THE OPEN UNIVERSITY OF SRI LANKA
FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING BACHELOR OF SOFTWARE ENGINEERING PROGRAMME -

LEVEL 4 EEX4465 - DATA STRUCTURES AND ALGORITHMS

Miniproject_Number_4

621436296

S92076296

Group D

Ayon costa

Introduction

Develop intelligent traffic signal control system based

- Reacts to real-time traffic flow.
- Detects emergency vehicles.
- Adjusts green-light durations dynamically to reduce congestion.

Implements these data structure

- Graphs
- Heaps
- Lists
- Queues

1. Traffic Queue Management- Traffic Queue Management is a system used by to control and organize vehicles waiting in line at joint, turns , toll booths(Highway) , or traffic signals. It aims to reduce congestion, waiting time, and improve the flow of traffic.
   EX.- Vehicles are arriving from all four directions, and small road and highways.
   If there is no proper management, all vehicles try to move at the same time — leading to Craziness , traffic jams, or accidents.
   Solution -At a 4-way junction:
   Traditional way-

   North road has 20 vehicles waiting.

   South has 5.

   East and West have 10 each.

   And you can use intelligent way.

   Like monitor and decision

2. Emergency Vehicle Detection and Priority - Emergency Vehicle Identification and Priority Scheduling is a smart traffic control strategy that helps ambulances, fire trucks, VIP squad and police vehicles get through traffic faster, especially during duty of responsibility.
   Ex. Imagine an ambulance is complicated to a hospital with a Danger patient between die or live.
   It's stuck at a red light while the other road is green and busy.
   If it waits, the patient could be in die in transport problems. That effect to Government. So
   This is where emergency vehicle detection and priority comes in.

3. **Traffic Network Modeling- used to** is the process of **creating a intelligent model of a city or area road network** to understand, analyze, get experience and improve traffic flow in real life.

Ex.- Imagine you live and drive through Colombo. It is very hard to everyday.

Traffic Network Modeling simulates how all these roads and vehicles interact.

About the this system they encourage to get past data for how model behavior each area, each road, each traffic signal.

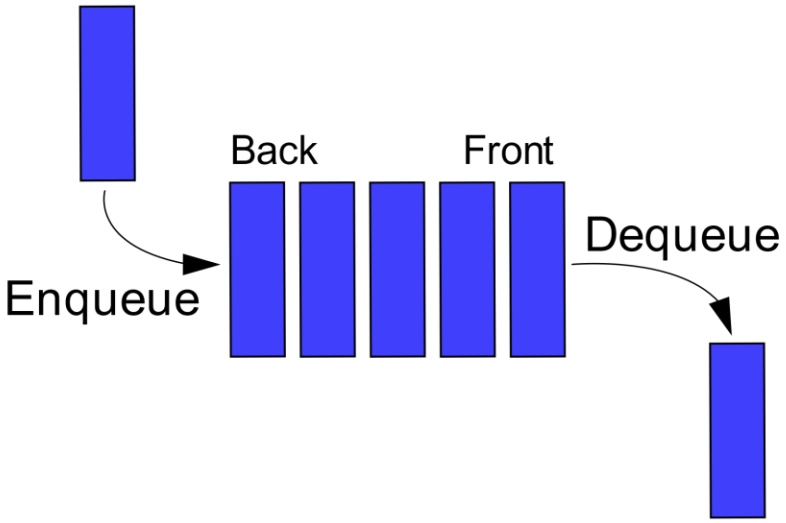
Data structures

Traffic Queue management used Queue (FIFO- First in, First out).

At every intersection, cars wait in line. You need to manage them by direction and vehicle type. So that reason we used FIFO method.

Ex. You can used special character to identify emergency vehicle. Even through this.

(This means the element that is added first will be the first to be removed. It works just like a line at a bank or a queue at a bus stop: the first person to join the line is the first to be served.)



