CSC172 LAB

DIJKSTRA'S ALGORITHM

1 Introduction

The labs in CSC172 will follow a pair programming paradigm. Every student is encouraged (but not strictly required) to have a lab partner. Labs will typically have an even number of components. The two partners in a pair programming environment take turns at the keyboard. This paradigm facilitates code improvement through collaborative efforts, and exercises the programmers cognitive ability to understand and discuss concepts fundamental to computer programming. The use of pair programming is optional in CSC172. It is not a requirement. You can learn more about the pair programming paradigm, its history, methods, practical benefits, philosophical underpinnings, and scientific validation at http://en.wikipedia.org/wiki/Pair_programming .

Every student must hand in his own work, but every student must list the name of the lab partner (if any) on all labs.

This lab has six parts. You and your partner(s) should switch off typing each part, as explained by your lab TA. As one person types the lab, the other should be watching over the code and offering suggestions. Each part should be in addition to the previous parts, so do not erase any previous work when you switch.

The textbook should present examples of the code necessary to complete this lab. However, collaboration is allowed. You and your lab partner may discuss the lab with other pairs in the lab. It is acceptable to write code on the white board for the benefit of other lab pairs, but you are not allowed to electronically copy and/or transfer files between groups.

2 Weighted Shortest Path

Previously, you implemented a basic graph ADT using the adjacency matrix data structure, and maybe the uniform-weight shortest path problem. In this lab, you will implement the weighted shortest path algorithm. This algorithm is discussed in detail in section 9.3.2 of your assigned reading (Weiss). A pseudocode description of the algorithm is given in Figure 9.31 of your text, and is repeated below. Use this approach to solve the weighted shortest path problem.

```
// Pseudocode for Dijkstra's algorithm (taken from Weiss Fig 9.31)
```

```
void dijkstra (Vertex s)
{
      for each Vertex v
      {
            v.dist = INFINITY;
            v.known = false ;
      }
      v.dist = 0 ;

      for (  ;  ; )
      {
            Vertex v  = smallest unknown distance vertex ;
                  if (v == NOT_A_VERTEX)  break ;
                  v.known = true ;
                  for each Vertex w adjacent to v
                        if(!w.known)
                              if (v.dist + cvw  < w.dist)
                              {
                                    // update w
                                    decrease (w.dist to v.dist + cvw) ;
                                    w.path = v ;
                              }
                  }
      }
}
```

1. Make certain that your previous implementation of the graph ADT is functional.
2. Define a file format for graphs as follows. The first line of a text file contains the number of nodes in the graph. The second line of the file contains either a 'U' signifying an undirected graph, or a 'D' signifying a directed graph. Every line that follows represents an edge/arc in the graph. Implement a method to read graphs from such a format into you graph ADT. Show your code working for the graph found in figure 9.20 (directed) of the Weiss text book
3. Test your graph reader method on at least three graphs. Show your code working for the graph found in figure 9.20 (directed) and of the Weiss text book and two graphs of at least equal complexity of your own design.
4. Implement the unweighted shortest path algorithm using the pseudocode given. You will have to make a number of design decisions in order to get this algorithm to work with your ADT.
5. Test your implementation of the shortest path algorithm on at least six graphs. Show your code working for the graph found in figure 9.20 (directed) of the Weiss text book. You also must show it working for at least five other graphs that you design yourself (don't copy from anyone

else).

6. Provide a runtime analysis of your code and discuss in your README how your implementation compares to the theoretical running time of the pseudocode presented in the text book.

3 Hand In

Hand in the source code from this lab at the appropriate location on the blackboard system at my.rochester.edu. You should hand in a single compressed/archived (i.e. “zipped”) file that contains the following. You will need to include the images of your plotted results.

1. A README that includes your contact information, your partner's name, a brief explanation of the lab (A one paragraph synopsis. Include information identifying what class and lab number your files represent.). And a one sentence explaining the contents of all the other files you hand in.
2. Several JAVA source code files representing the work accomplished for this lab. All source code files should contain author and partner identification in the comments at the top of the file. It is expected that you will have a file for the test program class.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

4 Grading

172/grading.html

Each section (1-6) accounts for 15% of the lab grade (total 90%)

(README file counts for 10%)