Name: Jemil Patel

Instructor: Gary Thai

CMSC 204 Project-6

**Approach, Design & Algorithm**

Project-6 was the one of the most challenging projects I encountered this semester, and it proves why it is given at the end. I approached the project first by reading the provided write-up file and understanding what it is about. I got a rough idea that I needed to create a graph structure that will help the user explore routes from one city to the other.

I started by creating Town class which was quite easy to implement. The only thing I was confused about here was that how could I keep track of adjacent towns, and therefore, I added getter, add, and remove methods for the list of adjacent towns. Next, I went over to Road class. In here the thing I needed to be most careful of was the contains method. Since a town can either be the source or the destination. Both Town and Road classes were easy to implement as I followed the given Javadoc.

Now comes the most tedious part of the whole project, which is to create Graph class. I am not considering other methods here, but I am especially talking about the dijkstraShortestPath and shortestPath methods. Before starting to code for these methods I had to first understand the algorithm thoroughly, and I would like to mention a YouTube video that helped me a ton to do this project successfully: [How Dijkstra’s Algorithm Works](https://www.youtube.com/watch?v=EFg3u_E6eHU). Finally, after hours of brainstorming, testing, and editing I was able to come up with the logic for these methods. In short, I would say that I converted the steps from the video into code. In order to keep track of the previous town for a given town, which is a part of the shortest path, I used a HashMap<Town (current), Town (previous)>. Apart from that, I also created a new private method that finds the unexplored town with the smallest weight, that I used in dijkstraShortestPath method.

Lastly, TownGraphManager class was again not much difficult to implement as the core logic was already done in Graph class. Not to mention, I also created my own student JUnit test cases and tested my project with it.

**Test Cases**

**Add Town (without giving any town name)**

**Graphical user interface, application

Description automatically generated**

**Add Road (without giving any details)**

**Graphical user interface

Description automatically generated**

**Find Connection (without any towns)**

Graphical user interface

Description automatically generated

**Add Road (negative weight)**

**Graphical user interface

Description automatically generated Graphical user interface, text, application

Description automatically generated**

**Read File (Display Towns and Display Roads using MD Towns(1).txt)**

**Graphical user interface

Description automatically generated**

**Find Connection (using data from MD Towns(1).txt)**

**Graphical user interface

Description automatically generated**

**Find Connection (using the same town as source and destination)**

**Graphical user interface

Description automatically generated**

**Read File (Display Towns and Display Roads using US Towns(1).txt)**

**Graphical user interface

Description automatically generated**

**Find Connection (using data from US Towns(1).txt)**

**Graphical user interface

Description automatically generated**

**Find Connection (using data from US Towns(1).txt)**

**Graphical user interface

Description automatically generated**

**Learning Experience**

Project-6 has been a great learning experience for me. As I said earlier, this was one of the most challenging projects I did this semester and after completing it, I got a thorough understanding of graphs and Dijkstra’s algorithm.

The toughest part for me was to implement the dijkstraShortestPath method. In order to do that, I first had to understand the Dijkstra’s algorithm completely. Then I had to keep track of explored towns, the next unexplored town with the smallest weight, the previous town to each town for the shortest path, and comparison between the newly calculated weight versus the original weight associated with that town.

If I had a chance to change anything, I would not because I gave my best and have tested the program several times to ensure that I do not leave any loose ends. All in all, I am very satisfied and proud with myself that I completed the project as per the given requirements.

**Assumptions / Notes:**

* Once you load a file into the GUI, then you need to restart the program in order to start working with another file. Since I noticed that the GUI does not clear the data of the previous file after loading the new file.
* In US Towns(1).txt, there exists no direct path between Chicago and Indianapolis. However, a direct path between the two cities is shown in the map image provided in the PPT. This might lead to unexpected output, and so there is no fault in my logic.