**Smart Traffic Signal Optimization**

**Name: CHUNDURI NAGA SAI SRIKANTH**

**Reg no:192311159**

**Tasks: 1. Data Collection and Modeling**

**Data Structure for Real-Time Traffic Data:**

**- \*\*TrafficSensorData\*\***

**- SensorID (PK)**

**- IntersectionID (FK)**

**- Timestamp**

**- VehicleCount**

**- AverageSpeed**

**- TrafficDensity**

**- QueueLength**

**- PedestrianCrossingCount**

**2. Algorithm Design:**

**Algorithm for Dynamic Signal Timing Optimization:**

**- \*\*Inputs:\*\***

**- Real-time traffic data (vehicle counts, speeds, density, queue length, pedestrian counts)**

**- Historical traffic patterns**

**- Time of day (peak hours vs. off-peak hours)**

**- \*\*Outputs:\*\***

**- Optimized signal timings (Green, Yellow, Red durations)**

**Pseudocode:**

**```plaintext**

**Algorithm OptimizeSignalTimings**

**Input: realTimeData, historicalData, timeOfDay**

**Output: signalTimings**

**1. Initialize signaltimings**

**2. For each intersection:**

**3. Retrieve current traffic data from realTimeData**

**4. Calculate traffic density and queue length**

**5. Determine if pedestrian crossing is active**

**6. Retrieve historical traffic patterns from historicalData**

**7. If current traffic density > threshold or queue length > threshold:**

**8. Increase green light duration**

**9. If pedestrian crossing is active:**

**10. Allocate sufficient time for pedestrian crossing**

**11. Else:**

**12. Adjust green, yellow, and red light durations based on historical patterns and timeOfDay**

**13. Return signalTimings ```**

**3. Implementation:**

**Java Code for Algorithm and Application Integration:**

**```java**

**import java.util.Map;**

**import java.util.HashMap;**

**import java.time.LocalTime;**

**public class TrafficSignalOptimizer {**

**private Map <string,intersection>intersections;**

**public TrafficSignalOptimizer() {**

**intersections = new HashMap<>();**

**}**

**public void optimizeSignalTimings() {**

**for (Intersection intersection : intersections.values()) {**

**TrafficData currentData = intersection.getCurrentTrafficData();**

**HistoricalData historicalData = intersection.getHistoricalTrafficData();**

**LocalTime timeOfDay = LocalTime.now();**

**SignalTimings newTimings = calculateOptimizedTimings(currentData, historicalData, timeOfDay);**

**intersection.updateSignalTimings(newTimings);**

**}**

**}**

**private SignalTimings calculateOptimizedTimings(TrafficData currentData, HistoricalData historicalData, LocalTime timeOfDay) {**

**SignalTimings timings = new SignalTimings(); if(currentData.getTrafficDensity()>THRESHOLD||currentData.getQueueLength() > THRESHOLD) {**

