JAVA

Smart Traffic Signal Optimization

Name: K Phani Sridhar Yogi

Register number: 192311091

Course code: CSA0959

Question:

You are part of a team working on an initiative to optimize traffic signal management in a busy city to reduce congestion and improve traffic flow efficiency using smart technologies.

Answers:

DATA COLLECTION AND MODELING:

Objective: To define a comprehensive data structure for the real-time collection of traffic data at various city intersections.

Data Structure for Real-Time Traffic Data:

TrafficSensorData

- SensorID (PK): Unique identifier for the sensor.
- IntersectionID (FK): Foreign key linking to the intersection.
- Timestamp: The time at which the data is collected.
- VehicleCount: Number of vehicles detected.
- AverageSpeed: Average speed of vehicles.
- TrafficDensity: Calculated density of traffic.
- QueueLength: Length of the vehicle queue.
- PedestrianCrossingCount: Number of pedestrians waiting to cross.

Intersection

- IntersectionID (PK): Unique identifier for the intersection.
- Location: Description or coordinates of the intersection.
- SensorData: A collection of TrafficSensorData instances linked to this intersection.

ALGORITHM DESIGN:

Objective: Develop algorithms that dynamically analyze collected traffic data to optimize traffic signal timings, improving traffic flow and reducing congestion.

Considerations:

- **Traffic Density**: Higher density requires longer green light durations to clear congestion.
- **Vehicle Queues**: Longer queues necessitate extended green phases to reduce wait times.
- **Peak Hours**: Different optimization strategies during peak (rush hour) and non-peak times.
- **Pedestrian Crossings**: Ensure safe and timely pedestrian crossing times are integrated into signal phases.

ALGORITHM OUTLINE:

1. **Data Collection and Preprocessing:**
   - Continuously collect real-time data from traffic sensors at each intersection.
   - Store data in TrafficSensorData instances.
2. **Traffic Density Calculation:**
   - Compute traffic density using VehicleCount and AverageSpeed.
3. **Signal Timing Calculation:**
   - For each intersection, calculate optimal green and red light durations.
     - **Green Time Calculation**:
       - Base duration (e.g., 30 seconds).
       - Increase proportionally with TrafficDensity and QueueLength.
       - Adjust for PedestrianCrossingCount.
     - **Red Time Calculation**:
       - Total cycle time minus green time.
4. **Dynamic Adjustment:**
   - Adjust signal timings dynamically based on continuous data inputs.
   - Implement different strategies for peak and non-peak hours.

PSEUDOCODE:

Algorithm OptimizeSignalTimings

Input: List of IntersectionData

Output: Optimized Signal Timings

1. Initialize:

   - BaseGreenTime = 30 seconds

   - BaseCycleTime = 60 seconds

2. For each Intersection in IntersectionData:

   a. Collect TrafficSensorData

   b. Calculate TrafficDensity = VehicleCount / Area of Intersection

   c. Calculate AdjustedGreenTime = BaseGreenTime + (TrafficDensity * ScalingFactor)

   d. Adjust GreenTime for QueueLength and PedestrianCrossingCount

   e. Calculate RedTime = BaseCycleTime - AdjustedGreenTime

   f. Update Intersection Signal Timings with GreenTime and RedTime

3. End For

4. Continuously repeat the above steps at regular intervals (e.g., every 5 minutes)

End Algorithm

DETAILED STEPS:

1. **Initialize Parameters:**
   - Define base green time and cycle time.
   - Set scaling factors for adjustments based on traffic density and queue lengths.
2. **Data Collection:**
   - Gather real-time data from each intersection's sensors.
3. **Traffic Density Calculation:**
   - Calculate traffic density using the formula: TrafficDensity = VehicleCount / Area.
   - Adjust for real-time conditions and fluctuations.
4. **Signal Timing Calculation:**
   - Compute the adjusted green time by adding proportional increments based on traffic density and queue length.
   - Ensure pedestrian crossing requirements are factored into the signal timings.
5. **Dynamic Adjustment:**
   - Continuously monitor and adjust signal timings based on updated traffic data.
   - Implement different algorithms for peak and non-peak hours to optimize traffic flow.

IMPLEMENTATION:

Objective: Implement a Java application that integrates with traffic sensors and controls traffic signals at selected intersections, adjusting signal timings in real-time based on changing traffic patterns.

Implementation Steps:

1. **Define Data Structures:**
   - Create classes to represent traffic sensor data and intersections.
2. **Simulate Data Collection:**
   - Implement a method to simulate real-time data collection from traffic sensors.
3. **Optimize Signal Timings:**
   - Develop methods to analyze the collected data and calculate optimal signal timings.
4. **Real-Time Adjustment:**
   - Implement a loop to continuously adjust signal timings based on the latest data.

