Data Structures Report

Coursework 2 — Flight Planner

Graph Implementation

Name: Jose Carlos Cerqueira Fernandes

Student Number: H00324200

Contents

Overall Implementation

The goals in parts A and B were successfully achieved but part C was partially achieved. The details about the partial achievement of part C are described in its section.

The major change in the layout of the part B and C was the creation of a public “FlightPlanner” function “userQueriesPartsBC”. This function handles the class scanner object, by instantiate it and then close it when the queries are finished. This class runs other private functions related to the required tasks that get the input and print the results, for part B and part C. The implementation of this function makes use of all the local variables and functions of the “FlightPlanner” class and avoids code reusing in the “FlyingPlannerMainPartBC” class.

The initial output is the listing of all graph airports. The user can then copy the airport codes and put them in the queries.

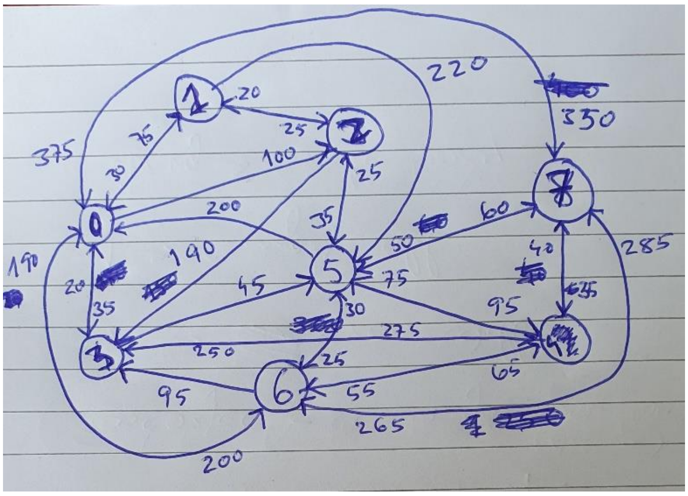
```
The following airports are used
( BKI ) - Kota Kinabalu International Airport
( SAN ) - San Diego International Airport
( LSE ) - La Crosse Municipal Airport
( GSP ) - Greenville Spartanburg International Airport
( BRO ) - Brownsville South Padre Island International Airport
( PVG ) - Shanghai Pudong International Airport
( WLG ) - Wellington International Airport
( PAH ) - Barkley Regional Airport
( TRD ) - Trondheim Airport Vaernes
( DUJ ) - DuBois Regional Airport
( DMM ) - King Fahd International Airport
( CID ) - The Eastern Iowa Airport
( ILM ) - Wilmington International Airport
( KFS ) - Kastamonu Airport
( PIH ) - Pocatello Regional Airport
( CCU ) - Netaji Subhash Chandra Bose International Airport
( BTS ) - M. R. Stefanik Airport
```

Overall testing data description

The testing data used to test the part B and C, is a custom graph with Airport as vertices and Flights as edges. This data was created previously in a piece of paper, will all the connection and weights already.

The reason behind the use of this small testing data is to have a graph easier to understand due to its dimension but also due to its display on the piece of paper. The custom graph was very insightful to spot the program's performance and its limitations.

