“Fly Imaraat” is a new airline originating from Dubai, and they are flying to more than 100 cities around the world. The task is to create a graph network (using Adjacency Maps, as done in labs) for their flights destined towards different cities in North America, Central America and South America; and then use it accordingly. Some of these destinations have direct, non-stop flights originating from Dubai, whereas many others have different layovers in between. These layovers may or may not be in the specified region.

The data set is defined in a CSV file called flight\_network.csv, which contains 4 columns: Origin City, Destination City, Flight Duration (in minutes), and Flight Distance (in miles). The given dataset will produce two separate graphs:

1. One graph will have vertices containing the city name, and edges will have flight duration between the two connected cities as their weight.
2. The second graph will also have vertices as cities, and edges will have flight distance as their weights.

Q1) Define the following function to create the two flight networks, i.e. weighted graphs as defined above:

* ***create\_flight\_network*** that takes two inputs: **filename** which is the CSV file of the dataset, and **option** as an integer:
  + **1** for creating graph with flight duration as the edge weights
  + **2** for creating graph with flight distance as the edge weights.

The function then returns the respective graph at the end

Q2) Define the following functions to get information on connected cities:

* ***get\_flight\_connections*** that takes three inputs: **graph** which is the graph created in Q1, **city** which is a string representing the vertex, and **option** which is a character: “i” for inbound flights, and “o” for outbound flights. The function then finds all the connected vertices based on the option selected, and returns the connected airport cities in the form of a list. If there is no connections to and from this city, the result will be an empty list.
* ***get\_number\_of\_flight\_connections*** that takes three inputs: **graph** which is the graph created in Q1, **city** which is a string representing the vertex, and **option** which is a character: “i” for inbound flights, and “o” for outbound flights. The function then calculates the total number of flights depending on the option selected, and returns that number

Q3) Define the following functions to get information regarding the connected flights in the graph:

* ***get\_flight\_details*** which takes three inputs: **graph** which is the graph created in Q1, **origin** which is a string representing the origin city, and **destination** – also a string representing the destination city. The function returns the time or distance to go from **origin** city to the **destination** city, depending on the graph being used. If the origin city is not in the graph, it returns None. If the destination is not connected to the origin city, it returns -1

Q4) Adding new flights between existing destinations

* ***add\_flight*** which takes four inputs: **graph** which is the graph created in Q1, **origin** which is a string representing the origin city, **destination** which is representing the destination city, and **weight** which is an integer representing the flight duration or flight distance depending on the graph sent as argument. This creates a direct connection between the **origin** and **destination** city, and adds the weight to the connection. If any of the origin or destination cities are not present in the network, it should print a message stating that the particular city not accessed by the flight network. If the connection already exists, it just updates the **weight** of the connection

Q5) Adding a new airport with at least one connection with one of the other existing destinations

* ***add\_airport*** which takes four inputs: **graph** which is the graph created in Q1, **city** which is a string representing the new destination being added to the existing graph, **destination** which is a string representing the city with which the new airport will have at least one connection, and **weight** which is an integer representing the flight duration or flight distance for that city with the **destination**. If the **city** is already in graph, just print a message stating that the airport already exists. Otherwise, it creates a new entry in the graph for the new airport

Q6) Getting secondary connecting flights from the given city for a multi-city itinerary

* ***get\_secondary\_flights*** which takes two arguments: **graph** which is the graph created in Q1, and **city** for which the secondary connections need to be checked. The function will first look for the immediate connections to that **city**, and from those immediate connections, it will get a list of all the connecting flights. The function returns a list of cities further connected to the immediate connection. Please note, we are only looking at the outbound connections. If the city does not exist, it returns None

Q7) Getting the intersecting airports for flights from two different cities

* ***counting\_common\_airports*** which takes three arguments: **graph** which is the graph created in Q1, **cityA** which is a string representing one of the two cities, and **cityB** which is also representing the other city. The function counts the number of common airports in the two cities’ connections and returns the count. If none common, it returns 0

Q8) Removing any flights between airports

* ***remove\_flight*** which takes three arguments: **graph** which is the graph created in Q1, **origin** which is a string representing the origin city, and **destination** which is a string representing the destination city. The function simply removes the connection between the two cities. If any of the origin or destination cities are not present in the network, it should print a message stating that the particular city not accessed by the flight network. Don’t forget that these fights exist in both the graphs.

Q9) Removing the airport and its associated flights

* ***remove\_airport*** which takes two arguments: **graph** which is the graph created in Q1, and **city** which is a string representing the city being removed from the flight network. The function removes the airport city and all its connections from the graph if it exists. If the given city is not present in the network, it should print a message stating that the particular city not accessed by the flight network. Don’t forget that these fights exist in both the graphs.

Q10) Traversing the flight network:

* ***find\_all­\_routes*** which takes three arguments: **graph** which is the graph created in Q1, **origin** which is a string representing the origin city, and **destination** which is a string representing the destination city. The function then finds all the flight routes connecting the origin city to the destination, as nested list. If the origin or destination city does not exist in the network, it returns None

Q11) Flight route details:

* ***find\_number\_of\_layovers***which takes three arguments: **graph** which is the graph created in Q1, **origin** which is a string representing the origin city, and **destination** which is a string representing the destination city. The function then finds all the possible layovers when travelling from origin city to the destination city, in the form of a list sorted in ascending order. Non-stop flights will have 0 layover. If the origin or destination city does not exist in the network, it returns None