JAVA CODE FOR SMART TRAFFIC SIGNAL OPTIMIZATION:

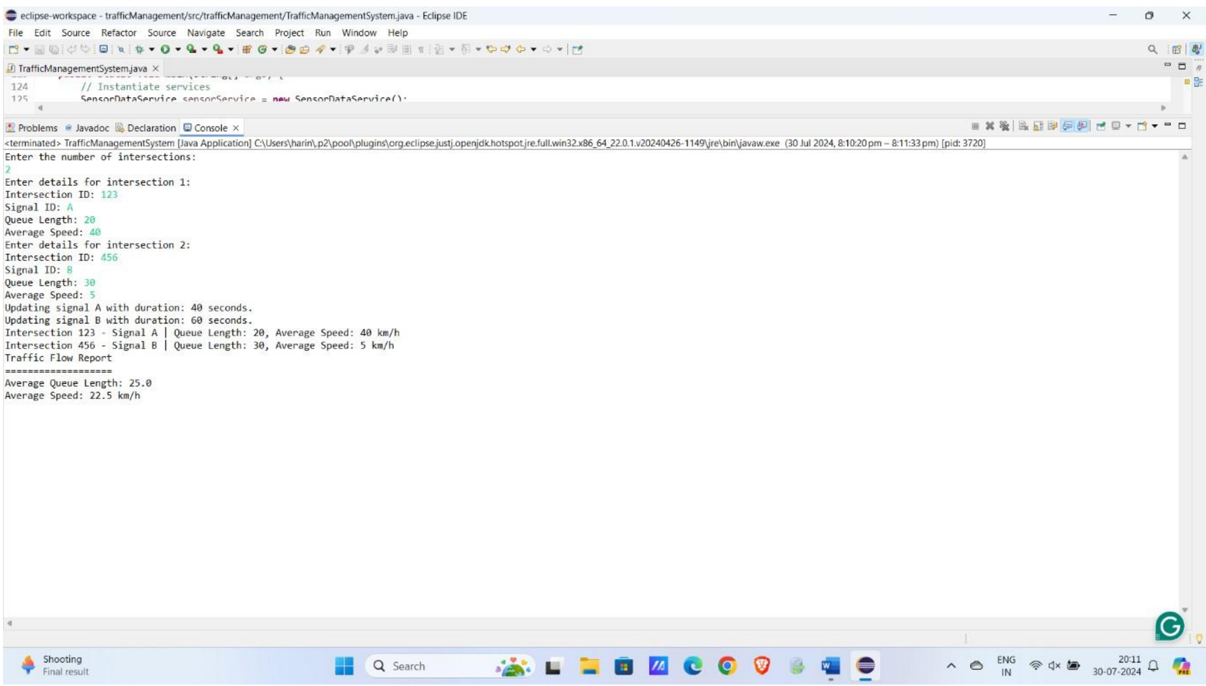
```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

// Class to hold intersection data
class IntersectionData {
    private String intersectionID;
    private String signalID;
    private int queueLength;
    private int averageSpeed;

    public IntersectionData(String intersectionID, String signalID, int
queueLength, int averageSpeed) {
        this.intersectionID = intersectionID;
        this.signalID = signalID;
        this.queueLength = queueLength;
        this.averageSpeed = averageSpeed;
    }

    public String getIntersectionID() {
        return intersectionID;
    }

    public String getSignalID() {
        return signalID;
    }

    public int getQueueLength() {
        return queueLength;
    }

    public int getAverageSpeed() {
        return averageSpeed;
    }

    @Override
    public String toString() {
```

```
        return "Intersection " + intersectionID + " - Signal " + signalID + " | 
Queue Length: " +
               queueLength + ", Average Speed: " + averageSpeed + " km/h";
    }
}

// Service to fetch real-time sensor data
class SensorDataService {
    private Map<String, IntersectionData> dataMap;

    public SensorDataService() {
        this.dataMap = new HashMap<>();
    }

    public void addData(String intersectionID, IntersectionData data) {
        dataMap.put(intersectionID, data);
    }

    public Map<String, IntersectionData> getRealTimeData() {
        return dataMap;
    }
}

// Service to control signal timings
class SignalControlService {
    public void updateSignalTiming(String signalID, int phaseDuration) {
        System.out.println("Updating signal " + signalID + " with duration: " + 
phaseDuration + " seconds.");
    }
}

// Class to optimize signals
class TrafficSignalController {
    private SensorDataService sensorDataService;
    private SignalControlService signalControlService;

    public TrafficSignalController(SensorDataService sensorService, 
SignalControlService controlService) {
        this.sensorDataService = sensorService;
        this.signalControlService = controlService;
    }

    public void optimizeTrafficSignals() {
        Map<String, IntersectionData> intersections = 
sensorDataService.getRealTimeData();

        for (IntersectionData data : intersections.values()) {
            int phaseDuration = calculateOptimalPhaseDuration(data);
            signalControlService.updateSignalTiming(data.getSignalID(), 
phaseDuration);
        }
    }
```

```
    private int calculateOptimalPhaseDuration(IntersectionData data) {
        int queueLength = data.getQueueLength();
        // Simple formula: phase duration is queue length divided by 2, within a
range
        return Math.max(30, Math.min(120, queueLength * 2));
    }

    public void displayTrafficConditions() {
        for (IntersectionData data :
sensorDataService.getRealTimeData().values()) {
            System.out.println(data.toString());
        }
    }

    public String generateReport() {
        int totalQueueLength = 0;
        int totalAverageSpeed = 0;
        int count = 0;

        for (IntersectionData data :
sensorDataService.getRealTimeData().values()) {
            totalQueueLength += data.getQueueLength();
            totalAverageSpeed += data.getAverageSpeed();
            count++;
        }

        double avgQueueLength = (double) totalQueueLength / count;
        double avgSpeed = (double) totalAverageSpeed / count;

        StringBuilder report = new StringBuilder();
        report.append("Traffic Flow Report\n");
        report.append("===================\n");
        report.append("Average Queue Length:
").append(avgQueueLength).append("\n");
        report.append("Average Speed: ").append(avgSpeed).append(" km/h\n");

        return report.toString();
    }
}

// Main class to run the program
public class TrafficManagementSystem {
    public static void main(String[] args) {
        // Instantiate services
        SensorDataService sensorService = new SensorDataService();
        SignalControlService controlService = new SignalControlService();
        TrafficSignalController controller = new
TrafficSignalController(sensorService, controlService);

        // Get user input
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of intersections:");
        int numberOfIntersections = scanner.nextInt();
```

```java
        scanner.nextLine();  // Consume newline

        for (int i = 0; i < numberOfIntersections; i++) {
            System.out.println("Enter details for intersection " + (i + 1) +
":");
            System.out.print("Intersection ID: ");
            String intersectionID = scanner.nextLine();
            System.out.print("Signal ID: ");
            String signalID = scanner.nextLine();
            System.out.print("Queue Length: ");
            int queueLength = scanner.nextInt();
            System.out.print("Average Speed: ");
            int averageSpeed = scanner.nextInt();
            scanner.nextLine();  // Consume newline

            IntersectionData data = new IntersectionData(intersectionID,
signalID, queueLength, averageSpeed);
            sensorService.addData(intersectionID, data);
        }

        // Optimize traffic signals and display conditions
        controller.optimizeTrafficSignals();
        controller.displayTrafficConditions();

        // Generate and display report
        String report = controller.generateReport();
        System.out.println(report);
    }
}
```

