Data Structures and Algorithms CW2 FlyPlanner F28DA

Date of Submission: 30/3/21

Name: Varun Senthil Kumar

Registration Number: H00332328

Degree Programme: BSc Computer Science (Hons)

Campus: Dubai Campus

<u>HelloFlyPlanner</u>

The HelloFlyPlanner class is used to represent direct flights and least cost connections. It represents the given set of airports as a graph.

A string of cities and a scanner object to take in user input is initialized at the beginning. Then a SimpleDirectedWeightedGraph is initialized with String as vertex and DefaultWeightedEdge as edge. The string holds the city name while the DefaultWeightedEdge holds the flight route & price.

The cities are added as vertices, and the flight routes along with the price are added as the edges of the graph. A table is printed for the user to see all the possible routes along with their prices. String.format() is used to align the display in the table.

The user is then asked to enter the starting city and the destination city. 'While' loops are used to check if the city entered by the user is present in the graph. If the user inputs the same city as origin and destination, he is asked to enter a different destination airport.

A DijkstraShortestPath graph is initialized to store the graph, while a GraphPath is created using the Dijkstra graph.

By iterating through the all the edges in the graph path, we print out the itinerary.

Part A

In part A, I have implemented the functions in AirportImpl.java, FlightImpl.java and TripImpl.java classes.

The FlightImpl.java class has a constructor that takes in a string of data, AirportImpl object 'to' and AirportImpl object 'from' as arguments. The string of data holds the flightCode, fromGMTime, toGMTime and cost.

The AirportImpl.java class has a constructor that only takes a string of data as arguments. The string of data holds the airport code and airport name.

The TripImpl.java class has a constructor that takes a List of FlightImpl objects, since the trip might need more than one flight.

The FlyPlannerImpl.java is the class that creates the functionality of the graph. The populate() method adds the vertices and edges for the graph. The airports are the vertices ,flights are the edges, and the costs are the edge weights of the SimpleDirectedWeightedGraph.

The leastCost() uses a DijkstraShortestPath graph to calculate a path between two airports with the least cost. The algorithm makes use of the edge weight of the graph, which is the cost. It returns a TripImpl object which stores the edge list of the path.

The leastHop() is similar to the leastCost(), but instead of DijkstraShortestPath, I have used a AsUnweightedGraph so as to remove the edge weights. Therefore, the path will be calculated based on the shortest path.

Both leastCost() and leastHop() will be overloaded if the user chooses to exclude certain airports in the trip, by passing a List of airports to be excluded as an argument. leastCost() is implemented in the main function.

Part B

For part B, I have implemented the functions in

AirportImpl.java,FlightImpl.java,TripImpl.java and FlyPlannerImpl.java which were previously left unfinished.

Helper function to convert time from hours and minutes format to minutes has been implemented in TripImpl.java and FlyPlannerImpl.java. Both helper functions are the same, with the code being reused within the project.

The leastCostMeetUp() takes in two arguments, departure, and arrival airport. Then a DijkstraShortestPath graph is used to plot a GraphPath between the two

travellers. All the vertices are then added to an airport list, where the vertices are the airports between the two travellers. A middle index is then found, which is the meeting point for both the travellers.

The leastHopMeetUp() is similar to the leastCostMeetUp() but uses a AsUnweightedGraph instead. This removes the edge weights, which is the cost. Thus, we get a path with the least hops.

The leastTimeMeetUp() is not yet complete.

directlyConnected(),setDirectlyConnected(),setDirectlyConnectedOrder() and getBetterConnectedInOrder() are either incomplete or not attempted due to my limited understanding of these method's functionality.

Limitations

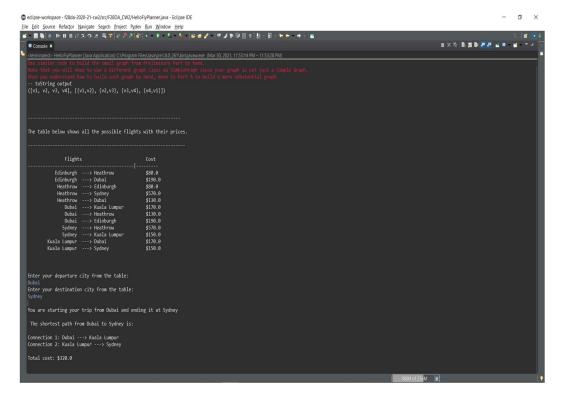
There are a few limitations to my code.

Firstly, the main function does not print the departing and arrival airport while displaying the itinerary. It prints the other data correctly, but I am not able to print the airport names. I am getting a nullPointerException when I try to print the names.

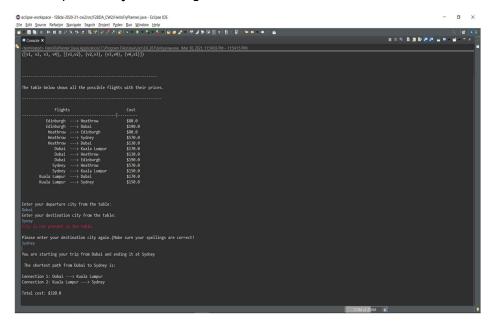
Secondly, in the main function, while printing a trip with airports to be excluded, the user can only exclude one airport.

Testing

Only 6 tests pass in the provided test file. I have added my own tests in the test file, where 1 test fails. The test that fails is testing the directly Connected().



HelloFlyPlanner.java running



HelloFlyPlanner.java with erroneous usage



FlyPlannerMain running with erroneous usage. It also shows a trip with one airport excluded.