→ Function of Main Driver

→ Per project specifications, the Graph will be brute forced within our code. We have designed the Graph class to follow in line with the method by having a *allPossiblePaths()* function which where all the main driver code will be located. This means the main driver itself will involve only instantiating a new Graph and then calling that specific function.

→ Responsibility of Classes

→ It has been determined that responsibility of the project different functions will be shared among whole classes rather than induvial functions. The *Distance* and *City* classes will be completed by **Araam** and the *Graph* class in addition to the *main driver* will be completed by **Fatima.**

→ Timeline

→ The timeline as per the *City* and *Distance* classes will be completed by **May 3rd at 11:59 PM**. The *Graph* class and *main driver* will be completed by **May 3rd at 11:59 PM**.

→ Collaborative Technologies

- → **Git:** Git will be used within the project to properly collaborate within the project and to keep a recorded history of each individual code.
- → **Discord:** Discord will be used within the project to promote voice and or text communications.
- → Visual Studio Code: VS Code feature to edit the project live promotes the use of debugging issues that may arise of the program in a collaborative manner. Instead of sending multiple Git pushes for a single problem, remoting into a partner's computer to tackle the problem would be far easier and efficient.

→ UML Diagram of Classes/Functions

Graph - m_cities[5]: City - m_distances: Distance + Graph(): constructor + Graph(City * cities, Distance * distances): constructor + Graph(Graph & other): constructor - -Graph(): destructor - getNumberCities(): int + getNumberEdges(): int + getEdgeWeight (City start. City end): int + isEmpty(): bool + edgeExists(): bool + addDistance(City start, City end, int mileage): void - removeDistance(City start, City end): void

Distance

- clear(): void

+ depthFirstTraversal(City start): void

+ allPossiblePaths(): void

+ operator=(Graph & rhs): Graph &

- m_city1: City
- m_city2: City
- m_mileage: int
- + Distance(): constructor
- + Distance(City city1, City city2, int mileage): constructor
 - + Distance(Distance & other): constructor
 - ~Distance(): constructor
 - + getCity1(): City
 - + getCity2(): City
 - + getMileage(): int
 - + setCity1(City & city1): void
 - + setCity2(City & city2): void
 - + setMileage(int mileage): void

City

- m_city: String
- + City(): constructor
- + City(String * city): constructor
- + City(City & city): constructor
 - ~City(): destructor
 - + getCity(): String
- + operator=(City & rhs): City &
- + setCity(String & city): void