**timings.increaseGreenDuration();**

**}**

**if (currentData.isPedestrianCrossingActive()) {**

**timings.allocatePedestrianCrossingTime();**

**} else {**

**timings.adjustBasedOnHistoricalPatterns(historicalData, timeOfDay);**

**}**

**return timings;  
}  
}**

**class intersection{**

**private string id;**

**private traffic data current traffic data;**

**private historical data historical data;**

**private signal timings signal timings;**

**public traffic data get current trafficdata(){**

**}**

**Public historicaldata gethistoricaltrafficdata(){**

**}**

**Public void updatesugnaltimings(signaltimingsnewtimings**

**}**

**}**

**Class trafficdata{**

**Private int vehiclecount;**

**Private double averagespeed;**

**Private double trafficdensity;**

**Private int queuelength;**

**Private Boolean pedestriancrossingactive;**

**}**

**Class historicaldata{**

**}**

**Class signaltimings{**

**private int greenduration;**

**private int yellowduration;**

**private int redduration;**

**public void increasegreenduration(){**

**}**

**Public void allocatepedestraincrossingtime){**

**}**

**Public void adjustbasedonhistoricalpatterns(historicaldata historicaldata,localtime timeofday){**

**}**

**}**

**4.visualization and reporting:**

**Visualizations:**

**- \*\*Real-Time Traffic Monitoring Dashboard:**

**\*\* - Map view with intersections highlighted**

**- Current traffic density and queue lengths**

**- Signal statuses and timings**

**- \*\*Reports:\*\***

**- Traffic flow improvements (before vs. after)**

**- Average wait times at intersections**

**- Congestion reduction metrics  
#### Reporting Code Example:**

**```java**

**public class TrafficReportGenerator {**

**public void generateTrafficFlowReport(List <intersection>intersections) {**

**}**

**public void generateWaitTimeReport(List<intersection> intersections) {**

**}**

**public void generateCongestionReductionReport(List<intersection> intersections) {**

**}**

**}**

**5. User Interaction:**

**User Interface (UI) Design:**

**- \*\*Traffic Manager Interface:\*\***

**- Real-time monitoring of intersections**

**- Manual override to adjust signal timings**

**- Alerts for unusual traffic conditions –**

**\*\*City Official Dashboard:\*\***

**- Performance metrics visualization**

**- Historical data and trend analysis**

**- Reports on traffic management effectiveness  
UI Example:**

**Java**

**public class TrafficManagerUI {**

**public static void main(String[] args) {**

**}**

**private void initializeUI() {**

**}**

**private void displayRealTimeData() {**

**}**

**private void allowManualOverride() {**

**}**

**private void showAlerts() {**

**}**

**}**

**public class CityOfficialDashboard {**

**public static void main(String[] args) {**

**}**

**private void initializeDashboard() {**

**}**

**private void displayPerformanceMetrics() {**

**}**

**private void showHistoricalData() {**

**}**

**private void generateReports() {**

**}**

**}**

**``` Testing:**

**Test Cases:**

**1. \*\*Functional Tests:\*\***

**- Verify real-time data collection from sensors**

**- Validate signal timing adjustments based on traffic conditions**

**- Ensure manual override functionality works**

**2. \*\*Performance Tests:**

**\*\* - Test system response under high traffic conditions**

**- Measure time taken to adjust signal timings**

**3. \*\*Integration Tests:\*\***

**- Verify integration with traffic sensors**

**- Ensure data flow from sensors to the optimization algorithm and signal controllers**

**4. \*\*User Interface Tests:\*\***

**- Validate usability of traffic manager interface**

**- Ensure city official dashboard displays accurate metrics and reports Test**

**Example:**

**public class TrafficSignalOptimizerTest {**

**public void testSignalTimingOptimization() {**

**TrafficData testData = new TrafficData();**

**HistoricalData historicalData = new HistoricalData();**

**LocalTime timeOfDay = LocalTime.of(8, 0);**

**TrafficSignalOptimizer optimizer = new TrafficSignalOptimizer();**

**SignalTimings timings = optimizer.calculateOptimizedTimings(testData, historicalData, timeOfDay);**

**assertEquals(expectedGreenDuration, timings.getGreenDuration());**

**assertEquals(expectedYellowDuration, timings.getYellowDuration());**

**assertEquals(expectedRedDuration, timings.getRedDuration());**

**}**

**}**

**``` Deliverable:**

**1. \*\*Data Flow Diagram:\*\***

**- Illustrate the flow from traffic data collection, analysis, and optimization to signal timing adjustments.**

**2. \*\*Pseudocode and Implementation:\*\***

**- Provide detailed pseudocode and Java code for the traffic signal optimization algorithms.**

**3. \*\*Documentation:\*\***

**- Explain design decisions, data structures, assumptions, and potential improvements.**

**4. \*\*User Interface:\*\***

**- Develop interfaces for traffic managers and city officials.**

**5. \*\*Testing:\*\* - Include comprehensive test cases to validate the system**