1. Think of every traffic intersection (junction), or traffic or T joint as a station that has 4 directions. Each direction will have its own line of vehicles.
2. Whenever a vehicle arrives at an intersection or traffic or T joint, you add it to the queue of its own or came direction. (Using ADD.APPEND)
3. Queue size (more vehicles = higher priority) 1 vehicle get first. And extra you can define emergency vehicles.
4. When a direction gets the green light, remove vehicles from that direction’s queue one by one (FIFO). (1,2,3,4,- remove/ 11,12,13- Hold)

For Emergency Vehicle Priority A Heap is a special binary tree -

based data structure used to quickly access the highest (or lowest) priority identification.

2 types of HEAP Maxheap, Minheap. But we used Maxheap

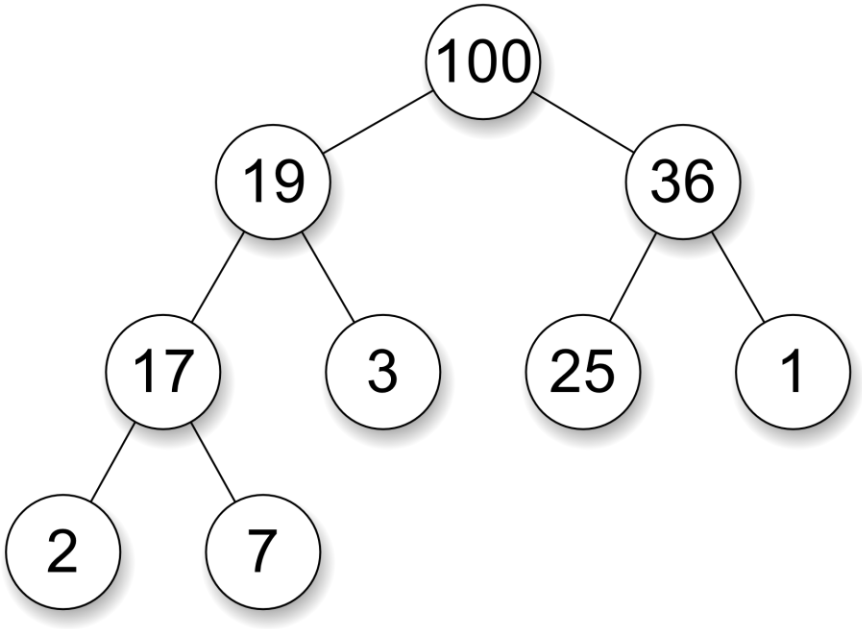
EX. When ambulance arrive from north to traffic then stop every vehicle, every area, and let it pass.

But when comes to Police cars same as the ambulance.

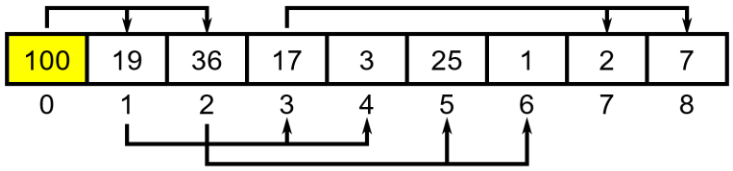
We can add some priority modify after like Bus arrived It will be high priority, Because of the improve public transport for people.

1. We need assign vehicle ID like
   Ambulance - 5
   Police- 4
   Other- 1
2. Use Insert and max method.

Tree representation



Array representation



For traffic network model data structure graph used Node, edge.

Edge - can have time, level, distance

Why used graph- simulate and visual traffic. Connect with road. Vehicle movement tracking. Find shortest path.

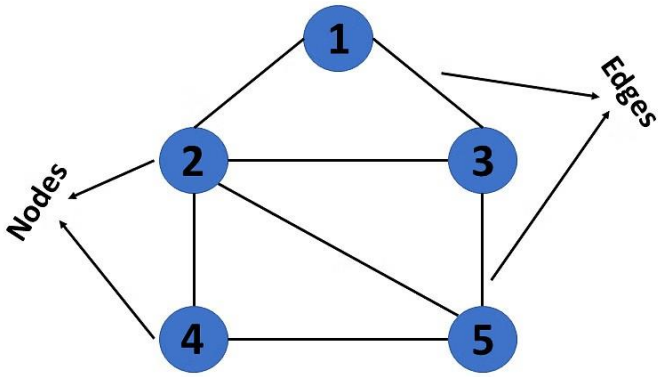
Node- Traffic

Edge- free road

Weight- time

List- store data

Pathfinder- shortest time distance.



