Assessed Individual Coursework 2 — Flight Itinerary

1 Overview

Your task is to use a graph to represent airline data, and to support searching. You should carefully test all of the code you write, generating new data files as necessary, and include the result of your tests in the report.
The coursework aims to reinforce your understanding of module material, specifically the following learning objectives:

- Gain an understanding of a range of graph classes and their use to represent realistic data.
- Gain further experience in object-oriented software engineering with a non-trivial class hierarchy: specifically selecting an appropriate class; reusing existing classes; extending existing classes.
- Using generic code: reusing generic graph classes, and parameterising a class with different types.
- You will also gain general software engineering experience, specifically downloading and using Open Source software, using a general method for a specific purpose, and issues with reusing existing code.
- Gain further experience with Java programming.

Information and guidelines

- This coursework should be done **individually**.
- Parts of the coursework should be demonstrated in the lab. We recommend that you tackle each part of the coursework in turn over a number of weeks to spread the load. The demonstration of all parts must be completed by the last lab session you attend in week 11.
- You should submit your report and all source code of your programs on Vision by 5:30PM on Thursday 24th of November, 2016.
- This coursework is worth 20% of this module (50% of the total coursework mark).

2 Coursework Parts

Preliminary Part: Installation and get familiar with JGraphT

You will need to download a personal copy of the Open Source JGraphT graph library. Having a single central copy of the library for all students on the course would save space, but it is a valuable experience for you to use an Open Source code repository.

Graph Package Download

- Download the Open Source JGraphT graph library by following the instructions on:
  `http://jgrapht.org/`
  The following instructions are for `jgrapht-1.0.0` on a Unix machine using bash.

- Decompress and extract the tarball (this will create a 20M `jgrapht-1.0.0` directory), e.g.

```
$ tar zxvf jgrapht-1.0.0.tar.gz
```

- Delete the tarball to save 4.8M (6.3M for the zip file), e.g.

```
$ rm jgrapht-1.0.0.tar.gz
```

Setup for Command Line Compilation and Execution

- Add JGraphT core JAR file to your class path by adding the following commands at the end of your `.profile` file in your home directory.

  In the following command you should replace `yourlogin` with *your login*, and `yourpath` with the *directory path* to where you have installed JGraphT. The first line adds JGraphT core JAR to your class path, the second adds JGraphT demo files your class path.

```
export CLASSPATH=/u1/cs2/yourlogin/yourpath/jgrapht-1.0.0/lib/↵
   jgrapht-core-1.0.0.jar:$CLASSPATH
export CLASSPATH=/u1/cs2/yourlogin/yourpath/jgrapht-1.0.0/↵
   source/jgrapht-demo/src/main/java/org/jgrapht/demo:↵
   $CLASSPATH
```

  For Jane Doe whose login is `jd42` and who is working on the assignment in the directory `F28DA/CW2`, the commands are:

```
export CLASSPATH=/u1/cs2/jd42/F28DA/CW2/jgrapht-1.0.0/lib/↵
   jgrapht-core-1.0.0.jar:$CLASSPATH
export CLASSPATH=/u1/cs2/jd42/F28DA/CW2/jgrapht-1.0.0/source/↵
   jgrapht-demo/src/main/java/org/jgrapht/demo:$CLASSPATH
```

- Execute your new `.profile`, e.g.

```
$ source ~/.profile
```

Setup for Eclipse Integration

To use and integrate JGraphT in Eclipse, you need to Configure the Java Build Path of your project. You will need to apply the following changes in Libraries:

- Add the external JAR `jgrapht-core-1.0.0.jar`
- Once the JAR is added, you can attach its sources by deploying its menu and edit Source attachment to point to the external location
  `/u1/cs2/yourlogin/yourpath/jgrapht-1.0.0/source/jgrapht-core/src`
  This will make the sources of JGraphT directly available within Eclipse for documentation and debugging purposes.
- Similarly, you can add the documentation by editing Javadoc location to be
  `/u1/cs2/yourlogin/yourpath/javadoc/`.
  This will make the documentation of JGraphT directly available within Eclipse.

Demonstration Programs

Go to the JGraphT source directory, e.g.

```
$ cd ~/yourpath/jgrapht-1.0.0
$ cd source/jgrapht-demo/src/main/java
```

Compile and execute the 1st demo program:

```
$ javac org/jgrapht/demo/HelloJGraphT.java
$ java org/jgrapht/demo/HelloJGraphT
```

Execute other demo programs, e.g. `PerformanceDemo` - takes several minutes!

View Javadoc

You will need to use the JGraphT Javadoc to locate appropriate classes and methods. Browse http://jgrapht.org/javadoc/

Editing & Compiling Graph Programs

Edit `HelloJGraphT.java` (`org/jgrapht/demo/HelloJGraphT.java`), by adding a new edge to one of the graphs. Compile and run your revised program Congratulations, you are ready to start writing graph programs.

Viewing Example Graph Programs

Often the easiest way to write a program is to reengineer, i.e. copy and modify, a similar program. More example programs are available in the test directory

```
$ cd ~/yourpath/jgrapht-1.0.0
$ cd source/jgrapht-core/src/test/java
```

Demonstration (week 9): Editing & Compiling must be demonstrated during week 9, and you should aim to complete Part A.

Part A: Representing Direct Flights

Write a program to represent the following direct flights with associated costs as a graph. For the purpose of this exercise assume that flights operate in both directions with the same cost, e.g. Edinburgh ↔ Heathrow denotes a pair of flights, one from Edinburgh to Heathrow, and another from Heathrow to Edinburgh. *Hint:* Flights are directed, i.e. from one airport to another, and weighted by the ticket cost, hence use the JGraphT `SimpleDirectedWeightedGraph` class. You should display the contents of the graph (and may omit the weights).

