Contributors: Jason Gray, Jacob Zacharia, Gavin Rosander

### Problem Description

This project prompts us to implement a map containing cities and one-way roads, and find the shortest distance and path between two valid cities. Our program builds its map from two text files, containing every city and every road. Then, it takes user input on the command line for two city names. It verifies multiple things with the input cities: They are both cities that exist, there are exactly two cities input, the two cities given are not the same location. If these are not met, a prompt is displayed detailing the issue. If they are all met, the program prints out some details about each city, and if a path exists, the distance and legs of the path.

### Program Design

1. For the shortest path problem, we used Dijkstra’s shortest path algorithm. We used this algorithm because it efficiently provides the shortest path from one city to every other, and it provides an easy way to calculate total distance as well as the path taken. It is important to note that this algorithm requires no negative weight cycles. Since this program implements a map with distances, it is impossible for there to be any edges with a negative weight.
2. For this program, we used a Graph class, with a City struct and a Road struct. The graph class was used to implement a directed graph data structure, using city objects as vertices and road objects as edges. The only graph specific functions that were necessary were to add a city and add a road. These classes were used because they allowed for the minimum required functionality in the program. We tried to keep the program as lightweight as possible.

### System Implementation

The graph class was implemented using a vector to maintain all of the cities, and a vector of vectors adjacency list to maintain all of the roads. In the cities vector, each vector index correlates to the index of each city object contained in the vector. In the adjacency list, the outside vector index correlates to the index of each city, and the vector for each index correlates to every road starting at that city. The graph class contains a constructor to build a graph, a function to add a city to the graph, a function to add a directed road to a graph, a function to print input cities, and dijkstra’s algorithm to find the shortest path. Dijkstra’s algorithm was written as a void function of the graph class, to enable us to use the results specifically as needed in the main method. In this case, it printed out the distance and the path from one city to another.

The city struct was implemented to represent a vertex in the graph. It contains member variables for ID, 2 letter code, name, population, and elevation. This struct contains a constructor, as well as an overload for << and >> for use with iostream.

The road struct was implemented to represent a directed edge between vertices in the graph. It contains member variables for starting city, destination city, and weight. This struct contains a constructor, as well as an overload for << and >> for use with iostream.

The main method did some heavy lifting in this program. It used file stream to parse inputs from a text file for every city, as well as to parse inputs from a text file for every road and used all of these file inputs to construct a graph. Additionally, as user input in the command line drives the results of the program, there is error checking to ensure that appropriate prompts are input. If there are no inputs, a usage message is displayed, and if inappropriate prompts are input, an error message detailing why is displayed. When appropriate prompts are displayed, information about the cities is displayed as well as the shortest distance between them and the path from one to the other. If no path exists, that is displayed.

The only problem we ran into during implementation was using a priority queue for Dijkstra’s algorithm. Since a priority queue uses a node with a priority, we determined that there are two solutions. In one solution, we could assign a member variable in the city class that represents the priority. In order to do that, we would also need to implement a compare class so that a priority queue uses that field as the priority. The second solution, and the one that we implemented, uses a pair object from the STL as the object inserted into the priority queue. This allowed us to add a pair of numbers, one representing the city id, and the other representing the priority (distance). We found this to be the simpler of the two solutions.

### Results

1. Did your results match the output in the “sample\_results.txt” file?

Yes.

1. What are your answers to the following questions:
   1. The shortest distance and path from FI to GG

**From City: IRWIN, population 4120, elevation 932**

**To City: GRPVE, population 913330, elevation 952**

**The shortest distance from IRWIN to GRPVE is 24**

**through the route: IRWIN->PARKER->GRPVE**

* 1. The shortest distance and path from PD to PM

**From City: PARKER, population 2190, elevation 1829**

**To City: POMONA, population 698300, elevation 298**

**The shortest distance from PARKER to POMONA is 133**

**through the route: PARKER->BOSSTOWN->TORRANCE->POMONA**

* 1. The shortest distance and path from PM to PD

**From City: POMONA, population 698300, elevation 298**

**To City: PARKER, population 2190, elevation 1829**

**The shortest distance from POMONA to PARKER is 357**

**through the route: POMONA->EDWIN->ANAHEIM->VICTORVILLE->CHINO->GRPVE->IRWIN->PARKER**

* 1. The Shortest distance and path from SB to PR

**From City: BERNADINO, population 1293200, elevation 1033**

**To City: RIVERA, population 189820, elevation 1190**

**The shortest distance from BERNADINO to RIVERA is 152**

**through the route: BERNADINO->ISABELLA->BREA->CHINO->RIVERA**

### Conclusion

In conclusion, our project properly solves the intended problem and we feel it is implemented in a lightweight and efficient manner. Our project properly checks for errors as well as gives a detailed and concise explanation if an error is discovered. From completing this project, we have learned how to implement multiple classes and functions for a real-world scenario, not just to pass a unit test. Additionally, we learned how to mold relatively standard functions to accept the inputs and provide the outputs that are necessary for our specific program.