Pseudocode each components

Traffic queue management

```
import java.util.*; // import all utility classes like map, hashmap, list,

public class First { // declare first class
```

```
    static Map<String, Queue<String>> queues = new HashMap<>();
//A map to store vehicle queues for each direction (north, south,
east, west).
    static Map<String, List<Integer>> waitTimes = new HashMap<>();
//map store list of wait times
    static Scanner scanner = new Scanner(System.in); // scan the
user input


    public static void main(String[] args) { // main method
        // Initialize directions
        String[] directions = {"north", "south", "east", "west"};
// array for correcr direction
        for (String dir : directions) {
            queues.put(dir, new LinkedList<>());
            waitTimes.put(dir, new ArrayList<>());
        } // input queue and time


        // Main loop
        while (true) { // keep run option 3 choose
            System.out.println("\n------- Traffic Signal Control
===");
            System.out.println("1. Add car to road");
            System.out.println("2. Serve green light request");
            System.out.println("3. Exit due to Emergency
vehicle");
```

```
            System.out.print("Choose an option: ");
            String choice = scanner.nextLine();  // get user input

            switch (choice) { // Handle user choice
                case "1":
                    addCar();
                    break; // id user add car function
                case "2":
                    String direction = getGreenLightDirection();
                    if (direction != null) {
                        serveVehicles(direction);
                    } else {
                        System.out.println("No cars in any
direction.");
                    }
                    break; // call the function
getGreenLightDirection();  to If a direction has vehicle it calls
serveVehicles() to let cars pass
                case "3":
                    System.out.println("Exiting traffic controller
due to emergency vehicle. Bye!");
                    return; // exit program
                default:
                    System.out.println("wroong option. Please
choose 1, 2, or 3.");
```

```
            } // invalid input
        }
    }

    static void addCar() {
        System.out.print("Enter direction (north/south/east/west): ");
        String direction = scanner.nextLine().trim().toLowerCase(); // convert format output and het direction prompt
        if (!queues.containsKey(direction)) {
            System.out.println("wrong  direction. Try again."); // valid direction input
            return;
        }

        System.out.print("Enter car type (regular/public/emergency): ");
        String carType = scanner.nextLine().trim().toLowerCase(); // get car type and convert lower
        if (!Arrays.asList("regular", "public", "emergency").contains(carType)) {
            System.out.println("Invalid car type.");
            return; // find valid car type
        }
```

```
        System.out.print("Enter wait time in seconds (10-100): ");//  ask time
        try {
            int waitTime = Integer.parseInt(scanner.nextLine());
            queues.get(direction).add(carType);
            waitTimes.get(direction).add(waitTime); // wait time adds with car type
            System.out.println(carType.substring(0, 1).toUpperCase() + carType.substring(1) +
                    " added to " + direction.substring(0, 1).toUpperCase() + direction.substring(1) + " queue."); // confirm message
        } catch (NumberFormatException e) {
            System.out.println("Invalid wait time. Please enter a number.");
        } // find wrong input time
    }

    static String getGreenLightDirection() {
        double maxScore = -1;
        String selectedDirection = null; // get green light function

        for (String direction : queues.keySet()) {
```

```
            int count = queues.get(direction).size();
            double avgWait = 0; // get number of cars
            if (count > 0) {
                int totalWait = 0;
                for (int t : waitTimes.get(direction)) {
                    totalWait += t;
                }
                avgWait = (double) totalWait / count;
            } // calculate averarge wait time
            double score = count + avgWait; //add score count time
            System.out.printf("%s - Vehicles: %d, Avg Wait: %.2f,
Score: %.2f%n",
                    capitalize(direction), count, avgWait, score);
// shoe data with each direction

            if (score > maxScore) {
                maxScore = score;
                selectedDirection = direction;
            }
        }
        return selectedDirection; // choose direction with high
speed
    }
```

```
    static void serveVehicles(String direction) {
        System.out.println("\nGreen light given to " +
direction.toUpperCase() + " direction:");
        int count = 0;// message for release vehicle
        Queue<String> carQueue = queues.get(direction);
        List<Integer> waitList = waitTimes.get(direction); //get
queue list and time

        while (!carQueue.isEmpty() && count < 3) {
            String car = carQueue.poll();
            waitList.remove(0);
            System.out.println("  Car (" + car + ") passed.");
            count++;
        }
    } // let 3 car release remove queue and list

    static String capitalize(String word) {
        return word.substring(0, 1).toUpperCase() +
word.substring(1);
    }
} // captal first letter
```