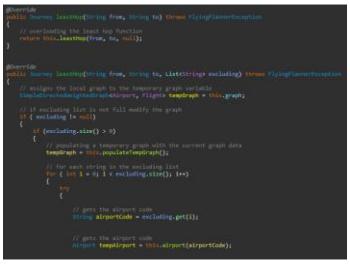
```java
@Before
public void initialize() {
    fi = new FlyingPlanner();
    
    HashSet<String[]> airportsSet = new HashSet<String[]>();
    HashSet<String[]> flightsSet = new HashSet<String[]>();
    int length = 8;

    for(int i = 0; i < length; i++)
    {
        String code = "APT" + i;
        String location = "location" + i;
        String airportName = "Airport number " + i;
        String[] tempAirport = { code, location, airportName};
        airportsSet.add(tempAirport);
    }

    String[] flight0 = {"F00","APT0","1245","APT1","1320","75"}; flightsSet.add(flight0);
    String[] flight1 = {"F01","APT1","0550","APT0","0700","30"}; flightsSet.add(flight1);
    String[] flight2 = {"F02","APT0","1020","APT2","1400","100"};  flightsSet.add(flight2);
    String[] flight3 = {"F03","APT0","2300","APT6","0400","200"};  flightsSet.add(flight3);
    String[] flight4 = {"F04","APT1","1600","APT5","1830","220"};  flightsSet.add(flight4);
    String[] flight5 = {"F05","APT2","1500","APT3","1900","190"};  flightsSet.add(flight5);
    String[] flight6 = {"F06","APT2","1500","APT5","1900","35"};  flightsSet.add(flight6);
    String[] flight7 = {"F07","APT2","1500","APT1","1900","20"};  flightsSet.add(flight7);
    String[] flight8 = {"F08","APT1","1500","APT2","1900","25"};  flightsSet.add(flight8);
    String[] flight9 = {"F09","APT3","1500","APT0","1900","20"};  flightsSet.add(flight9);
    String[] flight10 = {"F10","APT0","1500","APT3","1900","35"};  flightsSet.add(flight10);
    String[] flight11 = {"F11","APT3","1500","APT4","1900","275"};  flightsSet.add(flight11);
    String[] flight12 = {"F12","APT4","1500","APT5","1900","75"};  flightsSet.add(flight12);
    String[] flight13 = {"F13","APT4","1500","APT7","1900","40"};  flightsSet.add(flight13);
    String[] flight14 = {"F14","APT4","1500","APT6","1900","55"};  flightsSet.add(flight14);
    String[] flight15 = {"F15","APT5","1500","APT2","1900","25"};  flightsSet.add(flight15);
    String[] flight16 = {"F16","APT5","1500","APT7","1900","60"};  flightsSet.add(flight16);
    String[] flight17 = {"F17","APT5","1500","APT6","1900","25"};  flightsSet.add(flight17);
    String[] flight18 = {"F18","APT5","1500","APT0","1900","200"};  flightsSet.add(flight18);
    String[] flight19 = {"F19","APT6","1500","APT3","1900","95"};  flightsSet.add(flight19);
    String[] flight20 = {"F20","APT6","1500","APT4","1900","65"};  flightsSet.add(flight20);
    String[] flight21 = {"F21","APT6","1500","APT5","1900","30"};  flightsSet.add(flight21);
    String[] flight22 = {"F22","APT6","1500","APT7","1900","285"};  flightsSet.add(flight22);
    String[] flight23 = {"F23","APT6","1500","APT0","1900","190"};  flightsSet.add(flight23);
    String[] flight24 = {"F24","APT7","1500","APT5","1900","50"};  flightsSet.add(flight24);
    String[] flight25 = {"F25","APT7","1500","APT6","1900","265"};  flightsSet.add(flight25);
    String[] flight26 = {"F26","APT7","1500","APT4","1900","35"};  flightsSet.add(flight26);
    String[] flight27 = {"F27","APT7","1500","APT0","1900","375"};  flightsSet.add(flight27);
    String[] flight28 = {"F28","APT0","1500","APT7","1900","350"};  flightsSet.add(flight28);

    fi.populate(airportsSet, flightsSet);
}
```



Part B

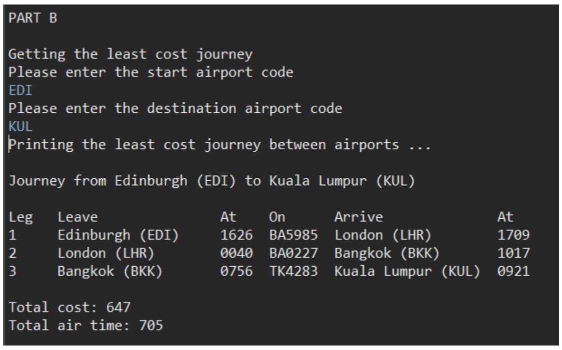
Implementation

The implementation in the part B, which part C is also built on, makes use of function overload to reduce the amount of code in the program. There is 3 examples of this technique, the “populate”, “leastCost” and the “leastHop” functions.



