A high-level approach to solve this challenge was first to ensure the code was modularised. Before programming, I drew UML diagrams to help me know the classes, fields, methods and algorithms I had to implement. I created classes: Airline class, Airport, Route, Node class. The airport class has a method which reads from the airports.csv file and returns a hashmap of countries and their cities as keys and their values as an ArrayList of available airports IATA codes. Similarly, the Route class has a method which reads from the routes.csv file and returns a hashmap of airport IATA codes as their keys and an ArrayList of a data type programmed by me called Route. The Route class encapsulates the details about a particular route into an object. Its attributes include the airline code, destination airport code, and the number of stops. The node class represents the form of the search tree, where each node consists of the state (airport name), parent Node, and path cost.

I represented the data in an adjacency matrix data to find the optimal route. This was to ensure I had an efficient runtime since hashmaps support constant time lookups. The choice of the data structures resulted from the algorithm I intended to use for my searching. The search algorithm used was the Breadth First Search. The algorithm was optimal since the flight cost would be the same between flights. From this challenge, I have learnt how to make decisions about choosing the right data structures either to ensure my algorithms have an efficient run-time complexity. Through using the data structures, I have learnt how to work with some java programming APIs such as Queues, Hashmaps, ArrayLists and Hashsets.

I have also learnt how to incorporate good programming style into my programming. For example, writing modularised code (separating code into methods and classes), handling exceptions, writing good comments and docstrings, and principles of Object Oriented Programming such as Encapsulation. Finally, by completing the challenge, I have come to understand that data might not always be clean as you want it to be; hence it is necessary to identify all edge cases and handle them when programming.