Department: **Software Engineering:**

**Name:** **Kashif khan (68409)** Program: **BS(SE)**

**Data Structures and Algorithms**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Announced date:10/06/2025** | | | **Due Date: 30/06/2025** | | **Total Marks = 20** |
| **Complex Computing Problem (CCP)** | | | | | |
| **Mapped CLO** | **SDG** | | **Complex Problem Solving Mapped** | | |
| CLO3 | 4 | | WP1 (Depth of knowledge required)  WP2 (Range of conflicting requirements required)  WP3 (Depth of analysis required)  WP4 (Familiarity of Issues) | | |

# Title: Efficient Traffic Management System Simulation Using Data Structures

# Problem Statement:

The objective of this problem is to design and implement a **Traffic Management System Simulator** that models the flow of vehicles at multiple intersections in a city. The system will utilize key data structures such as linked lists, queues, trees, and graphs to simulate and optimize vehicle flow, minimize congestion, and improve traffic efficiency. The system will incorporate features like dynamic vehicle queuing, intersection control, and traffic flow analysis.

## Constraints/Assumptions

1. **Constraints:**
   * The simulation must handle up to **10,000 vehicles** and **50 intersections**.
   * Traffic light timings must be adjustable and optimized based on real-time congestion.
   * Vehicle priority (e.g., emergency vehicles) should be supported and simulated.
   * The system should run efficiently with a response time of less than **1 second** for updates.
   * Assume the roads and intersections are predefined in a grid-like structure.
2. **Assumptions:**
   * All intersections are connected through roads, forming a graph structure.
   * Vehicle arrivals follow a predefined or randomly generated pattern.
   * Traffic rules (e.g., no vehicle should be allowed to cross on red) are strictly followed.
   * No accidents or roadblocks are simulated in this version.

## Identification of Areas Where Use of Computational/Modern Tools Is Required

1. **Traffic Simulation:**
   * Implement vehicle queues using data structures (queues for lanes, priority queues for emergency vehicles).
   * Use a graph to model the road network and calculate shortest paths using algorithms like Dijkstra’s or A\*.
2. **Traffic Optimization:**
   * Employ real-time traffic light control based on queue lengths and congestion levels.
   * Implement tree-based data structures for efficient searching and updating of traffic data.
3. **Visualization and Monitoring:**
   * Integrate modern tools or libraries like JavaFX or Swing for a graphical user interface to visualize traffic flow.
   * Use external libraries for real-time graph plotting and statistical analysis.
4. **Performance Analysis:**
   * Utilize computational tools to benchmark the system under varying traffic loads and identify bottlenecks.

## Expected Outcomes

1. **Functional System:**
   * A working simulation that models real-world traffic flow at intersections using data structures in Java.
   * Dynamic adjustment of traffic lights based on queue lengths.
2. **Optimized Performance:**
   * Improved traffic throughput and reduced congestion through effective use of algorithms and data structures.
3. **Real-Time Visualization:**
   * A graphical representation of traffic flow and congestion levels.
4. **Scalability:**
   * The ability to handle increasing traffic load and a larger number of intersections efficiently.
5. **Educational Value:**
   * Enhanced understanding of data structures like queues, graphs, and priority queues and their practical applications.

## Sample Use Case

**Sample Use Case for the Traffic Management System Simulation**

**Use Case: Optimize Traffic Flow at an Intersection**

**Actors:**

1. **System Administrator:** Configures and monitors the traffic simulation.
2. **Traffic System:** The automated simulator that handles vehicle flow and manages traffic light timings.

**Preconditions:**

1. The city road network and intersections are predefined and represented using a **graph** data structure.
2. Vehicles are dynamically added to the system, forming **queues** at each road segment.
3. The system is initialized, and all data structures (queues, graphs, priority queues) are properly set up.

**Steps:**

**1. Initialization:**

* The system administrator initializes the simulation with:
  + **Graph Input:** A road network with intersections (nodes) and roads (edges), along with weights representing distances or average traffic density.
  + **Vehicle Input:** A simulated stream of vehicles arriving at random intervals, modeled as a queue.