Correct use

The Part B has a very good performance and its displaying matches the styling given as example. The total cost and air-time are also correct and reveal great precision. The goal of getting the cheapest journey was accomplished and it is very stable and safe, as it does not return any unpredictable result.

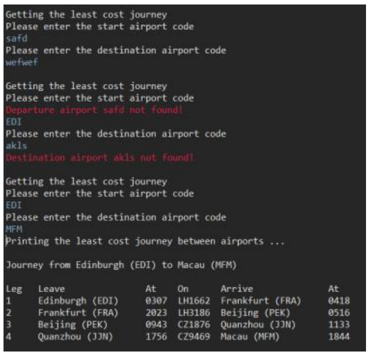


```
PART B

Getting the least cost journey
Please enter the start airport code
EDI
Please enter the destination airport code
KUL
Printing the least cost journey between airports ...

Journey from Edinburgh (EDI) to Kuala Lumpur (KUL)

Leg   Leave            At    On      Arrive             At
1     Edinburgh (EDI)  1626  BA5985  London (LHR)       1709
2     London (LHR)     0040  BA0227  Bangkok (BKK)      1017
3     Bangkok (BKK)    0756  TK4283  Kuala Lumpur (KUL) 0921

Total cost: 647
Total air time: 705
```

```
PART B

Getting the least cost journey
Please enter the start airport code
ANC
Please enter the destination airport code
DXB
Printing the least cost journey between airports ...

Journey from Anchorage (ANC) to Dubai (DXB)

Leg   Leave              At    On      Arrive             At
1     Anchorage (ANC)    0359  DL8005  Minneapolis (MSP)  0844
2     Minneapolis (MSP)  0118  UA1180  Toronto (YYZ)      0230
3     Toronto (YYZ)      1106  EK2442  Dubai (DXB)        2342

Total cost: 920
Total air time: 1113
```

Erroneous Use

It can handle bad input that does not match with any airport. It will check both airport Strings.



```
Getting the least cost journey
Please enter the start airport code
safd
Please enter the destination airport code
wefwef

Getting the least cost journey
Please enter the start airport code
Departure airport safd not found!
EDI
Please enter the destination airport code
akls
Destination airport akls not found!

Getting the least cost journey
Please enter the start airport code
EDI
Please enter the destination airport code
MFM
Printing the least cost journey between airports ...

Journey from Edinburgh (EDI) to Macau (MFM)

Leg   Leave            At    On      Arrive             At
1     Edinburgh (EDI)  0307  LH1662  Frankfurt (FRA)    0418
2     Frankfurt (FRA)  2023  LH3186  Beijing (PEK)      0516
3     Beijing (PEK)    0943  CZ1876  Quanzhou (JJN)     1133
4     Quanzhou (JJN)   1756  CZ9469  Macau (MFM)        1844
```

Known limitations

This part has no known limitations.

