Daniyar Nazarbayev

H00204990

Data Structures and Algorithms

Coursework 2

Section 1: Testing.

1.1 Program 1, Part A.

Part A asks to just create a directed graph and print out all the connections (edges). I did not face any problems doing that. I used 2 foreach loops: one with a vertex set and another with degrees of that vertex (amount of edges).

1.2 Program 1, Part B.

Part B asks to get the shortest path with Dijkstra's algorithm. I used a scanner to take input from the user. Once it gets the start and destination, it sends these values to another method that inputs them into the Dijkstra class. Before that, they are checked to see if such vertexes exist.

Once I get the path, I check whether PathEdgeList is equal to null – to see path exists at all. Then I use a for each loop to go through each edge. I use methods getSource and getTarget to get the vertexes of the edge. I use String.format before printing.

As for getting the total price, I use a variable total, to which I add the weights of each edge.

If PathEdgeList is null or such vertexes do not exist, it just calls the method again and makes user enter new input. Typing exit after successfully printing the itinerary, will close the scanner.

1.3 Program 2, Part C.

This one was tricky. At first I just wanted to input the flight information as the weight of the edge. Only later did I found out that it is impossible, since Weights are used by some algorithms to traverse the graph, like the Dijkstra's algorithm that is supposed to find the path with the least accumulated weight.

Then another idea came to my head - I wanted to create a class named Airport and then create a graph <Airport, DefaultWeightedEdge>. The airport would have a queue field in it, and each item in that queue will have fields containing arrival, departure, to what city it is heading, flight id.

But all of this felt wrong, since each airport will store all flights of all edges. Then I realized that I could replicate the same concept with an edge as well.

I found out how to extend the edge and created a FlightEdge which extends DefaultWeightedEdge. There I created a Queue that stores all the flights. The idea is to enqueue and dequeue flights. Due to the nature of a queue (first in, first out), it fit perfectly. That said, the queue has to be ordered, with elements at the front of the queue being upcoming flights. I wanted to implement my own queue, but I guess I got a bit lazy. The java's implementation should work perfectly, but there is a chance of human error breaking it.

For this I created a class called Flight, which has a constructor taking 3 input vars – plane id, departure, arrival. The time is written in "DD-MM-YY-HHMM" format, all in a String. I break it down into an array with split(-) and further get the hours and minutes with substring. I assign them to their specific global variables in the constructor.

1.4 Program 2, Part D.

Part D asks to print out that info. Nothing extraordinary. I used the shortestPath method from the program 1 and added a few modifications to it. Nothing is really different from Part B, except that in my foreach loop of Edges, I extract the edge's queue - specifically its first element. Before doing that I call a method removeObsolete(global_time) that removes any elements of the queue that are past the date (global_time).

1.5 Program 2, Part E.

Part E asks to count amount of hours spend in air. For this I have created a separate class called Time. At first it was just a method, but then the code for it kept growing and growing, to the point where I had to create a separate class for it.

Why did not I use any of the default time classes? Well, I guess because I did not want to erase the code that I worked so hard on.

I will not go into the details of this class, but I do want to mention that this class is a bit inaccurate when given a time period of multiple months. It is because some months have 30 days, some 31 and sometimes 28/29, so it may run short on a day or two.

In short the time method consists of 2 methods – difference and add. Difference to find the duration each flight. Add to add each flight's duration.

1.6 Program 3, Part F.

There are four questions in Part F: two involve working on how time works in your program and other two on traversal of the graph. Personally, I finished 1st, 2nd and 4th.

I decided not to make another package and instead continued building on it in program 2 package.

The first question asks to create time zones and local time for each airport. Since I have a custom time class, it was a bit tricky to me. That said I did find a way. I created a class named City and made my graph store <City, FlightEdge>. The City has a field that represents how much you need to add or subtract to convert to that time zone. Here, we must assume that everything is written in Greenwich time. None of the flight info is changed in the queue, it is just that before printing the time, I add the Target Vertexes' (City) field to the time and output that.

The second question asks to upgrade your time to span multiple days. When I was doing part E I already took care of that. In fact, it can span multiple months as well.

The third question asks to find a path where the difference between each flight is 1 to 5 hours. I have not finished it, but I have a few assumptions on how to do it. First I have to find all possible paths from the starting point to the end point. We have been doing Prolog in programming languages these past weeks, and the solution to this question seems very similar to it. The idea is to move forward from one vertex to another if the condition is true (1-5 hours), or backtrack if condition is false. When you move back, you take the next item from the queue that follows the condition, else go back again.

The forth question asks to find the path with the fewest changeovers. This you can do by simply making Dijkstra's algorithm take in a weightless graph. There is a class called AsDirectedUnweightedGraph. This is exactly what I used.

Section 2: Evaluation.

Well, I said all that had to be said. There were a few things I wanted to do when first starting.

At first I wanted to make a separate thread, and have that thread increment time (1 sec = 1 min) - a global field, and have a while true method that would update all the queues of all airports simultaneously. We studied threads a little in our Web Programming course, but I am afraid not enough. Later on I created an extra method for my FlightEdge that would take a parameter (global time), and would update the queue.

I would also like to note that my code looks exceptionally messy in some areas. The biggest offender if the time class. In that method each field needs one another, so it is very hard to separate it, I think.

String.format, while useful also does make code much messier.