Project 3

CS 241 – 02

Minwoo Soh

[msoh@cpp.edu](mailto:msoh@cpp.edu)

Thursday, March 8, 2018

Section 1. Project Description:

Input: Two files - one contains city data and the other contains road data.

* **city.dat**: This file contains information about cities, where each line has 5 attributes: *City Number, City\_Code (2 letters), Full\_City\_Name, Population,* and *Elevation*.
* **road.dat**: This file contains information about roads, where each line has 3 attributes: *From\_City, To\_City, and Distance*. Note that all roads are assumed to be one-way.

Output: A menu driven system which has the following options.

* Read the original data files and store the data to appropriate data structures.
* Let the user of this program enter a *City Code* and your program should print out the city information (the whole record).
* Find the connection between two cities.
  + The user will be asked to enter two *City Codes*. The program finds the shortest distance between the two cities.
* Insert a road (edge) between two cities.
  + The user will be asked to enter two *City Codes* and its *Distance*. Note that if a pair of *City Codes* already exists or if the *City Code* doesn't exist, print out a warning message.
* Delete a road (edge).
  + The user will be asked to enter two *City Codes* for a road. Note that if the road entered doesn't exist, print out a warning message.
* Exit.

Section 2. Project Specification:

The project required us to create a Directed Graph, or a Digraph as it is named in the project, and a main program to test out the default methods. The vertices were represented by different cities and the edges were represented by the one-way roads going from one city to another. The Digraph class used the provided files (city.dat and road.dat) to create a graph with the city information and the connecting roads. The project description specifically required the Digraph class to have functions that can provide full city information, add a new edge, remove an existing edge, and find the shortest route between two cities. Dijkstra’s Algorithm and ADT Priority Queue were necessary for the function that searched for shortest route. A separate class called Vertex was also made to keep track of visited vertices. A main program, named Project3, was made to test out the Digraph and the function with Dijkstra’s Algorithm by receiving commands from the user and running the specified functions.

Section 3. Testing Methodology:

Testing to see if the code worked correctly came down to running the program repeatedly with different inputs and comparing. An online tool called Graph Online (http://graphonline.ru/en/?graph=glAJwnEMAYoonhcv) was used to create the same graph outline using the provided information in the two given files. The tool was then used to test out Dijkstra’s algorithm implementation by using the tool’s own algorithm and comparing its results with the project’s results. This method was easy to test correctness since it displayed the shortest route length and the vertices it passed through to reach the destination, which was similar to the results of the project’s implementation. The program was run multiple times with different commands such as finding shortest routes for all other cites and deleting or adding multiple edges before testing the algorithm once again. The same steps were applied to the online tool, and both results were compared ensure the program’s results were correct.

Section 4. Lesson Learned:

Coding the Dijkstra’s Algorithm and testing out the program multiple times helped me understand how the time complexity of the algorithm was determined. The time complexity of the algorithm when using an adjacency matrix and an array for keeping track of distances is O(E + V^2). The algorithm traverses every road to get its weight and to reach each vertex, so E represents all roads in the graph. To get the shortest distance to other cities, every city must be visited so that the algorithm can calculate the distances. When a city is visited, it is possible that distance to all its neighbors will be altered. Both cases are combined to represent V^2.

Section 5. Analysis of Output:

The program was tested with the requested commands and all outputs matched the provided output in the project pdf. The user interface was so made to closely resemble the project’s specifications. The program was also tested with different inputs, and its results were compared with an online tool to ensure all necessary functions, such as Dijkstra’s Algorithm, were working correctly.