# Introduction

For the final project of my Discrete Structures class, I was tasked with creating a simple program that would take a CSV file, consisting of information about the distances between several major US cities, and turn it into a weighted graph. I was then to use that graph to determine the shortest distances, in miles, between any two cities. The functional requirements that I have deduced are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** |  | **Functional Requirement** |  | **Value** | **Stakeholder** |
| FID001 | **I want to** | Read and parse the CSV file | **so that** | I can easily use the information contained within | student |
| FID002 | **I want to** | Convert the parsed information into a weighted graph | **so that** | An algorithm may be applied to said information | student |
| FID003 | **I want to** | Apply an algorithm to the data to determine shortest paths | **so that** | The main purpose of the assignment will be met | instructor |
| FID004 | **I want to** | Create a functional user interface | **so that** | The user can interact with the program | instructor |

I believe that fulfilling the above functional requirements will complete the parameters of the assignment in a satisfactory manner.

# Analysis

The first task to tackle in this project was reading the provided CSV file into the system and parsing it [FID001]. To read the file, FileReader was employed, which fed the file into a BufferedReader, and finally read the data into an instance the Scanner class. The file was read line by line and parsed using a regular expression (regex) to remove quotes (“) and commas (,). The parsed data was divided into two arrays: valueArray[][], to contain all of the connections and distances between cities (or edge and weights), and labelArray[], to contain the names of all the cities (vertices) and their original indices.

Since the proposed graph would be a very dense one, consisting of every vertex having an edge to every other vertex, it was decided that an adjacency matrix implementation would be the wisest choice. Using a simple Graph class, valueArray was iterated through and a weighted edge was added for each set of connected vertices [FID002]. From here, the data was ready to be processed by a shortest path algorithm.

Given that the edge weights were all positive integers, research dictated that it would be best to employ Dijkstra’s Algorithm to determine the shortest paths between vertices [FID003]. Dijkstra’s Algorithm has a time cost of ***O(V2)***, where ***V*** represents the number of vertices. The implementation of the algorithm, which was modified to fit the project’s needs, charted the distance and shortest paths between a source vertex and all other vertices in the graph. It then printed those distances and paths to the console.

The algorithm works by starting at the chosen source node and checking the distance to each neighboring node. It stores those distances, and then moves out to neighbors of neighbors, each time checking distances and adding them to paths back to the source. Whenever a shorter path is found, it replaces the current path in storage. Once every node has been visited and all distances calculated, the shortest paths have all been found and stored.

The final task was to implement a rudimentary user interface using the Scanner class [FID004]. This involved printing a prompt to the screen to get the city that the user would like to use as the source node. Using this string, the appropriate vertex index was found and entered into the algorithm to complete execution of the program.

# Conclusion

Upon running the program, the CSV file will be processed and a list of the cities in the graph will be printed to the screen. A command prompt will ask which city the user would like to use as the source node. The user can type in any of the cities mentioned, and the algorithm will run, printing all of the distances and shortest paths from the source city to every other city in the list.

This was a difficult project for me, but it was rewarding when it all started coming together. I have a habit of making things more complicated than they need to be. I spent many hours trying to figure out how to implement a graph (with vertex and edge objects), and it turned out I kind of already had the graph structure that I needed in the valueArray. I actually wanted to remove the the Graph class, as it seemed like the valueArray was exactly the same structure, but for some reason that I couldn’t surmise, I was getting an index out of bounds error when I tried. Also, there are several improvements I would like to make to the program, such as cleaning up Main, like you said, and building a function to allow the user to choose the source node and a single destination node. Overall, as frustrating as it was at times, this was my favorite project that we’ve done. Looking forward to next term.

**Software Repository**: <https://github.com/cellson7170/CSVtoGraph>