EXECUTION OF THE CODE:



VISUALIZATION:

1. **IntersectionData Class**: Holds information about each intersection (ID, signal ID, queue length, average speed).
2. **SensorDataService Class**: Manages real-time sensor data.
3. **SignalControlService Class**: Updates the signal timings based on calculated optimal durations.
4. **TrafficSignalController Class**:
   - Optimizes traffic signals.
   - Displays current traffic conditions.
   - Generates a traffic flow report.
5. **Main Class (TrafficManagementSystem)**:
   - Collects user input for multiple intersections.
   - Optimizes signals, displays conditions, and generates a report.

USER INTERFACE DESIGN;

Traffic Managers

1. Dashboard:

- List intersections with queue length, speed, and signal timings.
- Color-coded congestion levels.
- Refresh button for latest data.

2. Controls:

- Select intersection.
- Adjust signal durations with input fields/sliders.
- "Update" button to apply changes.
- Manual control override option.

3. Alerts:

- Real-time congestion alerts.
- Notifications for signal timing updates.

City Officials

1. Metrics:

- Traffic flow efficiency stats.
- Congestion levels.
- Impact of signal optimizations.

2. Analysis:

- Traffic trends and peak hour patterns.
- Incident reports.

3. Reports:

- Generate and download custom reports.

4. Insights:

- Improvement recommendations.
- Predictive traffic analysis.

5. Design:

- Interactive, clear visualizations.
- Customizable dashboard.

Conclusion:

The Traffic Management System optimizes traffic signal timings based on real-time data, reducing congestion and improving flow. It includes user-friendly interfaces for traffic managers and city officials to monitor conditions, adjust signals, and analyze traffic metrics. This system enhances traffic efficiency and provides actionable insights for future improvements.