Flight	Cost
Edinburgh ↔ Heathrow	£80
Heathrow ↔ Dubai	£130
Heathrow ↔ Sydney	£570
Dubai ↔ Kuala Lumpur	£170
Dubai ↔ Edinburgh	£190
Kuala Lumpur ↔ Sydney	£150
Edinburgh ↔ Frankfurt	£90
Sydney ↔ Auckland	£120
Rio de Janeiro ↔ New York	£430
New York ↔ Santiago	£320

Part B: Least Cost Connections

Extend your program to search the flights graph to find the least cost route between two cities consisting of one or more direct flights.
Hint: use methods from the `DijkstraShortestPath` class to find the route. A possible interface for your program might be one where you suggest a start and an end city and the cost of the entire route is added up and printed.

```
The following airports are used:
       Edinburgh
       Heathrow
       ...
 Please enter the start airport
    Edinburgh
 Please enter the destination airport
    Kuala Lumpur
 Shortest (i.e. cheapest) path:
 1. Edinburgh -> Dubai
 2. Dubai -> Kuala Lumpur
 Cost of shortest (i.e. cheapest) path = £360
```

Demonstration (week 10): Part A and B must be demonstrated during week 10, and you should aim to complete at least Part C.

Part C: Additional Flight Information

Start a **new program** operating on a graph containing the same flights as in part A, but including the following information about each flight. The flight number, e.g. BA345; the departure time; the arrival time; the flight duration; and the ticket price, e.g. £100. All times should be recorded in 24 hour hhmm format, e.g. 1830.
Use your imagination to populate your graph with sensible flights.

Part D: Itinerary

Use the additional flight information to print itineraries for least cost journeys in a format similar to the following example. The key aspects are

1. A sequence of connecting flights (with least cost)
2. A total cost for the route

An example itinerary for parts E & F might resemble:

```
Itinerary for Edinburgh to Toronto
Leg Leave         At   On    Arrive        At
1   Edinburgh     0530 FZ345 Dubai         1100
2   Dubai         1230 EK657 Kuala Lumpur  1730
3   Kuala Lumpur  1800 QF652 Sydney        2130
4   Sydney        2200 BA216 Auckland      2330
Total Journey Cost    = £630
Total Time in the Air = 930
```

Demonstration (week 11): Part C and D must be demonstrated during week 11, and you should aim to complete at least Part E.

Part E: Itinerary Duration

Extend your program to calculate the total time in the air, i.e. the sum of the durations of all flights in the itinerary.
Hint: you will need to write functions to perform arithmetic on 24 hour clock times.

Part F: Alternative Extensions

Attempt at least three of the following extensions to your second program.

1. Airline itineraries record arrival and departure times in local time for the destination airport, also in 24 hour format. Calculate the total journey time as the sum of the flight durations plus the sum of the changeovers.
2. Extend your program to calculate journey time for journeys that span more than one day, e.g. takeoff at 21:30 and arrive at 04:00.
3. Extend your program to allow connections between two flights only if the arrival time of the first flight is between 1 and 5 hours of the departure time of the second flight.
4. Extend your program to locate routes with the fewest number of changeovers.

   Hint: use a standard graph traversal algorithm, available in JGraphT.
5. Extend your program in some other way. Carefully describe the extension in your report.

Your report must clearly indicate which extensions you implemented and must demonstrate them.

3 Coding Style

Your mark will be based partly on your coding style. Here are some recommendations:

- Variable and method names should be chosen to reflect their purpose in the program.
- Comments, indenting, and whitespaces should be used to improve readability.
- No variable declarations should appear outside methods ("instance variables") unless they contain data which is to be maintained in the object from call to call. In other words, variables which are needed only inside methods, whose value does not have to be remembered until the next method call, should be declared inside those methods.
- All variables declared outside methods ("instance variables") should be declared private (not protected) to maximize information hiding. Any access to the variables should be done with accessor methods.

4 Submission

Submit on Vision an archive containing your report and of the Java source code of your programs.

Source code of programs The electronic version consists of all the files of code you have written and the electronic version of your brief report. Your code should contain comments that clearly identify parts A, B etc. For example:

```
/*******************************************************
 * Part B: Least Cost Connections                      *
 *******************************************************/
```

Report The report should be brief (max. 7 pages, excluding the appendices), prepared using the text processor of your choice, and containing the following sections.

Section 1 Testing

This section presents a description of test data and testing outcomes, and includes reasons why test data was chosen.

1.1 Program 1, Part A: Representing Direct Flights
1.2 Program 1, Part B: Least Cost Connections
1.3 Program 2, Part C: Additional Flight Information
1.4 Program 2, Part D: Itinerary
1.5 Program 2, Part E: Itinerary Duration
1.6 Program 3, Part F: Alternative Extensions

In subsection 1.6 you should identify which extensions you have implemented, and give a demonstration of each.

Section 2 Evaluation

As part of your submission you should include a reflection on your coursework (max. 1 page). Please make it clear if parts of your code do not fully work. If some of your work falls into this category then include some analysis of the problem and how you would tackle it given more time.

5 Marking Scheme

Your **overall mark** will be computed as follows.

- Program 1, Parts A and B — 20 marks
- Program 2, Parts C, D and E — 25 marks
- Program 3, Part F — 20 marks
- Coding style — 10 marks
- Report with clear structure, content and appropriate length. — 10 marks
- Demonstration of your program during the labs in week 9 to 11. — 15 marks

Your coursework is due to be submitted by 5:30PM on Thursday 24th of November, 2016. If you hand in work late, without extenuating circumstances, 10% of the maximum available mark will be deducted from the mark awarded for each day late.