**2. Vehicle Flow Simulation:**

* Each intersection has a **priority queue** to manage emergency vehicles and normal vehicles.
* Vehicles are dequeued from the lane's queue and processed:
  + Emergency vehicles are prioritized and cleared first.
  + Normal vehicles are processed based on the traffic light status (green/red).

**3. Traffic Light Control:**

* The system dynamically adjusts the traffic light timing based on:
  + Queue lengths at each lane (tracked using queues).
  + Priority of vehicles in the queue.
* The adjusted timing is calculated and applied.

**4. Congestion Monitoring:**

* If a lane’s queue length exceeds a threshold, the system recalculates the shortest path for incoming vehicles using **Dijkstra’s Algorithm** on the graph.

**5. Visualization:**

* The system generates a **real-time graphical simulation** of vehicle movement and congestion levels:
  + Intersections show the number of vehicles in queues.
  + Roads between intersections display vehicle density.

**6. Reporting:**

* At the end of the simulation, the system generates:
  + Average wait time per vehicle.
  + Total time for congestion clearance.
  + Traffic flow efficiency metrics.

**Postconditions:**

1. Traffic flows smoothly with minimized congestion at intersections.
2. Emergency vehicles have prioritized clearance.
3. Metrics and logs of the simulation are available for further analysis.

**Example Data Input/Output:**

**Input:**

* Road Network (Graph):

Nodes: {A, B, C, D}

Edges: {A->B (2 mins), B->C (3 mins), A->D (5 mins), D->C (4 mins)}

* Vehicles (Queue at A): V1 (normal), V2 (emergency), V3 (normal), V4 (normal)

**Output:**

* Vehicle Clearance Order at Intersection A: V2 -> V1 -> V3 -> V4
* Traffic light timings at A:
  + Green for A->B: **45 seconds**
  + Red for A->D: **15 seconds**
* Visualization: A live traffic flow simulation.

## Deliverable:

1. **Documentation:**
   * Project proposal detailing objectives, scope, and methodology.
   * Design document with UML diagrams (e.g., class diagrams, sequence diagrams).
2. **Codebase:**
   * A well-documented Java codebase implementing the traffic management system.
3. **Simulation Tool:**
   * An interactive GUI-based application to visualize and simulate traffic scenarios.
4. **Test Cases and Results:**
   * Comprehensive test cases covering various traffic scenarios and corresponding performance results.
5. **Presentation:**
   * A detailed presentation of the project, including challenges faced, solutions implemented, and lessons learned.
6. **User Guide:**
   * A concise guide to help users interact with and test the system.

**Source Code**

public class Vehicle implements Comparable<Vehicle> {

private String id;

private boolean isEmergency;

private long arrivalTime;

public Vehicle(String id, boolean isEmergency, long arrivalTime) {

this.id = id;

this.isEmergency = isEmergency;

this.arrivalTime = arrivalTime;

}

public boolean isEmergency() {

return isEmergency;

}

public String getId() {

return id;

}

@Override

public int compareTo(Vehicle other) {

if (this.isEmergency && !other.isEmergency) return -1;

if (!this.isEmergency && other.isEmergency) return 1;

return Long.compare(this.arrivalTime, other.arrivalTime);

}

}

import java.util.\*;

public class Intersection {

private String name;

private PriorityQueue<Vehicle> vehicleQueue;

public Intersection(String name) {

this.name = name;

vehicleQueue = new PriorityQueue<>();

}

public void addVehicle(Vehicle vehicle) {

vehicleQueue.offer(vehicle);

}

public void processVehicles(int count) {

System.out.println("Intersection " + name + " cleared:");

for (int i = 0; i < count && !vehicleQueue.isEmpty(); i++) {

Vehicle v = vehicleQueue.poll();

System.out.println("Vehicle " + v.getId() + " cleared. Emergency: " + v.isEmergency());

}

}

public int getQueueLength() {

return vehicleQueue.size();

}

}

import java.util.\*;

public class RoadNetwork {

private Map<String, Map<String, Integer>> adjList = new HashMap<>();

public void addRoad(String from, String to, int time) {

adjList.putIfAbsent(from, new HashMap<>());

adjList.get(from).put(to, time);

}

public List<String> shortestPath(String start, String end) {

Map<String, Integer> dist = new HashMap<>();

Map<String, String> prev = new HashMap<>();