User interaction

After establishing the system. User get 3 questions Add car, green light, Exit emergency car. If user type 1 and asked direction, time, type of vehicle. And system add to the queue by direction. If user type 2 and release the vehicle after add number 1 option. When you add more vehicle it will be released by First to last you added. If user enter 3 then all emergency vehicles passed the traffic light. And after enter 3 loop is closed.

To primary goal is handle traffic with queue data structure.

Emergency Vehicle Detection and Priority Scheduling

```
import java.util.*;// import all utility classes like Map, HashMap, List, etc.


public class SecondR {


    // Vehicle variables class
    static class Vehicle {
        int priority;
        String type;
        String location;
        String time;
        String direction;
```

```
        Vehicle(int priority, String type, String location, String
time, String direction) {
            this.priority = priority;
            this.type = type;
            this.location = location;
            this.time = time;
            this.direction = direction;
        }// construct
    }

    // Tree node class
    static class TreeNode {//Define BST for treenode
        Vehicle vehicle;// each node store vehicle object
        TreeNode left, right; //Has two child references: left and
right

        TreeNode(Vehicle vehicle) {
            this.vehicle = vehicle;
        }// construct for tree node
    }

       static class VehicleBST {
        TreeNode root; // nested class manage BST treee node
calles root
```

```
        // Insert cars
        void insert(Vehicle v) {
            root = insertRec(root, v); //helper method for update
root
        }

        // Recursive method
        private TreeNode insertRec(TreeNode root, Vehicle v) {
            if (root == null) return new TreeNode(v); // if
current node full create new
            if (v.priority < root.vehicle.priority) //vehicle's
priority is less tha current node insert into left
                root.left = insertRec(root.left, v);
            else
                root.right = insertRec(root.right, v); //other
vise in to right
            return root;
        }

        // display vehicles high to low priority
        void serveVehiclesHighToLow(TreeNode node) {
            if (node == null) return;
            serveVehiclesHighToLow(node.right);
```

```
            Vehicle v = node.vehicle; //
            System.out.printf("GREEN for %s at %s (%s, %s,
Priority %d)%n", //after visit subtree print cuurebt node
                    v.type.toUpperCase(), v.location, v.time,
                    v.direction.toUpperCase(), v.priority);
//upgrade message with upperclass
            serveVehiclesHighToLow(node.left);
        }//recursively visit the left subtree
    }

    public static void main(String[] args) {
        // Priority map for each vehicle type
        Map<String, Integer> priorityMap = Map.of(
                "ambulance", 5,
                "firetruck", 4,
                "police", 3,
                "car", 1
        );// priority listt

        // Predefined vehicles data
        String[][] vehiclesData = {
                {"ambulance", "Junction A", "08:05AM", "north"},
                {"car", "Junction F", "08:05AM", "south"},
                {"firetruck", "Junction B", "08:06AM", "east"},
```

```
                {"car", "Junction G", "08:06AM", "east"},
                {"police", "Junction C", "08:07AM", "west"},
                {"car", "Junction H", "08:08AM", "north"}
        };

        VehicleBST tree = new VehicleBST(); // Create instance

        for (String[] v : vehiclesData) {//loop over vehicle
            String type = v[0], loc = v[1], t = v[2], dir = v[3];
//get type, location, time, direction.
            int prio = priorityMap.getOrDefault(type,
1);//get  priority map
            Vehicle vehicle = new Vehicle(prio, type, loc, t,
dir);
            tree.insert(vehicle); // Insert into BST

            // Print detection message
            System.out.printf(" Cars Detected: %s at %s, %s, going
%s (Priority %d)%n",
                    type.toUpperCase(), loc, t, dir.toUpperCase(),
prio);
        }

        // Serve vehicles in priority order
```

```
        System.out.println("\nGiving green light based on priority:");
        tree.serveVehiclesHighToLow(tree.root);
    }
}
```

