CMPE 343
Programming Homework 3

Halil Mert Güler

Section 2

Assignment 3

Question 1

1. Problem Statement and Code Design

The task is to manage the parking facility system and create a program that finds the smallest paths for every car with their total cost. For this assignment directed edge-weighted graph data structure is used. Also computing the total cost according to the capacity parking slots and parking fee.

Sub tasks

- Understanding the task: According to the problem, we should create a java code using directed edge weighted graph structure that takes input from user and finds the smallest path. While doing this, we also compute edge weights and general parking fee to find the total cost of each car.

- Choose the best algorithm for the task: We are using a directed graph with weighted edges, and we need to hold distances. According to these requirements, the proper algorithm is Dijkstra's shortest path algorithm.

- Create the graph data structure: When creating the graph class, it is important to implement it as an edge weighted directed graph because. So, when adding the edge between two vertices, we also need to hold a weight variable.

- Generating a code part to take input accordingly: The code part that takes input from user should be in String form. After the input line is taken, this string input must split to get digits. Finally, digits will be used in code accordingly.

- Implementation: After we create our road map, we should start the implementation with creating "Main.Java" class that includes runnable function, the digraph class and necessary algorithms.

- Testing: At this part we will test our program with the according input and check the result.

2. Implementation and Functionality

For the implementation part, “Main.java” class with runnable function created at first. After that, the edge-weighted directed graph class was implemented.

```
class Digraph {
    int numNodes;
    List<Edge>[] adjacencyList;

    public Digraph(int numNodes) {
        this.numNodes = numNodes;
        this.adjacencyList = new ArrayList[numNodes + 1];
        for (int i = 1; i <= numNodes; i++) {
            adjacencyList[i] = new ArrayList<>();
        }
    }

    public void addEdge(int from, int to, int weight) {
        adjacencyList[from].add(new Edge(from, to, weight));
    }
```

The graph class has an integer value numNodes that corresponds to number of Nodes, adjacency list to hold adjacency values in form of Edge linked list array, constructor and method for adding edges between vertices. This “addEdge” method is adding edges with weights.

After that, the code part that takes input and initializes the graph is created.

```
        Scanner scanner = new Scanner(System.in);
        int numNodes = scanner.nextInt();
        int numEdges = scanner.nextInt();
        int parkingFee = scanner.nextInt();

        Digraph graph = new Digraph(numNodes);
        int[] capacities = new int[numNodes + 1];

        for (int i = 1; i <= numNodes; i++) {
            capacities[i] = scanner.nextInt();
        }
        for (int i = 0; i < numEdges; i++) {
            int from = scanner.nextInt();
            int to = scanner.nextInt();
            int weight = scanner.nextInt();
            graph.addEdge(from, to, weight);
        }

        int numVehicles = scanner.nextInt();
        scanner.close();
```

After input management is done, a modified version of Dijkstra's shortest path algorithm to find distances of every parking slot from the start vertex.

```
public int[] shortestPaths(int source) {
    int[] distances = new int[numNodes + 1];
    Arrays.fill(distances, Integer.MAX_VALUE);
    distances[source] = 0;

    PriorityQueue<Node> pq = new PriorityQueue<>();
    pq.offer(new Node(source, 0));

    while (!pq.isEmpty()) {
        Node curr = pq.poll();
        int nodeId = curr.id;
        int distance = curr.distance;

        if (distance > distances[nodeId]) {
            continue;
        }

        for (Edge edge : adjacencyList[nodeId]) {
            int newDistance = distance + edge.weight;
            if (newDistance < distances[edge.to]) {
                distances[edge.to] = newDistance;
                pq.offer(new Node(edge.to, newDistance));
            }
        }
    }

    return distances;
}
```

This function applies Dijkstra's algorithm to find the shortest path distances from a given source node to all other nodes in the graph. It uses a priority queue to efficiently process nodes with the smallest distances first and updates the distances whenever a shorter path is found. The function returns an array containing the shortest path distances from the source node to all other nodes. After that we use the returned array when we calculate total costs.

```
int[] totalCosts = new int[numVehicles];

int[] distances = graph.shortestPaths(1);
```

```
for (int i = 0; i < numVehicles; i++) {
    int cost = parkingFee;

    int minDistance = Integer.MAX_VALUE;
    int chosenSlot = -1;

    for (int j = 1; j <= numNodes; j++) {
        if (capacities[j] > 0 && distances[j] < minDistance) {
            minDistance = distances[j];
            chosenSlot = j;
        }
    }

    if (chosenSlot != -1) {
        capacities[chosenSlot]--;
        cost += minDistance;
    } else {
        cost = -1;
    }

    totalCosts[i] = cost;
}

for (int cost : totalCosts) {
    System.out.print(cost + " ");
}
```

This code part finds the nearest available parking slot for each vehicle based on the shortest path distances and available capacities using the returned array from "shortestPaths" function. It calculates the total cost by summing the parking fee with the distance traveled to the chosen parking slot. If no parking slot is available, it sets the cost to -1. Finally, it prints the total costs for each vehicle.

3. Final Assessments

- <u>What were the trouble points in completing this assignment?</u>
  The trouble points were to find total costs while iterating over parking slots and math the distances with true vertex.

- <u>Which parts were the most challenging for you?</u>
  Same with above.

- <u>What did you like about the assignment? What did you learn from it?</u>
  The part that I liked at this assignment was adapting algorithm programming to a real-life problem and finding solutions. Because these kinds of algorithms are used in real life while handling similar situations.