Project 4 - Airline
CmpE 250, Data Structures and Algorithms, Fall 2023

TAs: Suzan Ece Ada, Kutay Altıntaş
SAs: Yiğit Kağan Poyrazoğlu, Emre Kılıç, Yunus Emre Özdemir

Due: December 24, 2023, 23:55 Strict

1 Introduction

You are working for the Lounge Aviation Inc., which is an aviation conglomerate with countless subsidiaries including regional, continental and international airline companies. Your job is to find an algorithm that gives the successive flights with the minimum total cost to an airport in a given time frame.

2 Definitions

2.1 Airport

Where planes depart, park and land. Every airport has a unique *AirportCode*, an associated *AirfieldName*, a *latitude*, a *longitude* and a *parkingCost*. Information on Airports will be provided in <airports-csv> file, refer to I/O Files section for further explanation.

AirportCode: A unique identifier for the airport.
Example: "SAW"

AirfieldName: Name of the airfield which the airport belongs, further explained in subsection 2.2.
Example: "AsiaMinor"

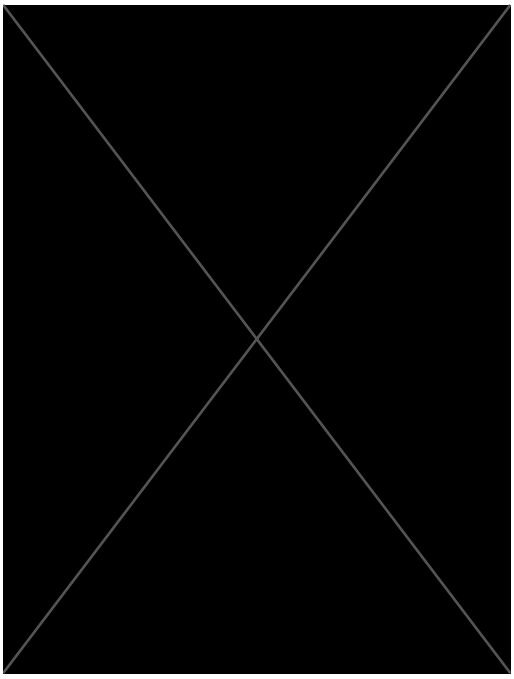
latitude: First decimal number to identify the location of the airport.
Example: 40.89833

$$-90 \leq latitude \leq +90$$

longitude: Second decimal number to identify the location of the airport.
Example: 29.30916

$$-180 \leq longitude \leq +180$$

parkingCost: The cost of parking at the airport for 6 hours.
Example: 142.72358



2.2 Airfield

An Airfield is a group of airports that share the same weather conditions. At a given *AirfieldName* and *Time*, there is a corresponding *WeatherCode*. This information is given in the <weather-csv> file, refer to I/O Files section for further explanation.

AirfieldName: A unique identifier for the airfield.
Example: "AsiaMinor"

Time: A Unix time stamp in long integer format.
Example: 1701864924

WeatherCode: Represents the weather condition, can be decoded to calculate the Weather Multipliers. Refer to Weather Multiplier subsection for further information.
Example: 30

$$0 \leq w_c \leq 31, w_c \in \mathbb{N}$$

2.3 Weather Multiplier

Weather multipliers are multipliers used in the calculation of the cost to fly from an airport to another one. They represent the effects of weather conditions on the cost of flight.

Weather multiplier of an airport at a given time can be inferred from the *WeatherCode* w_c that is associated with the *Airfield* the airport belongs to and given *Time*.

Let B_w, B_r, B_s, B_h, B_b be variables that represent whether the corresponding weather conditions are present: Wind, Rain, Snow, Hail, Bolt. These values are determined by their corresponding bits of the 5-bit unsigned binary representation of the w_c.

Example: *WeatherCode*: 18. Binary: 10010. $B_w = 1$, $B_r = 0$, $B_s = 0$, $B_h = 1$, $B_b = 0$. Windy with hail.