User interaction

This system not have input only for emergency vehicle identify by and they released at first. Other vehicles move with first to last. First code detected all vehicles. Then find emergency vehicles.

Traffic module

```
import java.util.*;//import all utility classes like map, hashmap, list,

public class Third { // starts main class

    static Map<String, List<Edge>> graph = new HashMap<>(); // define static graph

    static class Edge {
        String neighbor; //destination node
        int weight; // time cost
```

```
        Edge(String neighbor, int weight) {
            this.neighbor = neighbor;
            this.weight = weight; // constructors for variables
        }
    }

    static class Node implements Comparable<Node> {
        String name; // A B C D
        int cost;
        List<String> path; // helper class for dijksta

        Node(String name, int cost, List<String> path) {
            this.name = name;
            this.cost = cost;
            this.path = new ArrayList<>(path);
            this.path.add(name);
        } // Constructor for node

        @Override
        public int compareTo(Node other) {
            return Integer.compare(this.cost, other.cost); // for
min-heap
        } //compare cost for priority queue
```

```
    }

    public static int dijkstra(String start, String end) { //use
Dijlksta metod
        PriorityQueue<Node> queue = new PriorityQueue<>();
        Set<String> visited = new HashSet<>();//  pick node lowest
total and avoid same value

        queue.add(new Node(start, 0, new ArrayList<>())); // cost
=0

        while (!queue.isEmpty()) {
            Node current = queue.poll(); //queue not empty take
node with lowest total

            if (visited.contains(current.name))
                continue; //already visited skip

            visited.add(current.name); //mark the node

            if (current.name.equals(end)) {
                System.out.println("Shortest path: " +
String.join(" → ", current.path));
                System.out.println("Answer  Total travel time is
:  " + current.cost + "  minutes");
```

```
                return current.cost;
            } // print shortest path and return cost

            List<Edge> neighbors =
graph.getOrDefault(current.name, new ArrayList<>()); // list all
nighbors of current node graph
            for (Edge edge : neighbors) {
                if (!visited.contains(edge.neighbor)) {
                    queue.add(new Node(edge.neighbor, current.cost
+ edge.weight, current.path));
                }
            }
        } // new cost = cuurent cost+ weight path= current path+
neighbour

        System.out.println("No path found.");
        return Integer.MAX_VALUE;
    } // if loop fail destination not found

    public static void main(String[] args) { //main method
        graph.put("A", List.of(new Edge("B", 4), new Edge("C",
2)));
        graph.put("B", List.of(new Edge("A", 4), new Edge("D",
5)));
```

```
        graph.put("C", List.of(new Edge("A", 2), new Edge("D",
1)));
        graph.put("D", List.of(new Edge("B", 5), new Edge("C",
1))); //Manually builds the graph with how weight


        Scanner scanner = new Scanner(System.in); //get user input


        System.out.print("Enter starting point A B C D: "); // get
first value convert uppercase
        String start = scanner.nextLine().trim().toUpperCase();


        System.out.print("Enter destination point A B C D: ");
        String end = scanner.nextLine().trim().toUpperCase();//
last destination


        dijkstra(start, end); // call method
    }
}
```

User interaction

User need to enter the First and last path. Then if user enter correct and accurate data then. Show the shortest path, total travel time with tuple list by minutes. This output get by this list

```
graph = {
    "A": [("B", 4), ("C", 2)],
    "B": [("A", 4), ("D", 5)],
    "C": [("A", 2), ("D", 1)],
    "D": [("B", 5), ("C", 1)]
}
```

