Programming Assignment #3
Airline Travel Scheduler

Problem Statement

There are numerous application domains where problems and tasks are formulated by graphs. The purpose of this programming assignment is to practice graph theory by applying a *shortest path algorithm* to a real-life problem. You will implement Dijkstra's single-source shortest path algorithm to develop an airline travel scheduler. The airline travel scheduler will find an itinerary (*i.e.*, a sequence of flights) that allows one to depart from an origin airport and arrive at a destination airport at the earliest possible time (*i.e.*, the shortest travel time).

Requirements & Implementation

- You will be given two tables as input: an *airport table* and a *flight table*. The airport table consists of a set of airports each of which a minimum connecting time is associated with. The flight table consists of a set of flights, and for each flight, the following information: departure and destination airports, departure time, and arrival time. The airline travel scheduler will be built based on the two tables. We will use the flight information obtained from real-world commercial carriers (*e.g.*, Hawaiian Airlines and Lufthansa Airlines).
- An airline traveler (or customer) submits to the scheduler (1) an origin airport, (2) a final destination airport, and (3) his/her earliest departure time from the origin airport.
- Given the information from a traveler, the scheduler will find a sequence of flights that allow him/her to arrive at the final destination at the earliest possible time when departing from the origin *at* or *after* the earliest departure time. Minimum connection times at intermediate airports (*i.e.*, excluding the origin and destination airports) should be observed. Airline travelers are willing to take a next-day flight if no same-day flight is available or the next-day flight allows them to arrive at the final destination earlier. Do not assume that an itinerary will always be found for any given origin and destination, because it may not exist at all.
- Airport names are encoded in the standard 3-letter symbols (*e.g.*, ICN for Seoul-Incheon). There may be an airport with outgoing or incoming flights only. All flight times are given in the 24-hour notation, also referred to as military time (*e.g.*, 1330 for 1:30pm).[1] Assume that the international community has grown tired of different time zones and abolished the whole concept.
- You are to write Java classes that implement four ADTs, namely, `Airport`, `Flight`, `Itinerary` and `Planner` whose skeletons are provided at eTL. These skeletons merely contain two or three methods including constructors and print methods. Copy the skeletons to your working directory, and provide implementations for the methods defined in the skeleton classes. Care should be taken when implementing the print methods so that the output of your code matches exactly what is provided in the test cases. You can add more methods as you wish, but are not allowed to remove or change any method defined in the skeleton classes. A main class `MainAir.java` is also provided at eTL.
- Your code is expected to be an Airline Travel Scheduler *of production quality*. This implies that your scheduler should be both efficient and scalable. You should be able to produce an itinerary quickly for hundreds of airports and thousands of flights without increasing the initial memory allocation of JVM running on a Linux platform (*i.e.*, with neither `Xms` nor `Xmx` option).
- Follow the directions given in the first assignment handout regarding 'no `main()` method.'

[1] Some commercial carriers use military times incorrectly. For example, an arrival time at 2436 must be interpreted as 00:36am.

Grading

This assignment is worth 15 percent of your final grade. In writing the code, correctness is the most important attribute, followed by efficiency. General grading guidelines are:

10 points	Program compiles without errors and is on the right track,
0-30 points	The Scheduler works correctly on small test cases,
0-40 points	The Scheduler works correctly on large test cases,
0-20 points	The Scheduler works efficiently on large test cases.

To evaluate the efficiency of your class implementation, we will measure the time spent on planning and flight scheduling. If it is more than 10 (100 or 1000) times slower than my own implementation, you will lose 10% (50% or 100%) of the credits reserved for efficiency.

The late submission and regrading policies are described in the course syllabus.

Submission

Refer to the first programming assignment for submission instructions.

Due date

The programming assignment is handed out on Monday Nov. 01, 2021, and due by 11pm on Sunday Nov. 28, 2021.