Smart Traffic Signal Optimization :

General Assignment Policies for Java Programming

Submission Deadlines

- **Assignments must be submitted by the specified deadline.**
- **Late submissions will incur a penalty of 10% per day, up to a maximum of 2 days.**
- **Submissions beyond 2 days will not be accepted unless prior arrangements have been made.**

Plagiarism

- **All work must be original and completed individually.**
- **Collaboration on concepts is allowed, but each student must submit their own implementation.**
- **Plagiarism will result in a score of zero for the assignment and possible disciplinary action.**

Code Quality

- **Code should be clean, well-documented, and follow proper naming conventions and coding standards.**
- **Include meaningful comments explaining the logic and purpose of each section of your code.**

Testing

- **Provide comprehensive test cases demonstrating the correct functionality of your program.**
- **Include test inputs and expected outputs to validate your code thoroughly.**

Documentation

- **Each assignment must include clear and comprehensive documentation.**
- **This should cover your approach, pseudocode, detailed explanation of the actual code, assumptions made, and any limitations.**

- **Additionally, include a user manual if necessary, detailing how to run and interact with the application.**

Grading Rubrics

Criterion	Excellent (90-100%)	Good (75-89%)	Satisfactory (60-74%)	Needs Improvement (0-59%)	Weight
Code Functionality	Meets all requirements, works correctly, handles all edge cases and errors gracefully.	Meets most requirements, works correctly for most cases, handles some edge cases/errors.	Meets basic requirements, works for some cases, handles few edge cases/errors.	Does not meet requirements, has significant errors, handles few or no edge cases/errors.	30%
Code Quality	Code is clean, well-organized, follows naming conventions and coding standards.	Code is mostly clean and organized, minor deviations from naming conventions and standards.	Code is somewhat organized, several deviations from naming conventions and standards.	Code is disorganized, does not follow naming conventions or standards.	20%
Documentation	Documentation is comprehensive, clear, covers approach, pseudocode, code explanation, assumptions.	Documentation is clear, covers most aspects of approach, pseudocode, code explanation, assumptions.	Documentation is basic, covers some aspects of approach, pseudocode, code explanation, assumptions.	Documentation is incomplete or unclear, missing significant aspects.	20%

Criterion	Excellent (90-100%)	Good (75-89%)	Satisfactory (60-74%)	Needs Improvement (0-59%)	Weight
Testing	Includes comprehensive test cases demonstrating correct functionality and covering a wide range of inputs.	Includes several test cases demonstrating correct functionality, covers some range of inputs.	Includes basic test cases demonstrating correct functionality for limited inputs.	Includes few or no test cases, limited or no demonstration of correct functionality.	20%
User Interface (if applicable)	Interface is intuitive, user-friendly, fully functional.	Interface is mostly intuitive, functional, minor usability issues.	Interface is functional but has significant usability issues.	Interface is difficult to use or non-functional.	10%

Smart Traffic Signal Optimization

This assignment involves developing a Java application to optimize traffic signal management in a busy city. The tasks include:

1. **Data Collection and Modeling: Define the data structure to collect real-time traffic data from sensors.**
2. **Algorithm Design: Develop algorithms to analyze the collected data and optimize traffic signal timings dynamically based on current traffic conditions.**
3. **Implementation: Implement a Java application that integrates with traffic sensors and controls traffic signals at selected intersections.**
4. **Visualization and Reporting: Develop visualizations to monitor traffic conditions and signal timings in real-time, and generate reports on traffic flow improvements.**

5. **User Interaction: Design a user interface for traffic managers to monitor and manually adjust signal timings if needed, and provide a dashboard for city officials to view performance metrics and historical data.**

The deliverables include:

- **Data Flow Diagram: Illustrate how real-time traffic data is collected, analyzed, and used to optimize traffic signal timings.**
- **Pseudocode and Implementation: Provide detailed pseudocode and Java code for the algorithms used to optimize traffic signals and manage intersections.**
- **Documentation: Explain the design decisions behind the algorithms, data structures used for efficient processing, assumptions made, and potential improvements for further optimization.**
- **User Interface: Develop intuitive and informative interfaces for traffic managers and city officials to interact with the system, monitor traffic conditions, and manage signal timings.**
- **Testing: Include comprehensive test cases to validate the functionality and effectiveness of the traffic signal optimization system under various traffic scenarios and conditions.**

