JAVA ASSIGNMENT

M.Mahathi

192324098

This project comprises a number of crucial processes, including user interaction, algorithm design, implementation, and data collection. Let's examine each task and the necessary deliverables in detail:

**1. Data Collection and Modeling**

Data Structure Definition

- Data Points: This includes vehicle counts, speeds, traffic density, vehicle queues, peak hours, pedestrian crossings, weather conditions, and accident reports.

- Sensors: These comprise traffic cameras, inductive loop sensors, radar sensors, and pedestrian crossing buttons.

Sample Data Structure:

Java code:

class TrafficData {

int intersectionId;

long timestamp;

int vehicleCount;

double averageSpeed;

int pedestrianCount;

boolean isPeakHour;

String weatherCondition;

int accidentCount;

// Constructor, getters, and setters

}

**2. Algorithm Design**

Optimization Algorithm

- Inputs: Real-time traffic data

- Outputs: Adjusted signal timings

Factors to Consider:

- Traffic density

- Vehicle queues

- Peak hours

- Pedestrian crossings

Pseudocode:

Algorithm OptimizeSignalTiming

Input: Real-time traffic data

Output: Adjusted signal timings

Initialize priorityQueue as empty

For each intersection in traffic data:

Calculate traffic density = vehicleCount / averageSpeed

Add intersection with traffic density to priorityQueue

While priorityQueue is not empty:

intersection = ExtractMax(priorityQueue)

Adjust signal timings for intersection based on traffic density, vehicle queues, and pedestrian crossings

Return adjusted signal timings

End Algorithm

**3. Implementation**

Java Application

Java Code:

import java.util.\*;

class TrafficSignalOptimizer {

static class TrafficData {

int intersectionId;

long timestamp;

int vehicleCount;

double averageSpeed;

int pedestrianCount;

boolean isPeakHour;

String weatherCondition;

int accidentCount;

// Constructor, getters, and setters

}

public Map<Integer, Integer> optimizeSignalTiming(List<TrafficData> trafficDataList) {

PriorityQueue<TrafficData> priorityQueue = new PriorityQueue<>((a, b) -> {

double densityA = a.vehicleCount / a.averageSpeed;

double densityB = b.vehicleCount / b.averageSpeed;

return Double.compare(densityB, densityA);

});

for (TrafficData data : trafficDataList) {

priorityQueue.add(data);

}

Map<Integer, Integer> signalTimings = new HashMap<>();

while (!priorityQueue.isEmpty()) {

TrafficData data = priorityQueue.poll();

int newTiming = calculateSignalTiming(data);

signalTimings.put(data.intersectionId, newTiming);

}

return signalTimings;

}

private int calculateSignalTiming(TrafficData data) {

// Logic to calculate new signal timing based on traffic data

// Example: return 60; (60 seconds green light)

return 60;

}

public static void main(String[] args) {

// Sample main method to demonstrate the optimizer

List<TrafficData> trafficDataList = Arrays.asList(

new TrafficData(/\* sample data \*/),

new TrafficData(/\* sample data \*/)

);

TrafficSignalOptimizer optimizer = new TrafficSignalOptimizer();

Map<Integer, Integer> timings = optimizer.optimizeSignalTiming(trafficDataList);

// Output or further processing of the signal timings

}

}

**4. Visualization and Reporting**

Real-time Monitoring and Reporting

-Tools: Utilize JavaFX or web technologies (HTML, CSS, JavaScript) for the UI.

-Libraries: Use libraries like JFreeChart for visualizations.

**5. User Interaction**

**User Interface Design:**

- Traffic Manager Interface: Monitor traffic conditions, adjust signal timings.

- City Officials Dashboard: View performance metrics and historical data.

**Sample UI Components:**

-Real-time Traffic Map: Displays current traffic conditions at intersections.

-Control Panel: Allows manual adjustment of signal timings.

-Performance Dashboard: Shows traffic flow improvements, average wait times, congestion reduction.

Deliverables

1. Data Flow Diagram:

- Illustrates the flow of real-time traffic data from sensors to the optimization system and the adjustment of traffic signals.

2. Pseudocode and Implementation:

- Detailed pseudocode and Java code for the algorithms.

- Explanation of data structures and design decisions.

3. Documentation:

- Design decisions behind algorithms.

- Assumptions (e.g., sensor reliability).

- Potential improvements.

4. User Interface:

- Intuitive interfaces for traffic managers and city officials.

5. Testing:

- Comprehensive test cases to validate functionality under various scenarios.

This structured approach will help in systematically addressing the problem and ensuring a robust traffic signal optimization system. If you need further details on any specific section, let me know!