Information of Group Members Group Member 1: Name: Begüm Şara Surname: Ünal Group Member 2: Name: Giray Berk Surname: Kuşhan Section:04 AssignmentNumber: 3

PROGRAMMING HOMEWORK 3 REPORT

CMPE 343/224-04

Table of Contents

[Problem Statement and Code Design 2](#_Toc135863027)

[Question 1 2](#_Toc135863028)

[Question 2 2](#_Toc135863029)

[Implementation and Functionality 2](#_Toc135863030)

[Question 1 2](#_Toc135863031)

[Node Class 2](#_Toc135863032)

[Q1 Class 2](#_Toc135863033)

[Question 2 3](#_Toc135863034)

[Q2 Class 3](#_Toc135863035)

[Testing 4](#_Toc135863036)

[Final Assessments 4](#_Toc135863037)

# Problem Statement and Code Design

There are two questions in this assignment.

## Question 1

The first question is calculating the minimum cost of reaching a destination using a transport network. The code we have written to achieve this purpose includes the Node and Edge classes. Also, this code contains the methods of Dijkstra's algorithm that we use to start the graph, add edges, and calculate the minimum cost.

## Question 2

The purpose of the second question is to calculate the minimum time required to reach each station in a transportation system with more than one bus and station. To achieve this goal, the code we wrote implements Breath First Search. The purpose of our Breath First Search application is to calculate the minimum time.

# Implementation and Functionality

This section describes the methods and classes we use to solve questions.

## Question 1

We used two classes, Q1, and Node class, to solve the first question.

Node Class

A ***Node class*** is defined by index and cost attributes. A comparable interface is used to make comparisons within the class.

Q1 Class

It also includes the ***Edge class*** within the ***Q1*** ***class***. The ***Q1 class*** also includes ***graph*** methods. That is, it creates the graph and adds the edges. It also initializes the variables and uses the ***Dijkstra algorithm*** to find the minimum cost. The class operations generally read the input, call the necessary methods, and give the total cost.

***Edge Class:*** The ***Edge class*** hides the target and the weight. The ***Edge class*** represents an edge in the graph.

***initGraph():*** This method aims to set up the graph by initializing an array of “ArrayLists,” where each “ArrayList” will represent the adjacency list for a particular vertex in the graph.

***addEdge():*** Adds an undirected edge between two nodes in a graph.

***dijkstra():*** This method implements Dijkstra's algorithm to find the shortest paths from a given source node in a graph. Once the priority queue is empty, all the shortest distances from the source node have been found. Return the array `dist` containing the shortest distances.

***Main:*** Takes values that represent the number of nodes, the number of edges, and the fixed cost, respectively. Creates an array `capacity` of size `n` to store the capacity of each node. After that, calls the `initGraph()` method to initialize the graph.

Uses a loop to read `m` sets of integers representing each edge's starting node, ending node, and weight. Calls the `addEdge()` method to add each edge to the graph.Reads an integer `k` representing the number of iterations.Creates two arrays, `totalCost` and `currCapacity`, both of size `k.` Copies the `capacity` array into `currCapacity` using `Arrays.copyOf() `.

Inside the loop, calls the `dijkstra()` method to find the shortest paths from the source node and store the result in the `dist` array. Checks if the node `currCapacity[j] 's current capacity is greater than 0. If the condition is true, the code calculates the cost of using node `j` by adding the shortest distance `dist[j]` and the fixed cost `f.` If the calculated cost exceeds the current minimum cost, the code updates `minCost` with the new cost and `minSlot` with the node’s index. After the inner loop, checks if `minSlot` is still -1. There are no available slots if it is, so sets `totalCost[i]` to -1. Otherwise, stores the minimum cost in `totalCost[i]` and decrements the selected slot’s capacity by 1.

## Question 2

In the ***Q1 class***, rather than dividing it into classes like ***Graph, Edge, and BFS***, we did it all in one class.

Q2 Class

The ***Q2 class*** uses the ***Edge class*** to represent the edges in the transport network. This class also implements ***Breath First Search*** to calculate the minimum time to reach each station. The class reads the input, builds the adjacency list, calls BFS, and prints the results.

***Edge Class:*** The Edge class represents an edge in the graph. The edge connects two nodes and contains information such as the target node, an ID for a bus associated with the edge, and the departure and arrival times.

***breathForSearch():*** A method that takes as input the starting station, a graphical representation, an array to store distances, and a list of bus times.

***Main:*** The user gives the input code to calculate the number of stations (N) and buses (M). The transportation network is then represented as an adjacency list. The code reads each bus's information and its route. It creates the graph by including edges to the adjacency list based on the bus routes.

The minimal time needed to travel between each station and the starting station is stored in an array called “dist,” which the code initializes with a size of N + 1 after creating the graph. The code then invokes the "breathForSearch " function, using the arguments starting station, graph, “dist,” and list of bus routes. If the station cannot be reached, then the code prints "-1". Otherwise, each station's minimum time is printed.

# Testing

We tested our code using various scenarios to test the accuracy of the code we wrote to solve the first and second questions. We created various graphs to test the code we wrote to calculate the minimum cost, so we checked whether it did the calculation correctly. We also printed the graph to test whether our code rendered the graph correctly based on the inputs entered.

# Final Assessments

* What were the trouble points in completing this assignment?
* The problem points in completing this assignment might be applying Dijkstra's algorithm for the first question and Breath First Search for the second question.
* Which parts were the most challenging for you?
* The most challenging part of this assignment was using Dijkstra's algorithm to calculate the minimum time.
* What did you like about the assignment? What did you learn from it?
* What we liked about this assignment was that it provided the opportunity to reinforce the subjects we learned and gave an idea about how the “EGO Cepte” application, which we use frequently in our daily life, works. While doing this assignment, we learned how Dijkstra's algorithm can be applied to solve optimization problems in transport networks.