ASSIGNMENT 5: FLIGHT TRIP PLANNER

Goal: The goal of this assignment is to learn about how to model a real life problem using graphs, and to implement Dijkstra's algorithm.

Problem Statement: We are given information about a set of flights between different cities. You want to go from a city A to a city B and would like to find out the earliest time by which you can reach B. The cities have names which can be strings, for example, "Delhi", "Mumbai", etc. Each flight is described by a flight number, which is again a string (e.g., "AI 102"), source city, destination city, departure time, arrival time. Assume that no flight crosses midnight, i.e., departure time is after midnight and arrival time is before midnight. A query will specify a source city, a destination city and a departure time. For example, it can specify source city as "Delhi", destination city as "Mumbai" and departure time as 10:00 (assume that all times are specified in 24 Hour format, i.e. 00:00 to 23:59). There will be no two cities with the same name. Your goal is to find a sequence of flights starting from the source city ("Delhi" in this example), with the first flight starting at or after 10:00 and reaching Mumbai as early as possible. Note that the trip could involve multiple flights. Also assume that you always want to reach the destination the same day (i.e., before midnight), if it is not possible to do so, you should print an error message.

There is one main complication. You must make sure that the arrival time of a flight in this trip is before the departure time of the next flight in the trip, plus a layover time. Ideal layover time is 2 hours or more. But, if you cannot find an itinerary with 2+ hours layover then you may relax the problem to have a layover time 1.5 hours or more. If you still cannot find an itinerary with 1.5 hours layover, then you may relax the problem further with 1 hour or more layover. No further relaxations are permitted. Note that for our problem (not for real world) "two 1.5+ hour but less than 2 hour layovers" and a combination of "one 1.5+ hour layover and one 2 hour layover" will be considered equivalent with respect to the layover constraint. Of these two itineraries you should output the one that reaches the destination the soonest.

You will need to think about how to transform this problem into one or more shortest path problems. You should use Hash functions (you can use build-in Java hash functions) to map between city names and distinct integers (which may be needed to represent vertices). Moreover, the use of build-in Java Linked list or heap is allowed.

```
public interface TripPlannerInterface{

public void addCity(String cityname); /* Adds a new city with name "cityname". If this function is called multiple times with the same cityname, keep exactly one copy of that city. */

public void addFlight(String flightid, String startcity, String destcity, int departuretime, int arrivaltime) throws IllegalFlightException; /* Adds a new flight, which has flight number given by flightid, and goes from startcity to destcity. Assume that startcity is not the same as destcity.
```

Also, assume that arrival and departure times are given as integers, e.g., 15:00 is given as '1500' or 09:45 is given as 945. You should throw an exception if the city name is not in your database, or if the arrival or departure times are not specified correctly, or if the flightid has been used before. */

public String **findTrip**(String startcity, String destcity, int departuretime) throws IllegalFlightException; /* the parameters are again startcity, destcity, departure time in integer format, and an exception should be thrown if these are not valid. Assume that startcity is not the same as destcity. The function prints the sequence of flight numbers (spaced by commas) starting from the first flight from startcity and the last flight reaching destcity. If no such flight sequence is possible, the function should print "No Flight Exists". */
}

Write a program which implements this interface. You need to think about how to use graphs.

Example 1:

Suppose there are four cities: A, B, C, and D

There is a flight named J1 from A to B with departure time = 1:00 and arrival time = 6:00

There is a flight named J2 from A to C with departure time = 1:00 and arrival time = 2:00

There is a flight named J3 from B to D with departure time = 7:00 and arrival time = 12:00

There is a flight named J4 from B to D with departure time = 8:00 and arrival time = 13:00

There is a flight named J5 from C to B with departure time = 4:00 and arrival time = 5:00

Suppose we want the least time consuming trip from A to D with departure time 1:00

There are four possible routes:

1. J1 to J3
2. J1 to J4
3. J2 to J5 to J3
4. J2 to J5 to J4

Of these routes, the first route is invalid since the layover time between two flights is only 1 hour.

Among the other routes, the least time consuming route is J2 to J5 to J3.

Now, Suppose we want the least time consuming trip from C to D with departure time 4:30

The possible routes are: "J5 to J4" and "J5 to J3". However, the departure time of flight J5 is earlier at 4:00. Therefore, both the routes is invalid.