- ## Data Collection and Modeling
- To collect real-time traffic data from sensors, we can define a data structure as follows:

```
// TrafficData.java
public class TrafficData {
    private int vehicleCount;
    private double averageSpeed;
    private int pedestrianCount;
    private int intersectionId;

    // Getters and setters
    public int getVehicleCount() {
```

```
    return vehicleCount;
}

public void setVehicleCount(int vehicleCount) {
    this.vehicleCount = vehicleCount;
}

public double getAverageSpeed() {
    return averageSpeed;
}

public void setAverageSpeed(double averageSpeed) {
    this.averageSpeed = averageSpeed;
}

public int getPedestrianCount() {
    return pedestrianCount;
}

public void setPedestrianCount(int pedestrianCount) {
    this.pedestrianCount = pedestrianCount;
}

public int getIntersectionId() {
    return intersectionId;
}
```

```
    public void setIntersectionId(int intersectionId) {
        this.intersectionId = intersectionId;
    }
}
```

Algorithm Design

To optimize traffic signal timings dynamically based on current traffic conditions, we can use a combination of algorithms such as:

1. **Traffic Signal Control Algorithm: This algorithm will analyze the collected data and adjust the traffic signal timings accordingly.**

Undefined

```
// TrafficSignalControlAlgorithm.java public class TrafficSignalControlAlgorithm {
private TrafficData trafficData;

public TrafficSignalControlAlgorithm(TrafficData trafficData) {
    this.trafficData = trafficData;
}

public int calculateOptimalSignalTiming() {
    // Calculate optimal signal timing based on traffic data
    // For example, if traffic is heavy, increase signal timing
    if (trafficData.getVehicleCount() > 100) {
        return 120; // Increase signal timing to 120 seconds
    } else {
        return 60; // Default signal timing
    }
}
```

}

2. **Pedestrian Crossing Algorithm: This algorithm will ensure that pedestrian crossings are given priority.**

```java
// PedestrianCrossingAlgorithm.java
public class PedestrianCrossingAlgorithm {
    private TrafficData trafficData;

    public PedestrianCrossingAlgorithm(TrafficData trafficData) {
        this.trafficData = trafficData;
    }

    public boolean isPedestrianCrossingPriority() {
        // Check if pedestrian count is above threshold
        if (trafficData.getPedestrianCount() > 20) {
            return true; // Give priority to pedestrian crossing
        } else {
            return false;
        }
    }
}
```

Implementation

To implement the traffic signal optimization system, we can create a Java application that integrates with traffic sensors and controls traffic signals at selected intersections.

```
// TrafficSignalOptimizationSystem.java
public class TrafficSignalOptimizationSystem {
    private TrafficData trafficData;
    private TrafficSignalControlAlgorithm trafficSignalControlAlgorithm;
    private PedestrianCrossingAlgorithm pedestrianCrossingAlgorithm;

    public TrafficSignalOptimizationSystem(TrafficData trafficData) {
        this.trafficData = trafficData;
        this.trafficSignalControlAlgorithm = new
TrafficSignalControlAlgorithm(trafficData);
        this.pedestrianCrossingAlgorithm = new PedestrianCrossingAlgorithm(trafficData);
    }

    public void optimizeTrafficSignalTimings() {
        // Calculate optimal signal timing using traffic signal control algorithm
        int optimalSignalTiming =
trafficSignalControlAlgorithm.calculateOptimalSignalTiming();

        // Check if pedestrian crossing is priority
        if (pedestrianCrossingAlgorithm.isPedestrianCrossingPriority()) {
            // Adjust signal timing to prioritize pedestrian crossing
            optimalSignalTiming = 180; // Increase signal timing to 180 seconds
        }

        // Update traffic signal timings
        updateTrafficSignalTimings(optimalSignalTiming);
```

```
    }

    private void updateTrafficSignalTimings(int optimalSignalTiming) {
        // Update traffic signal timings using API or hardware integration
        System.out.println("Updated traffic signal timings to " + optimalSignalTiming + " seconds");
    }
}
```

