Flight Itinerary

COURSEWORK 2

Tomasz Mosak | Data Structures & Algorithms | 27/11/2017

Implementation

During the implementation of 'part a' within the program, getting used to some of the newly found functions took a bit of time. However, once a level of understanding was reaching it wasn't too much of a hassle to create the vertices and edges to then be passed into the Dijkstra shortest path which allows the user in the end to see the cheapest and quickest way from 1 airport to another.

Part B was a tricky one, having to first pass information from the FlightsReader class into a newly made class which stores full details of each flight in a string to be used later in the outputting procedure. Then having to extend the DefaultWeightedEdge to allow flight data to be set as the weight via the creation of a new class called RouteEdge. Finally having to pull it all together in the FlightItinerary by firstly creating a graph of all the routes available from each airport by calling the extention class RouteEdge which then triggers the Flight class allowing the population of the graph with required information. Then the user is prompted with starting and finishing destination and the magic of Dijkstra happens again but this time, the user is presented with much more data about the flight due to the Flight class storing all that data.

Part C, isn't very different from Part B, however, instead of finding the cheapest path, this time the program focuses on making the LEAST amount of changeovers possible. This is done in a very similar way to how the cheapest path is found but instead of using the weight (price) we set the graph to have no weight and perform the same calculations therefore finding the real shortest path. The exclusions extension to the program was implemented via finding the 'RED' airports and marking them as unusable(roadblocks). Unfortunately, due to time constrains, the submission lacks the meetup function.

Representation

Part A ----->

```
Enter your starting location
edinburgh
Enter destination
sydney
Shortest (i.e. cheapest) path:
1. Edinburgh  -> Dubai
2. Dubai  -> Kuala Lumpur
3. Kuala Lumpur  -> Sydney
Cost of shortest (i.e cheapest) path = £510.00
```

Part B ----->

```
Enter your starting location
Warsaw
Enter destination
Rome
CHEAPEST ROUTE:
Leg     Leave               At          On          Arrive              At
1       Warsaw WAW          1512        LH5048      Munich MUC          1607
2       Munich MUC          0756        LH4732      Rome FCO            0850
Total travel time: 17 hrs, 38 min
Time in air: 1 hrs, 49 min
Total Cost: £133
Total Changeovers: 1
```

Part C ----->

```
Enter your starting location
edinburgh
Enter destination
paris
LEAST CHANGEOVERS:
Leg      Leave            At          On          Arrive          At
1        Edinburgh EDI    1626        BA5985      London LHR      1709
2        London LHR       1115        BA0549      Paris ORY       1147
Air Time: 1 hrs, 15 min
Total Time: 19 hrs, 21 min
Total Cost: £114
Total Changeovers: 1
```

Testing

For Part A, testing is very trivial as all the user must do is choose one of the airports available from part A as a starting point and another as their Destination. Then by the power of Dijkstra shortest path calculations, the user is presented with the quickest and cheapest path to their desired destination.

Here are some tests for Part A:

```
Enter your starting location
Edinburgh
Enter destination
Edinburgh
Shortest (i.e. cheapest) path:
Cost of shortest (i.e cheapest) path = £.00
```

This is what happens if the user inputs the same starting point and destination. The calculation doesn't work because they are already at their destination!

```
Enter your starting location
sydney
Enter destination
heathrow
Shortest (i.e. cheapest) path:
1. Sydney  -> Kuala Lumpur
2. Kuala Lumpur  -> Dubai
3. Dubai  -> Heathrow
Cost of shortest (i.e cheapest) path = £450.00
```

Another test example, however, this time even though the user only has 5 destinations to choose from, they still must take 3 trips! Might be the 'cheapest' route but most likely not very time effective.

For Part B of the program, the test was very similar to part A as the major differences don't change the way, the user would interact with the program. Only noticeable differences are the amount of information presented to the user and the large variety of airports available to choose from.

Here are some tests for Part B:

In this test case we can see that the formatting becomes a little wonky when an airport with 2 words is chosen, however, that's due to string.format being told to include a certain amount of \t between each output to create the 'table' effect.

```
Enter your starting location
AKL
Enter destination
OSL
CHEAPEST ROUTE:
Leg     Leave                   At          On          Arrive              At
1       Auckland AKL            1914        UA9926      San Francisco SFO           0622
2       San Francisco SFO           0342        UA2137      London LHR          1226
3       London LHR              2208        BA3503      Oslo OSL            2346
Total Time: 52 hrs, 32 min
Time In Air: 21 hrs, 30 min
Total Cost: £1114
Total Changeovers: 3
```

Another interesting test case is where we take a very short flight that everyone would just take directly from Edinburgh to London, however, due to the fact we specify to the program that we want the cheapest route, it offers a route that would take just under 2 days!

```
Enter your starting location
EDI
Enter destination
YXU
CHEAPEST ROUTE:
Leg     Leave                   At          On          Arrive              At
1       Edinburgh EDI           0941        LH7356      Newark EWR          1507
2       Newark EWR              0311        UA1396      Detroit DTW         0413
3       Detroit DTW             1408        UA0141      Chicago ORD         1451
4       Chicago ORD             2124        UA1905      London YXU          2211
Total Time: 36 hrs, 30 min
Time In Air: 7 hrs, 58 min
Total Cost: £448
Total Changeovers: 4
```

For Part C, due to the fact only the main implementation and the first extension was fully implemented, the test was very similar to what Part B looks like due to the lacking content which allowed meetups and exclusions.

Here are some tests for Part C:

The way part C is implemented, the program tries to find the longest way that a plane can take the user towards their destination to minimize changeovers, however, therefore costing the user more in the end. The same starting location and destination ran in part B gives us a shorter airtime, total time and cost!

```
Enter your starting location
SIN
Enter destination
YVR
LEAST CHANGEOVERS:
Leg     Leave               At          On          Arrive              At
1       Singapore SIN       0739        BA1461      London LHR          1759
2       London LHR          1755        BA1176      Vancouver YVR       0230
Air Time: 18 hrs, 55 min
Total Time: 42 hrs, 51 min
Total Cost: £1148
Total Changeovers: 2
```

In this example, the least changeover path finding method is more effective if the user was under time constrains as its quicker by over 20 hours!

```
Enter your starting location
DEN
Enter destination
ZAG
LEAST CHANGEOVERS:
Leg     Leave               At          On          Arrive              At
1       Denver DEN          1828        UA4985      Frankfurt FRA       0343
2       Frankfurt FRA       0052        LH5539      Zagreb ZAG          0151
Air Time: 10 hrs, 14 min
Total Time: 11 hrs, 33 min
Total Cost: £521
Total Changeovers: 2
```

Summary

Overall, the flight itinerary coursework allowed for a deeper understanding of how jgrapht works and its possibilities. With the ability to create, populate, exclude and calculate paths efficiently within massive graphs, the implementations are infinite. It was nice seeing something that can and is used in real life working so beautifully.