4.2. Problem Description
In this section, describe the problem you are solving and the functionalities of your program to solve the problem.

We are given 2 txt files, which have information about cities and the roads, and we have to create a program that finds the most optimal route between the cities. The program we created uses Dijkstra's algorithm to optimally sort through the various cities using the road distances between them.

4.3. Program Design
Describe the overall design of your program.

What algorithm did you choose for implementing the shortest path problem? Why this algorithm is applicable here?
What classes do you use or add? Why do you choose to use those classes?

We used Dijkstra's algorithm because it is useful when trying to find the shortest path between nodes of a data structure, which in this case was the cities and roads.

We created a CityGraph class in order to create a graph of all of the cities and the roads that connected them. We also created a PriorityQueue class to ensure that the Dijkstra's algorithm was able to sort the data correctly.

4.4. System Implementation
Describe the details of your implementations. Did you run into problems in your implementation? How did you overcome those problems?

We created 2 classes for this program, CityGraph and PriorityQueue. Since priority queue is a well known C++ data type, its implementation was relatively easy since there was lots of documentation online as to how it worked. For CityGraph, we defined 2 structs, one for cities and one for roads, that had various data members to store all of the codes, names, distances, etc. Then, in the class itself, the member functions mostly served to read the data from the 2 included text files using the fstream function, storing the road and cities structs in unordered maps.

For the Dijsktra's algorithm, we followed the guidelines included in the class notes, first setting the optimal distances to the integer limit, and then updating them as the algorithm iterated through the unordered maps of the cities and roads to find the optimal paths.

4.5. Computational Complexity
Must answer the following.

What is the computational complexity of the algorithm you used to find the shortest path between cities? How did you determine this complexity? Please justify why this algorithm and its complexity are acceptable given the criteria of the problem.

For the time complexity, we used Dijsktra's algorithm using a min heap data structure. When we initialize the program, we iterate through all of the cities, which takes O(V), V being the number of vertices, which in this program represent cities. The main loop of the algorithm uses pop and push in the priority queue, both of which are O(log V) and it also looks at all of the roads connected to a city, which is at most O(E), where E is the number of roads. Thus, the time complexity of the entire algorithm is O(V+E)*log V.

For the space complexity, we used an adjacency list, which has a complexity of O(E). Additionally, the previous and distances maps each have a space complexity of O(V), so the total space complexity is O(V+E).

For the justifications, because there are only 20 cities, the time complexity of O(V+E)*log V is effecient. If there were more cities, the

efficiency would get worse and worse.

4.6. Results (See the results page)

Our results did match the results.txt file.

4.7. Conclusion
Did your results match the output of the test program? Did you run into problems in your implementation? How did you overcome those problems?

Yes, our results did match the provided samples. We did run into a few problems in our implementation. One problem we ran into is that the constructor and destructor of the CityGraph class were compliling incorrectly, so we had to create a separate CPP file to define them, whereas the rest of the member functions were defined direclty in the .h file. Additionally, we initially programmed it to accept user input after being compiled and ran, but had to change it so that the user provided the input directly while running the executable.

4.6 Results

FI to GG

```
Enter From City Code: FI
Enter To City Code: GG
From City: IRWIN, population 4120, elevati
on 932
To City: GRPVE, population 913330, elevati
on 952
The shortest distance from IRWIN to GRPVE
is 24
Through the route: IRWIN -> PARKER -> GRPV
E
```

PD to PM

```
Enter From City Code: PD
Enter To City Code: 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
```

PM to PD

```
Enter From City Code: PM
Enter To City Code: 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
```

SB to PR

```
Enter From City Code: SB
Enter To City Code: 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
```