The memory of my reaction when the code finally produced the required output and satisfied my test cases is still vivid in my mind. However, it was not an easy road. There were a lot of semantic and syntax errors along the way. Initially, I wanted to just write one single method that takes a start city and start country and a destination city and destination country and then returns either true or false depending on whether the path was found or not. Without even thinking of a pseudocode, I started writing lines of code in a method I named testing. I was able to get the method to work but l had many edge cases that l did not consider. Therefore, I had to go back to the drawing board. I started out by writing down the necessary data l would need from the files given. After doing that l defined how I would want to represent my data and the approach that I chose was to create object to represent the necessary data from each file. For example, for airport data, I needed the longitude, latitude, and the airport code so I created an airport class. The next stage was to create UML diagrams for the classes and how they were to relate to each other. I learned that the best approach whenever I encounter a problem, is to plan what I want to do first rather than planning while writing the code. The second solution I came with far much better than the first. The first solution was an implementation using Breadth first search, and the second solution I implemented it with A\* algorithm. After running various test cases, I found out that the A\* algorithm was more optimal in terms of distance. In calculating the evaluation function for my A\*, I used the haversine function to calculate the distance from the latitude and longitude. The formula for the evaluation function, f, was f = h + d, where h is the heuristic and d is the distance. The heuristic was the function from the current airport to the destination and the distance was the distance travelled from the start city to the current city. The evaluation function helped me to eliminate other routes which would be very far from the destination, hence making the search tree smaller. I also encountered a lot of exceptions and error and most of them l tried to handle them, and I also considered edge cases like when a person enters a city that is not in the airports file. However, there were some errors encountered while implementing my solution. As such, I introduced the OutOfMemory and ArrayIndexOutOfBounds Exceptions that handled cases where it took a longer time to find the expected solution and cases were a record had its length greater than what the file holds due to extra commas respectively. In total, I have learnt how important it is to take into consideration the design and data structures used in writing efficient codes and implementing optimal solutions especially in this individual project.