PriorityQueue<String> pq = new PriorityQueue<>(Comparator.comparingInt(dist::get));

for (String node : adjList.keySet()) {

dist.put(node, Integer.MAX\_VALUE);

prev.put(node, null);

}

dist.put(start, 0);

pq.add(start);

while (!pq.isEmpty()) {

String current = pq.poll();

if (current.equals(end)) break;

for (Map.Entry<String, Integer> neighbor : adjList.get(current).entrySet()) {

int newDist = dist.get(current) + neighbor.getValue();

if (newDist < dist.get(neighbor.getKey())) {

dist.put(neighbor.getKey(), newDist);

prev.put(neighbor.getKey(), current);

pq.add(neighbor.getKey());

}

}

}

List<String> path = new ArrayList<>();

for (String at = end; at != null; at = prev.get(at)) {

path.add(at);

}

Collections.reverse(path);

return path;

}

}import java.util.\*;

public class TrafficSimulator {

private Map<String, Intersection> intersections = new HashMap<>();

private RoadNetwork network = new RoadNetwork();

public void addIntersection(String name) {

intersections.put(name, new Intersection(name));

}

public void addRoad(String from, String to, int time) {

network.addRoad(from, to, time);

}

public void addVehicleToIntersection(String intersection, Vehicle vehicle) {

if (intersections.containsKey(intersection)) {

intersections.get(intersection).addVehicle(vehicle);

}

}

public void simulateTraffic() {

for (Intersection i : intersections.values()) {

i.processVehicles(3); // clear 3 vehicles per cycle

}

}

public void rerouteCongestedPaths() {

for (Map.Entry<String, Intersection> entry : intersections.entrySet()) {

if (entry.getValue().getQueueLength() > 5) {

System.out.println("Congestion at " + entry.getKey() + ", finding alternative paths...");

List<String> path = network.shortestPath(entry.getKey(), "C");

System.out.println("Suggested Path: " + path);

}

}

}

}

public class Main {

public static void main(String[] args) {

TrafficSimulator simulator = new TrafficSimulator();

simulator.addIntersection("A");

simulator.addIntersection("B");

simulator.addIntersection("C");

simulator.addIntersection("D");

simulator.addRoad("A", "B", 2);

simulator.addRoad("B", "C", 3);

simulator.addRoad("A", "D", 5);

simulator.addRoad("D", "C", 4);

simulator.addVehicleToIntersection("A", new Vehicle("V1", false, System.currentTimeMillis()));

simulator.addVehicleToIntersection("A", new Vehicle("V2", true, System.currentTimeMillis()));

simulator.addVehicleToIntersection("A", new Vehicle("V3", false, System.currentTimeMillis()));

simulator.addVehicleToIntersection("A", new Vehicle("V4", false, System.currentTimeMillis()));

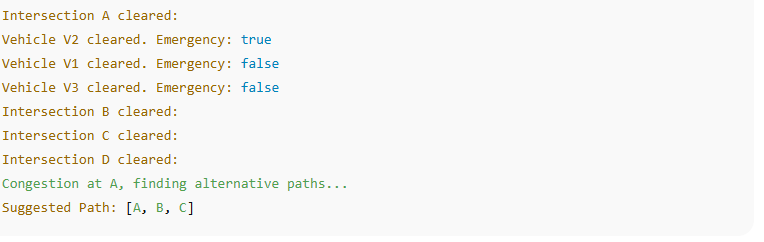
simulator.simulateTraffic();

simulator.rerouteCongestedPaths();

}

}

**Output**:



|  |  |  |
| --- | --- | --- |
| **CCP Attributes mapped** | | |
| **Attributes of Complex Problem Solving** | | **Justification** |
| **WP** | **Description** | **DSA CCP Mapping** |
| **WP1** | Depth of Knowledge | Requires advanced knowledge of data structures, Java programming, and optimization techniques. |
| **WP2** | Range of Conflicting Requirements | Handles conflicts between vehicle prioritization and traffic flow efficiency. |
| **WP3** | Depth of Analysis | Demands abstract thinking and innovative algorithms to model and optimize traffic flow. |
| **WP4** | Familiarity of Issues | Involves unique problems like real-time traffic management and dynamic congestion control. |