Part C

Implementation

The part C implementation relies on the previous highly on the "leastCost" and "leastHop" functions. An example is the "leastHopMeetUp" that uses the "leastHop" (excluding variant) to get the least hop journey, excluding specific cases.

```
@Override
public String leastHopMeetUp(String at1, String at2) throws FlyingPlannerException
{
    // gets airport 1 and airport 2
    Airport airport1 = this.airport(at1);
    Airport airport2 = this.airport(at2);

    // gets the a set of directly connected airports to airport 1
    Set<Airport> directlyConnectedToA1 = this.directlyConnected(airport1);

    // if both airports are directly connected
    if ( directlyConnectedToA1.contains(airport2) )
    {
        throw new FlyingPlannerException("These two airports are already directly connected");
    }

    // It gets the list with the union airports that are not completely directly connected
    List<String> except = this.airportExceptCodes(at1, at2);

    // Checks is the path was found
    boolean pathFound = false;

    // Stores the meet up airport
    String meetup = null;

    // it will loop while the path was not found
    while ( !pathFound )
    {
        // gets the first journey
        // avoiding certain airports contained in the except list
        Journey j1 = this.leastHop(at1, at2, except);

        // gets the stops list of first journey
        List<String> stps1 = j1.getStops();

        // removes the origin stop, from the first journey
        stps1.remove(0);
        // removes the destination stop, from the first journey
        stps1.remove(stps1.size() - 1);

        // gets the second journey (back)
        // avoiding certain airports contained in the except list
        Journey j2 = this.leastHop(at2, at1, except);
```

Correct use

It provides great performance when calculating the least cost and hops. It returns accurate data relatively to flights, cost, hops and with precise air, connection and total journey times.

```
PART C

Getting the least cost journey
Please enter the start airport code
EDI
Please enter the destination airport code
KUL
Printing the least cost journey between airports ...

Journey from Edinburgh (EDI) to Kuala Lumpur (KUL)

Leg   Leave              At    On      Arrive              At
1     Edinburgh (EDI)    1626  BA5985  London (LHR)        1709
2     London (LHR)       0040  BA0227  Bangkok (BKK)       1017
3     Bangkok (BKK)      0756  TK4283  Kuala Lumpur (KUL)  0921

Total cost: 647
Total hops: 3
Total air time: 705
Total connection time: 1750
Total journey time: 2455
```

```
Getting the least hop journey
Please enter the start airport code
KUL
Please enter the destination airport code
EDI
Printing the least hop journey between airports ...

Journey from Kuala Lumpur (KUL) to Edinburgh (EDI)

Leg   Leave              At    On      Arrive            At
1     Kuala Lumpur (KUL) 1928  AF0457  Paris (CDG)       0811
2     Paris (CDG)        0702  AF8390  Edinburgh (EDI)   0806

Total cost: 690
Total hops: 2
Total air time: 827
Total connection time: 1371
Total journey time: 2198
```

The meet up places are also given without any evident issues. It will be discussed further issues in the limitations section.

Erroneous Use

The part C also handles the wrong airport codes cases specified in the previous erroneous use section.

When looking for the least cost and hop meet up airports, the program will handle the error when the user types airports that are bi directionally connected (connected directly on both ways). This was set up that way because there is no point on getting a meet up airport in this case.

```
Getting the least cost meetup
Please enter the start airport code, for traveller 1
EDI
Please enter the start airport code, for traveller 2
LHR

Getting the least cost meetup
Please enter the start airport code, for traveller 1
These two airports are already directly connected
```

```
Getting the least hop meetup
Please enter the start airport code, for traveller 1
EDI
Please enter the start airport code, for traveller 2
LHR
These two airports are already directly connected
```

Known limitations

The part C has partially met the goals specified for its full success. The “FilePlanner class” has no implementation for the “leastTimeMeetUp”, “setDirectlyConnectedOrder” and “getBetterConnectedInOrder”, which are needed for the conclusion of part C.

