. Here are some helpful suggestions:

- Use the appropriate maximum value for your data type as the indication that there is no such edge (*Integer.MAX_VALUE* for *int* in Java, *MAX_INT* from <*climits*> for *int* in C++) instead of zero to allow for the possibility of zero and negative weight edges in the future. This value serves in place of *infinity*, and you must be careful to treat it as a separate value.
- Supplement the function/method that returns an all-zero graph with one that returns an all *infinity* graph.
- Add a *maxWeight* parameter to the functions/methods that return random graphs to put an upper bound on the weight of an edge. You can use a low value for this parameter to insure that several or even many edges have the same weight and a high value to increase the likelihood that no two edges have the same weight.
- Choose the edges as before, but instead of putting a one in the matrix for an edge, randomly select a weight between one and *maxWeight*, inclusive, and put that value in the matrix.

Dijkstra's algorithm

Implement Dijkstra's algorithm from Figure 4.8 in the textbook. You can use the built-in Java *PriorityQueue* class or C++ *priority_queue* class for your priority queue. Note, however, that neither of these supports the *decreaseKey* operation, so you'll need to remove the element and re-insert it to get it to its proper place. If you prefer, you can use your own priority queue from CSCI 200 and modify it to include a `decreaseKey` method.

Testing the algorithms

Using several six to ten-vertex graphs, run Dijkstra's algorithm on them and print out the *prev* and *distance* arrays for each graph in a form that allows you to see the values for each vertex together. Verify that your results are correct by comparing these values to the graph itself, either in its adjacency matrix representation or in its diagram.