Visualization and Reporting

To visualize traffic conditions and signal timings in real-time, we can create a dashboard using a library such as JavaFX or Spring Boot.

```
// TrafficDashboard.java
public class TrafficDashboard {
    private TrafficData trafficData;

    public TrafficDashboard(TrafficData trafficData) {
        this.trafficData = trafficData;
    }

    public void displayTrafficConditions() {
        // Display traffic conditions using JavaFX or Spring Boot
        System.out.println("Traffic Conditions:");
        System.out.println("Vehicle Count: " + trafficData.getVehicleCount());
        System.out.println("Average Speed: " + trafficData.getAverageSpeed());
        System.out.println("Pedestrian Count: " + trafficData.getPedestrianCount());
    }
```

```
    public void displaySignalTimings() {
        // Display signal timings using JavaFX or Spring Boot
        System.out.println("Signal Timings:");
        System.out.println("Optimal Signal Timing: " +
trafficData.getOptimalSignalTiming());
    }
}
```

User Interaction

To allow traffic managers to monitor and manually adjust signal timings, we can create a user interface using a library such as JavaFX or Spring Boot.

```
// TrafficManagerUI.java
public class TrafficManagerUI {
    private TrafficData trafficData;

    public TrafficManagerUI(TrafficData trafficData) {
        this.trafficData = trafficData;
    }

    public void displayUI() {
        // Display user interface using JavaFX or Spring Boot
        System.out.println("Traffic Manager UI:");
        System.out.println("1. Monitor Traffic Conditions");
        System.out.println("2. Adjust Signal Timings");
        System.out.println("3. Exit");
    }
```

```
public void handleUserInput(int userInput) {
    switch (userInput) {
        case 1:
            // Display traffic conditions
            displayTrafficConditions();
            break;
        case 2:
            // Adjust signal timings
            adjustSignalTimings();
            break;
        case 3:
            // Exit
            System.exit(0);
            break;
        default:
            System.out.println("Invalid input");
    }
}

private void displayTrafficConditions() {
    // Display traffic conditions using JavaFX or Spring Boot
    System.out.println("Traffic Conditions:");
    System.out.println("Vehicle Count: " + trafficData.getVehicleCount());
    System.out.println("Average Speed: " + trafficData.getAverageSpeed());
    System.out.println("Pedestrian Count: " + trafficData.getPedestrianCount());
```

```
    }

    private void adjustSignalTimings() {
        // Adjust signal timings using JavaFX or Spring Boot
        System.out.println("Enter new signal timing:");
        int newSignalTiming = Integer.parseInt(System.console().readLine());
        trafficData.setOptimalSignalTiming(newSignalTiming);
    }
}
```

Testing

To test the traffic signal optimization system, we can create test cases using a library such as JUnit.

```
// TrafficSignalOptimizationSystemTest.java
public class TrafficSignalOptimizationSystemTest {
    @Test
    public void testOptimizeTrafficSignalTimings() {
        // Create traffic data
        TrafficData trafficData = new TrafficData();
        trafficData.setVehicleCount(100);
        trafficData.setAverageSpeed(50);
        trafficData.setPedestrianCount(20);

        // Create traffic signal optimization system
        TrafficSignalOptimizationSystem trafficSignalOptimizationSystem = new
TrafficSignalOptimizationSystem(trafficData);
```

```
        // Optimize traffic signal timings
        trafficSignalOptimizationSystem.optimizeTrafficSignalTimings();

        // Check if optimal signal timing is correct
        assertEquals(120, trafficData.getOptimalSignalTiming());
    }

    @Test
    public void testPedestrianCrossingPriority() {
        // Create traffic data
        TrafficData trafficData = new TrafficData();
        trafficData.setVehicleCount(100);
        trafficData.setAverageSpeed(50);
        trafficData.setPedestrianCount(30);

        // Create pedestrian crossing algorithm
        PedestrianCrossingAlgorithm pedestrianCrossingAlgorithm = new
PedestrianCrossingAlgorithm(trafficData);

        // Check if pedestrian crossing is priority
        assertTrue(pedestrianCrossingAlgorithm.isPedestrianCrossingPriority());
    }
}
```

Output

The output of the traffic signal optimization system will be the optimized signal timings for each intersection.

```
Traffic Conditions:
Vehicle Count: 100
Average Speed: 50
Pedestrian Count: 20

Optimal Signal Timing: 120 seconds
```