```
 /**********************************************************************
* Project Report Template
* Project 3 (Map Routing), ECE368
**********************************************************************/
```

Name: David Pimley
Login: dpimley

The way I solved this problem was to utilize Dijkstra's shortest path algorithm with a few optimizations. The optimized version of this algorithm runs in O((|E| * |V|) * log(|V|)) time due to use of a binary heap. Essentially the algorithm starts out with a heap of distances all set to INT_MAX, with the source being set to 0. The algorithm will then extract the minimum from this heap and traverse each of the neighbors and choose the one with the minimum distance to traverse next. This happens until the destination is extracted at which point the path is printed along with the distance of the target node which represents the minimum distance to that node. A few other optimizations I tried to implement were to find a better square root method and to also use a Fibonacci heap instead of a binary heap. The benefit of using a better square root method is that it's almost wasteful to use the generic Math.h method because it's more precise than I needed. To help this along I tried to implement the fast inverse square root method (John Carmack) to speed the process along. However, I couldn't get this method to be faster than the Math.h function. **Therefore I will need my code to be compiled with the Math.h flag -lm.** Second, one way to also increase speed would be to also implement a Fibonacci heap which has faster methods for inserting and decreasing the key value.

In the original roadCA.txt my program would segfault, however, in the new input file my program runs fine. One limitation of my program is the use of directed / undirected graphs. This program will only work on undirected graphs.

The only help I utilized on this project was from Wikipedia. I utilized their resources to figure out when to stop the running of the Dijkstra's algorithm, i.e. when the target was found.
There were two serious problems that I encountered while doing this project. The first of which was the speed at which my program was running. Originally to run usa10.txt it would take my program around 16 seconds to execute. My issue was the way my decrease key function was working for my heap. It was searching through the entire heap for the right index. To solve this I stored the heap index of each vertex in the adjacency list so that the index could be found in O(1) time. This made my program go from running in 16 seconds on usa10.txt to 0.35 seconds on usa100.txt. The only drawback to this solution was that I was utilizing more memory to store the heap index. The second issue I encountered was that the roadCA.txt was crashing my program. I believe the reason this was happening was because the input coordinates were too large. I believe this to be the case because the new roadCA.txt works just fine.

I really enjoyed this project because it was a practical implementation of an algorithm that is used in the real world today and led me to try and think of abstract solutions to speed up my code.