The other very significant limitation happens when the least cost and hop journey have the same amount of hop. The least hops function implementation assigns a constant value to not distinguish edges, including distinguish them by price. The result is not wrong but it may retrieve a more expensive journey, even though it returns a journey with the least amount of hops.

```
Getting the least hop journey
Please enter the start airport code
EDI
Please enter the destination airport code
MEL
Printing the least hop journey between airports ...

Journey from Edinburgh (EDI) to Melbourne (MEL)

Leg   Leave                 At    On      Arrive                At
1     Edinburgh (EDI)       2113  AF1174  Paris (CDG)           2217
2     Paris (CDG)           0712  AF3093  Ho Chi Minh City (SGN)1843
3     Ho Chi Minh City (SGN)0917  QF1388  Melbourne (MEL)       1627

Total cost: 1316
Total hops: 3
Total air time: 1185
Total connection time: 1409
Total journey time: 2594
```

```
PART C

Getting the least cost journey
Please enter the start airport code
EDI
Please enter the destination airport code
MEL
Printing the least cost journey between airports ...

Journey from Edinburgh (EDI) to Melbourne (MEL)

Leg   Leave                 At    On      Arrive                At
1     Edinburgh (EDI)       0307  LH1662  Frankfurt (FRA)       0418
2     Frankfurt (FRA)       2158  LH6123  Hong Kong (HKG)       0708
3     Hong Kong (HKG)       0553  CX1971  Melbourne (MEL)       1323

Total cost: 981
Total hops: 3
Total air time: 1071
Total connection time: 2425
Total journey time: 3496
```

The “leastCostMeetup” and “leastHopMeetup” although handle erroneous user input and return a reasonable meeting airport, it points to the next airport from one of the source vectors instead of pointing at the middle of the graph. In this case both users will save money or travelling to meet, but one of them will benefit more than the other instead of equally sharing the benefits of meeting eachother. The Airport given as a meet up point is part of both sides’ shortest paths, so it is indeed a good meeting point but it will be always be the next airport to the first traveller airport.

The evidence is clear in the meet up Edinburgh/Melbourne (both ways) and considering also the individual shortest paths of Edinburgh and Melbourne between themselves. The shortest paths (least cost/hops) from Edinburgh to Melbourne and the reverse shortest path (very last screenshot) do not share any common edges, or middle vertices. The meet up places from Edinburgh to Melbourne and the reverse meet up is also not any of the previous shortest paths vertices.

According to all these evidences, it is safe to assert that London Heathrow and Hong Kong Airports are part of the common shortest paths and the meet up point might be one of this places or a middle one. It was attempted to get the middle value of one of the common path but the result did not match with the existing test cases.

```
Getting the least cost meetup
Please enter the start airport code, for traveller 1
EDI
Please enter the start airport code, for traveller 2
MEL
Printing the least cost meetup airport ...
London (LHR)

Getting the least hop meetup
Please enter the start airport code, for traveller 1
EDI
Please enter the start airport code, for traveller 2
MEL
Printing the least hop meetup airport ...
London (LHR)
```

```
Getting the least cost meetup
Please enter the start airport code, for traveller 1
MEL
Please enter the start airport code, for traveller 2
EDI
Printing the least cost meetup airport ...
Hong Kong (HKG)

Getting the least hop meetup
Please enter the start airport code, for traveller 1
MEL
Please enter the start airport code, for traveller 2
EDI
Printing the least hop meetup airport ...
Hong Kong (HKG)
```

```
Getting the least cost journey
Please enter the start airport code
MEL
Please enter the destination airport code
EDI
Printing the least cost journey between airports ...

Journey from Melbourne (MEL) to Edinburgh (EDI)

Leg   Leave              At    On      Arrive              At
1     Melbourne (MEL)    0808  CZ0140  Guangzhou (CAN)     1613
2     Guangzhou (CAN)    1224  CZ0463  Amsterdam (AMS)     2101
3     Amsterdam (AMS)    0755  KL0808  Edinburgh (EDI)     0848

Total cost: 945
Total hops: 3
Total air time: 1055
Total connection time: 1865
Total journey time: 2920

Getting the least hop journey
Please enter the start airport code
MEL
Please enter the destination airport code
EDI
Printing the least hop journey between airports ...

Journey from Melbourne (MEL) to Edinburgh (EDI)

Leg   Leave              At    On      Arrive              At
1     Melbourne (MEL)    2154  QF2641  Shanghai (PVG)      0625
2     Shanghai (PVG)     0659  AF4837  Paris (CDG)         1656
3     Paris (CDG)        0702  AF8390  Edinburgh (EDI)     0806
```