README

Charles Bennett
EE450 Section 2
USC ID: 8311964095

Format of Messages Exchanged: All messages exchanged are given delimited by spaces.
The client relays all messages without additional formatting. Once server A has combined information from client request and map the beginning of the files exchange remains constant Like this for the example 51539607552 A 4 15 1.18.
Then, the format is node name + space + value (distance or delay).
Code Used: All networking code is taken from beej networking guide. Makefile tutorial I used was from https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html.

Furthermore, I used pseudocode for the Djinsktra's algorithm from Wikipedia https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm as well as information from the book.
.
Code only works if valid inputs are given as depicted by project. Little to no exception handling is done.

All three files except for client use UDP classes which initiate and deconstruct UDP connections and maintain the use of a single socket. The same is true for TCP on AWS.

TCP/UDP are managed by class based structures in the three files because this allows for convenient organization of pertinent information i.e. sockfds. This also ensures that no additional child processes are created by tracking connections. When the classes are destructed, the socket is torn down.

AWS
Relays input from client back to server A then from server A to server B then to client. The only meaningful processing that is done is for printing.
Server A
Hashes map names and constructs graph to find shortest paths and the relays this back to AWS. Maps are stored in a map of maps structure which maps map ids to a class with all the information of the map and all the required methods. This class computes shortest distance given a request class input and gives all outputs needed to send.
Server B
Does calculations and relays back to AWS. This is code is also centralized around one class which processes request and computes all the distances.

```
struct sockaddr getPort;
socklen_t len = sizeof(getPort);
getsockname(sockfd, &getPort, &len);
```

And you can get your port from

(struct sockaddr_in *)&getPort)->sin_port

This piece o code is taken from piazza.