Example: *WeatherCode*: 25. Binary: 11001. $B_w = 1$, $B_r = 1$, $B_s = 0$, $B_h = 0$, $B_b = 1$. Windy and rainy with bolts.

Example: *WeatherCode*: 0. Binary: 00000. $B_w = 0$, $B_r = 0$, $B_s = 0$, $B_h = 0$, $B_b = 0$. Clear.

Then the weather multiplier W can be calculated with the formula:

$$W = (B_w * 1.05 + (1 - B_w)) * (B_r * 1.05 + (1 - B_r)) * (B_s * 1.10 + (1 - B_s)) * (B_h * 1.15 + (1 - B_h)) * (B_b * 1.20 + (1 - B_b))$$

2.4 Distance

Distance between the airports is given by The Haversine Formula.

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{lat_2 - lat_1}{2}\right) + \cos(lat_1) * \cos(lat_2) * \sin^2\left(\frac{lon_2 - lon_1}{2}\right)}\right)$$

Take $r = 6371$ which is the radius of the earth in kilometers.
Hint: Store your latitude and longitude in double data type for correct significance.
Hint2: Remember to convert degrees to radians if you want to use standart Math library.

2.5 Flight Durations

Three main subsidiaries of Lounge Aviation are Lounge Turkey, Lounge Continental and Lounge International. These subsidiaries use different planes with different flight durations.

Lounge Turkey uses Carreidas 160. It's flight durations can be calculated using: *(d is distance)*

$$T(d) \begin{cases} 6 \text{ hours} & \text{if } d \leq 175 \\ 12 \text{ hours} & \text{if } 175 < d \leq 350 \\ 18 \text{ hours} & \text{if } 350 < d \end{cases}$$

(a) Carreidas 160

(b) Orion III

(c) Skyfleet S570

(d) T-16 Skyhopper

Figure 2: The aircrafts

Lounge Continental uses Orion III in Asia. It's flight durations can be calculated using: *(d is distance)*

$$T(d) \begin{cases} 6 \text{ hours} & \text{if } d \leq 1500 \\ 12 \text{ hours} & \text{if } 1500 < d \leq 3000 \\ 18 \text{ hours} & \text{if } 3000 < d \end{cases}$$

Lounge Continental uses Skyfleet S570 in Europe. It's flight durations can be calculated using: *(d is distance)*

$$T(d) \begin{cases} 6 \text{ hours} & \text{if } d \leq 500 \\ 12 \text{ hours} & \text{if } 500 < d \leq 1000 \\ 18 \text{ hours} & \text{if } 1000 < d \end{cases}$$

Lounge International uses T-16 Skyhopper. It's flight durations can be calculated using: *(d is distance)*

$$T(d) \begin{cases} 6 \text{ hours} & \text{if } d \leq 2500 \\ 12 \text{ hours} & \text{if } 2500 < d \leq 5000 \\ 18 \text{ hours} & \text{if } 5000 < d \end{cases}$$

2.6 Possible Flights

There is a set of possible flights from each airport to other airports. These are given in the <directions-csv> file, for further information, refer to the I/O Files & Execution section.

2.7 Flight Cost

Let d be the distance between the departed airport and the landing airport, W_D weather multiplier of the departed airport at the time of departing, W_L weather multiplier of the landing airport at the time of landing. Cost of the flight is given by the formula:

$$c_f = 300 * W_D * W_L + d$$

2.8 Parking Cost

A parking operation is waiting at an airport for 6 hours with a cost of *parkingCost*. Every airport has a *parkingCost* value. Successive parking operations can also be made without a limit. For example, 4 successive parking operations would be equivalent to a 24 hours of parking. Keep in mind that the cost is $4 * parkingCost$ in this example.

2.9 Total Cost

Total cost of a sequence of successive flight and park operations is the sum of flight costs and parking costs. For the calculation of flight and parking costs, refer to the previous subsections.

