CSCI 3901 Assignment 3

Due date: Friday, October 29, 2021 at 4:00pm Halifax time. Submissions through your CS gitlab repository in https://git.cs.dal.ca/courses/2021-fall/csci3901/assignment3/????.git where ???? is your CS id.

Problem 1

Goal

Work with graphs.

Background

Travel in the times of the covid-19 pandemic can be tricky. Different jurisdictions have different requirements on

- whether or not you can fly directly to them from other jurisdictions,and
- what kinds of tests you must have before entering.

Add the complexity of different travel opportunities by air vs by train and you start to need an advanced degree just to make travel arrangements.

Travellers can also have different priorities when making travel plans. Some want to minimize overall cost. Others want to travel with the least time in shared planes and trains. Still others want to have the fewest stops in their travels.

You will develop a Java class that will help travellers optimize their travel plans to get from one city to another.

Problem

Implement a class called TravelAssistant that will accept information that describes the travel conditions for flights and train rides. The TravelAssistant then finds the best travel plan for individuals basd on their start and destination cities, their vaccination status, and a description of what they prioritize in their travel plans.

Your TravelAssistant class will include the following methods, at a minimum:

- boolean addCity(String cityName, boolean testRequired, int timeToTest, int nightlyHotelCost) throws IllegalArgumentException
- boolean addFlight(String startCity, String destinationCity, int flightTime, int flightCost)) throws IllegalArgumentException
- boolean addTrain(String startCity, String destinationCity, int trainTime, int trainCost)) throws IllegalArgumentException

- List<String> planTrip (String startCity, String destinationCity, boolean isVaccinated, int costImportance, int travelTimeImportance, int travelHopImportance) throws IllegalArgumentException

Each of the methods operates as follows:

addCity

Indicate that a particular city is a possible starting point or destination in the travel plans. The parameters provide information about the city:

- cityName – the name of the city, which should be kept unique.
- testRequired – true if someone who is unvaccinated requires a negative covid test to travel into the city. Vaccinated individuals can enter the city whether or not testRequired is true or false.
- timeToTest – the number of days that it takes to get a covid test result in the city. Someone who isn't starting in this city would spend this many days in a hotel while waiting for test results. If timeToTest is negative then you cannot get a covid test in the city.
- nightlyHotelCosts – the cost of spending one night in a hotel in this city. All hotel costs will be normalized to the same currency before being reported to the TravelAssistant.

The method throws an exception if the input parameters are unacceptable. The method returns true if the city can now be used in the TravelAssistant, and returns false if the information contradicts data already known by the TravelAssistant.

addFlight

Record the existence of a one-way flight from the startCity to the destinationCity. Parameter flightTime is the in-air time of the flight (in minutes) and parameter flightCost is the cost of that flight.

The method throws an exception if the input parameters are unacceptable. The method returns true if the flight can be used by the TravelAssistant, and returns false if the information contradicts data already known by the TravelAssistant.

addTrain

Record the existence of a one-way train ride from the startCity to the destinationCity. Parameter trainTime is the on-train time of the trip (in minutes) and parameter trainCost is the cost of that train trip.

The method throws an exception if the input parameters are unacceptable. The method returns true if the train ride can be used by the TravelAssistant, and returns false if the information contradicts data already known by the TravelAssistant.

planTrip

Determine the sequence of plane, train, and city stays needed to travel from the startCity to the destinationCity. You can determine your own way to find the best path, but I recommend starting with looking at Dijkstra's algorithm, which computes the shortest path in a graph.

The best path is defined by what the traveller feels is most important. Those importances are defined by the relative weights of each of cost (both travel cost and hotel cost), travel time, and number of hops in the travel (taking one flight or one train ride is one "hop"; staying in a city is not a hop). Those relative weights are non-negative integers:

- costImportance
- travelTimeImportance
- travelHopImportance

For example, if cost is the only factor then costImportance can be 1 with travelTimeImportance and travelHopImportance both being 0. On the other hand, if the cost of a flight is equally important to each minute spent travelling then costImportance can be 1, travelTimeImportance can be 1 and travelHopImportance can be 0.

Boolean variable isVaccinated lets the TravelAssistant know whether or not the traveller is vaccinated; an unvaccinated individual may need to plan to get a covid test somewhere during the trip to enter one of the cities.

The method returns the list of cities visited on the trip along with the mode of travel. Each string in the returned list consists of the mode of travel (start, fly, train), a space, and then the city at the end of the flight or train ride (or the starting point as the first entry).

If there is no way to travel between the given cities then return null as the list.

Assumptions

You may assume that

- All costs are in comparable currencies.
- All travel times are in minutes.
- When testing your code, the "best" travel plan will be unique, so you don't need to worry about which travel plan to return in the case of a tied cost.

Constraints

- You may use any data structures from the Java Collection Framework.
- You may not use a library package to for your graph or for the algorithm on your graph.
- If in doubt for testing, I will be running your program on timberlea.cs.dal.ca. Correct operation of your program shouldn't rely on any packages that aren't available on that system.

Notes

- You will need to write your own "main()" driver to test your class, or use JUnit to test your class.
- You will need a class to store each vertex of the graph and a class to store each travel "hop".
- You will need to choose whether to store the graph as an adjacency matrix, as an adjacency list, or as an incidence matrix. Your best bet will likely be an adjacency list.
- You might consider writing your own method to print the graph. It might help simplify your debugging.
- It may be that you cannot plan travel between every pair of cities.
- When providing relative importance of cost, travel time, and hops, notice that the underlying units (dollars, minutes, and hops) may not be directly comparable. Valuing each factor equally may not look like the same integer for each of the factors. Instead, you might think of the importance as the dollar value to me of each minute of travel time and the dollar value to me of making one fewer hop. That interpretation shouldn't change your code, but it may help you determine test case values.

Marking scheme

- Documentation (internal and external), program organization, clarity, modularity, style – 4 marks
- Thorough (and non-overlapping) list of test cases for the problem – 3 marks
- Efficiency of your data structures and algorithms for the given task, as explained by your own description of that efficiency (part of external documentation) – 2 marks
- Ability to create the travel network from the inputs and to return appropriately in error conditions – 4 marks
- Ability to report travel paths correctly – 12 marks