ALL combine code

```
import java.util.*; // import all utility classes like Map, HashMap, List, etc.

public class all {

    static Scanner scanner = new Scanner(System.in); // scan user input

    static class VehicleInfo {
        String type;
```

```
        int waitTime;

        VehicleInfo(String type, int waitTime) {
            this.type = type;
            this.waitTime = waitTime;
        }
    }//hold car type and time

        static Map<String, LinkedList<VehicleInfo>>
queues = new HashMap<>();//map that stores direction

    // Directions
    static final String[] DIRECTIONS = {"north",
"south", "east", "west"};

    // add empty queues  direction
    static void initQueues() {
        for (String d : DIRECTIONS) {
            queues.put(d, new LinkedList<>());
        }
```

```
    }

        static void addCar() { //add car
            System.out.print("Enter direction
(north/south/east/west): ");
            String dir =
scanner.nextLine().trim().toLowerCase(); // ask the
user direction
            if (!queues.containsKey(dir)) {
                System.out.println("Invalid
direction.");
                return;
            }// if user enter wrong direction


            System.out.print("Enter car type
(regular/public/emergency): ");
            String type =
scanner.nextLine().trim().toLowerCase(); // aske
type
```

```
        if (!type.equals("regular") &&
!type.equals("public") && !type.equals("emergency"))
{
            System.out.println("Wrong car type.");
            return;
        } //if user input wrong type


        System.out.print("Enter wait time in
seconds: "); // ask time
        try {
            int wait =
Integer.parseInt(scanner.nextLine().trim());
            queues.get(dir).add(new
VehicleInfo(type, wait));//addnew car to queue
            System.out.printf("%s added to %s
queue.\n", capitalize(type), capitalize(dir));// to
message for
        } catch (NumberFormatException e) {
            System.out.println("add INTeger ");
        } // find that wrong
    }
```

```
    static String capitalize(String s) {
        return s.substring(0, 1).toUpperCase() +
s.substring(1);
    } // ex-emergency -- Emergency


     static double
avgWaitTime(LinkedList<VehicleInfo> queue) {
        if (queue.isEmpty()) return 0;
        int sum = 0;
        for (VehicleInfo v : queue) sum +=
v.waitTime;
        return (double) sum /
queue.size();//calculate time in queue
    }


    static double scoreQueue(LinkedList<VehicleInfo>
queue) {
        return queue.size() + avgWaitTime(queue);//
Score = num cars + avg time
    }
```

```
    static String getGreenLightDirection() {
        double maxScore = -1;
        String bestDir = null;
        for (String d : DIRECTIONS) {
            double score =
scoreQueue(queues.get(d));
            if (score > maxScore) {
                maxScore = score;
                bestDir = d;
            }
        }
        return bestDir;
    }// Finds the direction that should get green
light based on score.


       static void serveVehicles() { //start new
fuction
        int served = 0; //count for how many
vehicles
```

```
        for (String d : DIRECTIONS) {
            LinkedList<VehicleInfo> queue =
queues.get(d);
            Iterator<VehicleInfo> it =
queue.iterator();
            while (it.hasNext() && served < 3) {
                VehicleInfo v = it.next();
                if (v.type.equals("emergency")) {
                    System.out.printf("GREEN light
to %s: EMERGENCY vehicle passed.\n",
d.toUpperCase());
                    it.remove();
                    served++;
                }
            } // serves up to 3 emergency vehicles.
        }

        if (served > 0) return; // stop if any
emergency vehicle was served

        String dir = getGreenLightDirection();
```

```
        if (dir == null ||
queues.get(dir).isEmpty()) {
            System.out.println("No vehicles
waiting.");
            return;
        } // Find the best direction and check if it
has vehicles.


        System.out.printf("GREEN light to %s:\n",
dir.toUpperCase());
        LinkedList<VehicleInfo> queue =
queues.get(dir); //prepare to release


        served = 0;
        List<String> priorities = List.of("public",
"regular"); // no emergency in here
        for (String priority : priorities) {
            Iterator<VehicleInfo> it =
queue.iterator();