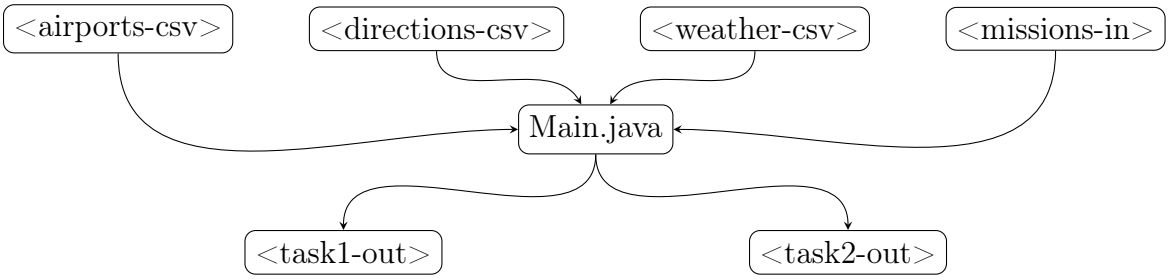
2.10 Mission

Every mission consists of 4 elements: *AirportOrigin*, *TimeOrigin*, *AirportDestination* and *Deadline*. Starting from the *AirportOrigin* at *TimeOrigin*, you must come up with a sequence of successive possible flight and park operations to reach the *AirportDestination* before the *Deadline* with the minimum total cost. Please note that some missions might be impossible to accomplish before the deadline. For further information, refer to the I/O Files & Execution section.

2.11 Time

Every flight and park operation takes time and time marches forward with every operation. Don't forget to take this into account when accessing weather data or taking the *Deadline* into account.

3 I/O Files & Execution



3.1 CSV File Format

A CSV (Comma-Separated Values) file is a simple and widely used text-based format for storing and exchanging tabular data. In a CSV file, each line represents a row of data, and individual values within a row are separated by commas. The first row often contains headers that define the names of the columns.

3.2 <airports-csv>

This is a .csv file. In this file, you are given the essential attributes of the Airports. For more detailed descriptions of these attributes refer to the Airport subsection. Here is an example of what the <airports-csv> file will look like:

AirportCode,AirfieldName,Latitude,Longitude,ParkingCost
OBBS,BH,25.918399810791016,50.590599060058594,977
OBKH,BH,26.034533,50.524544,862

3.3 <directions-csv>

This is a .csv file. In this file, you are given the possible directions of two different airports. Please note that the lines are directed. That means, in a given row, you can reach the airport under the to column from the airport under the from column, but there is no guarantee that the vice versa is true. Here is an example of what the <directions-csv> file will look like:

from,to

OJHR,EG-0004
OJHR,OEDM

3.4 <weather-csv>

This is a .csv file. In this file, you are given the the weather conditions at specific times and airfields. For more detailed information about these attributes, refer to the Airfield section. Here is an example of what the <weather-csv> file will look like:

AirfieldName,Time,WeatherCode
AD,1680296400,14
AD,1680318000,27

3.5 <missions-in>

This is an .in file. In the first line of this file, you are given the plane model which the missions will be accomplished with. In the subsequent lines, you will be given *AirportOrigin*, *AirportDestination*, *TimeOrigin* and *Deadline* informations about the mission sequentially. Here is an example of what the <mission-in> file will look like:

Orion III
JO-0002 VDKK 1680620400 1680771600
TW-0034 ZMOT 1682586000 1682607600

3.6 Main.java

This is the entry point of your program.
Your program should be able to accomplish two tasks and write their outputs to their respective files <task1-out> and <task2-out>.

3.6.1 <task1-out>

Task 1 is a simplified version of the Task 2.

In this task, you are expected to find successive possible flight operations from *AirportOrigin* to *AirportDestination*. Assume there is no deadline, and all flights are happening at *TimeOrigin* for each mission. More specifically, all departures and landings are happening at the *TimeOrigin*. Which means, there is no need to increment time after flights.

Keep in mind that there is no park operations or impossible flights in this task.