Input for Example 1:

addCity(“A”);

addCity(“B”);

addCity(“C”);

addCity(“D”);

addFlight(“J1”,“A”,”B”,100,600);

addFlight(“J2”,“A”,”C”,100,200);

addFlight(“J3”,“B”,”D”,700,1200);

addFlight(“J4”,“B”,”D”,800,1300);

addFlight(“J5”,“C”,”B”,400,500);

findTrip(“A”,”D”,100);

findTrip(“C”,”D”,430);

Output for Example 1:

J2,J5,J3

No Flight Exists

Example 2:

Suppose there are five cities: A, B, C, D, and E

There is a flight named J1 from A to B with departure time = 1:00 and arrival time = 2:00

There is a flight named J2 from B to C with departure time = 3:50 and arrival time = 4:30

There is a flight named J3 from B to D with departure time = 3:50 and arrival time = 5:30

There is a flight named J4 from C to E with departure time = 7:00 and arrival time = 9:00

There is a flight named J5 from D to E with departure time = 7:00 and arrival time = 8:00

Suppose we want the least time consuming trip from A to E with departure time 1:00

There are two possible routes:

1. J1 to J2 to J4
2. J1 to J3 to J5

Both these routes are invalid since the layover time between J1 and J2 is 1 hour and 50 minutes only. The same is the case between J1 and J3. Therefore, we relax the condition of layover time to 1 hour and 30 minutes. Now, both these paths are valid.

Among these two routes, J1 to J3 to J5 is the least time consuming route; therefore, it is the best route.

Input for Example 2:

addCity("A");

addCity("B");

addCity("C");

addCity("D");

addCity("E");

addFlight("J1","A","B",100,200);

addFlight("J2","B","C",350,430);

addFlight("J3","B","D",350,530);

addFlight("J4","C","E",700,900);

addFlight("J5","D","E",700,800);

findTrip("A","E",100);

Output for Example 2:

J1,J3,J5

What is being provided?

Your assignment folder contains four java files:

1. IllegalFlightException.java: Defines a custom exception (Do not modify this file)

2.TripPlannerInterface.java: Defines the simulation interface (Do not modify this file)

3. TripPlanner.java : You have to implement all the functions of TripPlannerInterface in this file and you can add your own classes and methods also.

4. TripPlannerTester1.java : File with the main function to test example 1.

5. TripPlannerTester2.java : File with the main function to test example 2.

What to submit?

1. Submit your code in a .zip file named in the format .zip. Make sure that when we unzip your file a folder named should yield, in addition to the Starter folder, a "writeup.txt" file should be produced.

2. In summary if your entry number is 2016CS50393, then you need to submit a zip file 2016CS50393.zip. This zip file on extraction gives a directory structure as follows:

   2016CS50393

   ------- Starter (Contains classes folder and all java files)

   ------- writeup.txt

   You will be penalized for any submissions that do not conform to this requirement.

3. The writeup.txt should have a line that lists names of all students you discussed/collaborated with (see guidelines on collaboration vs. cheating on the course home page). If you never discussed the assignment with anyone say None.

   After this line, you are welcome to write something about your code, though this is not necessary.

Evaluation Criteria

The assignment is worth 6 points. Your code will be autograded at the demo time against a series of tests. Four points will be provided for correct output and two points will be provided for your explanations of your code and your answering demo questions appropriately. **Also, you will be graded based on the**

appropriateness and efficiency of data structures and algorithms. Making suboptimal data structure choices will lead to significant negative penalties.

What is allowed? What is not?

1. This is an individual assignment.
2. Your code must completely be your own. You are not to take guidance from any general purpose code or problem specific code meant to solve these or related problems. Remember, it is easy to detect this kind of plagiarism.
3. You are not allowed to use built-in (or anyone else's) implementations of trees, nodes or other basic data structures. A key aspect of the course is to have you learn how to implement these data structures. You are, however, required to use Java's built in hash functions.
4. You should develop your algorithm using your own efforts. You should not Google search for direct solutions to this assignment. However, you are welcome to Google search for generic Java-related syntax.
5. You must not discuss this assignment with anyone outside the class. **Make sure you mention the names in your write-up in case you discuss with anyone from within the class.** Please read academic integrity guidelines on the course home page and follow them carefully.
6. Your submitted code will be automatically evaluated against another set of benchmark problems. You get a significant penalty if your output is not automatically parsable and does not follow input-guidelines.
7. We will run plagiarism detection software. Anyone found guilty will be awarded a suitable penalty as per IIT rules.