            while (it.hasNext() && served < 3) {
                VehicleInfo v = it.next();
```

```
                if (v.type.equals(priority)) {
                    System.out.printf("  %s vehicle
passed.\n",  v.type.toUpperCase());
                    it.remove();
                    served++;
                }
            }
            if (served >= 3) break;
        }//Removes up to 3 vehicles of list
    }

    static class Edge {
        String neighbor;
        int weight;

        Edge(String neighbor, int weight) {
            this.neighbor = neighbor;
            this.weight = weight;
        }
    } // store connection between nodes
```

```
    static class Node implements Comparable<Node> {
        String name;
        int cost;
        List<String> path;

        Node(String name, int cost, List<String>
path) {
            this.name = name;
            this.cost = cost;
            this.path = new ArrayList<>(path);
            this.path.add(name);
        }

        @Override
        public int compareTo(Node other) {
            return Integer.compare(this.cost,
other.cost);
        }
    }//Dijkstra algorithm
```

```
    static Map<String, List<Edge>> graph = new
HashMap<>(); //main graph


    static void buildGraph() {
        graph.put("A", List.of(new Edge("B", 4), new
Edge("C", 2)));
        graph.put("B", List.of(new Edge("A", 4), new
Edge("D", 5)));
        graph.put("C", List.of(new Edge("A", 2), new
Edge("D", 1)));
        graph.put("D", List.of(new Edge("B", 5), new
Edge("C", 1)));
    } //add roads between points and weights


    static void findShortestPath() {
        System.out.print("Enter starting point
(A/B/C/D): ");
        String start =
scanner.nextLine().trim().toUpperCase();
```

```
        System.out.print("Enter destination point (A/B/C/D): ");
        String end = scanner.nextLine().trim().toUpperCase(); // ask user entry

        if (!graph.containsKey(start) || !graph.containsKey(end)) {
            System.out.println("Invalid nodes.");
            return;
        } // check input

        PriorityQueue<Node> pq = new PriorityQueue<>();
        Set<String> visited = new HashSet<>();
        pq.add(new Node(start, 0, new ArrayList<>())); //queue and visited set.

        while (!pq.isEmpty()) {
            Node current = pq.poll();
```

```
            if (visited.contains(current.name))
continue;
            visited.add(current.name); // loop find
shortest path


            if (current.name.equals(end)) {
                System.out.println("Shortest path: "
+ String.join(" -> ", current.path));
                System.out.println("Total travel
time: " + current.cost + " minutes");
                return; // end found show to user
            }


            for (Edge e :
graph.getOrDefault(current.name,
Collections.emptyList())) {
                if (!visited.contains(e.neighbor)) {
                    pq.add(new Node(e.neighbor,
current.cost + e.weight, current.path));
                } // add neighboring road
            }
```

```
        }

        System.out.println("No path found.");
    } // if path not find

    public static void main(String[] args) { //main
        initQueues();
        buildGraph();

        while (true) {
            System.out.println("\n--------======---
==== TRAFFIC CONTROL SYSTEM ");
            System.out.println("1. Add car to
queue");
            System.out.println("2. Serve vehicles
(emergency first)");
            System.out.println("3. Find shortest
route (Dijkstra)");
            System.out.println("4. Exit");
            System.out.print("Choose an option: ");
// ask questions
```

```
            String choice =
scanner.nextLine().trim(); //get input from user


            switch (choice) {
                case "1":
                    addCar(); // go addcar function
                    break;
                case "2":
                    serveVehicles(); //go
serveVehicles function
                    break;
                case "3":
                    findShortestPath(); // go
findShortestPath function
                    break;
                case "4":
                    System.out.println("Exiting.
Stay safe!");
                    return;
```

```
                default:
                    System.out.println("Invalid
option."); //wrong input
                }
            }
        }
    }
```

END