<task1-out> is an .out file. Every line of the <task1-out> file should be the solution of the corresponding mission in the <missions-in> file. The solution consists of the airports

in the sequence and the total cost. For example, if you started from "IST", flew to "BER", and then flew to "JFK" which was the destination, and accumulated a total cost of 747.54, you should write:

IST BER JFK 747.54000

3.6.2 <task2-out>

In this task, your program should be able to read missions from <mission-in> and find a sequence of successive possible flight and park operations starting from the *AirportOrigin* at *TimeOrigin* to reach the *AirportDestination* before the *Deadline* with the minimum total cost. <task2-out> is an .out file. Every line of the <task2-out> file should be the solution of the corresponding mission in the <missions-in> file.

If a mission can be accomplished, that is, there is a sequence of flight or park operations that starts from the *AirportOrigin* at *TimeOrigin* and reaches the *AirportDestination* before the *Deadline*, you should write the airports in the sequence, the park operations in between and the total cost. For example, if you started from "IST", flew to "BER", parked there for 12 hours, and flew to "JFK" which was the destination, and accumulated a total cost of 435, you should write:

IST BER PARK PARK JFK 435.00000

If a mission can't be accomplished, you should write:

No possible solution.

4 File Structure & Submission

Your code will be graded automatically. Therefore, it is important that you follow the submission instructions.

First, all of your source files should be collected under **Project4/src**. Then, you should zip the **Project4** folder and rename it to <**student_id**>**.zip**. This zip file will be submitted through Moodle. Name of your main class **must** be **Main.java** as described above.

Note: Use **zip**, not rar or 7z. Put only **.java** files under the src, not .class files or any other file.

Your program will be compiled with the command below:

```
javac Project4/src/*.java -d ./
```

The input and output files can be in any folder. Design your code to accept the full path for file arguments. Your program will be run with the command below:

```
java Main <airports-csv> <directions-csv> <weather-csv> <missions-in> <task1-out> <task2-out>
```

Make sure your final submission compiles and runs with these commands.

5 Grading

Your code will be tested with Regional, Continental and International cases, which you can think as Small, Medium and Large cases respectively. If your Task 1 implementation produces correct outputs under the corresponding time limits you will get a total of 50 points. If your Task 2 implementation produces correct outputs under the corresponding time limits you will get a total of 50 points. You can get partial grades in these Tasks if you pass a portion of the cases. The input sets we will evaluate your code with may not be the same input sets you will be given to test your code. Keep in mind that any violation of Warnings can also lead to a point reduction.

6 Time Limits

You will be provided corresponding time limits for Task 1 and Task 2 alongside with the inputs. Note that these time limits are based on our implementation on our machines. So if your times are no more than 3-4 times these limits, you should be okay.

7 Warnings

1. Use double-precision for all floating-point values. Write 5 digits after the floating-point in the outputs. Do not use comma as a separator, use decimal points. Examples: $4.56 \rightarrow 4.56000$, $4.567688 \rightarrow 4.56769$, $4.567682 \rightarrow 4.56768$.

2. This is an individual project.

Figure 3: Bro's tired of doin' projects.

3. All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code. Any sign of cheating will be penalized by at least **-100** points at the first attempt and disciplinary action in case of recurrence.

4. You cannot include **any** external libraries to your project. Do not use external CSV parser libraries.

5. You are strongly encouraged to write and use appropriate data structures covered in lectures to achieve decent running times for large inputs. Don't forget that a significant portion of points will be given based on performance with large inputs. Therefore, using correct data structures is an integral part of this project.

6. Parallel Programming is not allowed. Directly using any parallel programming structure such as Stream and Thread will get a **0**.

7. Make sure you document your code with necessary inline comments and use meaningful variable names. Do not over-comment or make your variable names unnecessarily long. This is very important for partial grading.

8. You can add as many files as you can as long as they are in the "src" folder, and your project can be compiled as above. But the entry points of your program should be "Main.java" as described above.

9. Make sure that the white spaces in your output are correct. You can disregard the ones at the end of the line.

10. Please use the discussion forum at Moodle for your questions and check if it has already been answered before.

11. Please do not upload zip bombs.