JAVA

Smart Traffic Signal Optimization

Name: Dharshini.J

Register number: 192324059

Course code: CSA0959

Data Collection and Modeling

Objective : Establish an extensive and efficient data structure for real-time traffic data collection across city intersections. This data is critical for understanding traffic patterns and making informed decisions to optimize traffic signal timings.

Data Structure for Real-Time Traffic Data

TrafficSensorData

- **SensorID (PK)** :A unique identifier assigned to each sensor.
- **IntersectionID (FK):** A foreign key that links the sensor data to a specific intersection.
- **Timestamp:** The precise time at which the data is recorded.
- **VehicleCount:** The total number of vehicles detected by the sensor.
- **AverageSpeed**: The average speed of vehicles detected, which helps gauge traffic flow.
- **TrafficDensity:** A calculated measure of how congested the traffic is, derived from VehicleCount and road area.
- **QueueLength**: The length of the vehicle queue, indicating the number of vehicles waiting at the intersection.
- **PedestrianCrossingCount:** The number of pedestrians waiting to cross the street, important for ensuring pedestrian safety.

Intersection

- **IntersectionID (PK):** A unique identifier for each intersection, enabling precise data mapping.

- **Location:** A description or geographical coordinates of the intersection, facilitating data analysis and decision-making.
- **SensorData:** A collection of TrafficSensorData instances associated with this intersection, providing a detailed traffic profile.

Algorithm Design

Objective: Develop robust algorithms that analyze real-time traffic data to dynamically adjust traffic signal timings. The goal is to enhance traffic flow efficiency and reduce congestion by adapting to varying traffic conditions.

Key Considerations

- **Traffic Density**: Adjust green light durations to accommodate higher traffic volumes, especially during peak times.
- **Vehicle Queues**: Lengthier queues require longer green phases to minimize wait times and prevent spillbacks.
- **Peak Hours:** Different optimization strategies for peak (rush hour) and non-peak periods, ensuring smooth traffic flow throughout the day.
- **Pedestrian Crossings**: Integration of pedestrian crossing times into the signal phases to ensure safe and timely crossings.

Algorithm Outline:

1. Data Collection and Preprocessing

- Continuously gather real-time data from sensors at each intersection.

- Store the collected data in instances of TrafficSensorData.

2. Traffic Density Calculation

- Compute traffic density using the number of vehicles detected and their average speed. This provides a metric to understand the level of congestion.

3. Signal Timing Calculation

- For each intersection, determine the optimal durations for green and red lights.

Green Time Calculation

- Start with a base duration (e.g., 30 seconds).

- Increase proportionally based on TrafficDensity and QueueLength.

- Adjust the timing based on the number of pedestrians waiting to cross.

- Red Time Calculation

- Subtract the green time from the total cycle time to determine the red light duration.

4. Dynamic Adjustment

- Continuously update signal timings based on the latest traffic data.

- Employ different algorithms for different times of the day, such as 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 the base green time and cycle time for traffic lights.

- Set scaling factors for adjusting timings based on traffic density and queue lengths.

2**. Data Collection**

- Continuously gather real-time data from traffic sensors at each intersection.

3 **Traffic Density Calculation**

- Calculate the traffic density using the formula: TrafficDensity = VehicleCount / Area.

- Adjust for real-time variations in traffic conditions.

4. **Signal Timing Calculation**

- Calculate the adjusted green time by adding increments proportional to traffic density and queue length.

- Ensure pedestrian crossing needs are accounted for in the signal timings.

5. **Dynamic Adjustment:**

- Continuously monitor and adjust signal timings based on the latest traffic data.

- Implement distinct algorithms for peak and off-peak hours to optimize traffic flow.

Implementation

Objective Develop a Java-based application that interfaces with traffic sensors and manages traffic signals at selected intersections. The system should adjust signal timings in real-time, responding to changes in traffic patterns.

Implementation Steps

1. **Define Data Structures**

   - Create Java classes to represent traffic sensor data and intersections, ensuring a structured and organized approach to data management.

2. **Simulate Data Collection**

   - Implement methods to simulate real-time data collection from traffic sensors, allowing for testing and refinement of algorithms.

3. **Optimize Signal Timings**

   - Develop algorithms to analyze the collected data and calculate optimal signal timings, ensuring efficient traffic management.

4. **Real-Time Adjustment**

   - Implement a continuous loop that adjusts signal timings based on real-time data, providing dynamic and responsive traffic control.

JAVA CODE FOR SMART TRAFFIC SIGNAL OPTIMIZATION:

```
package TrafficSignalOptimization;

import java.util.List;
```

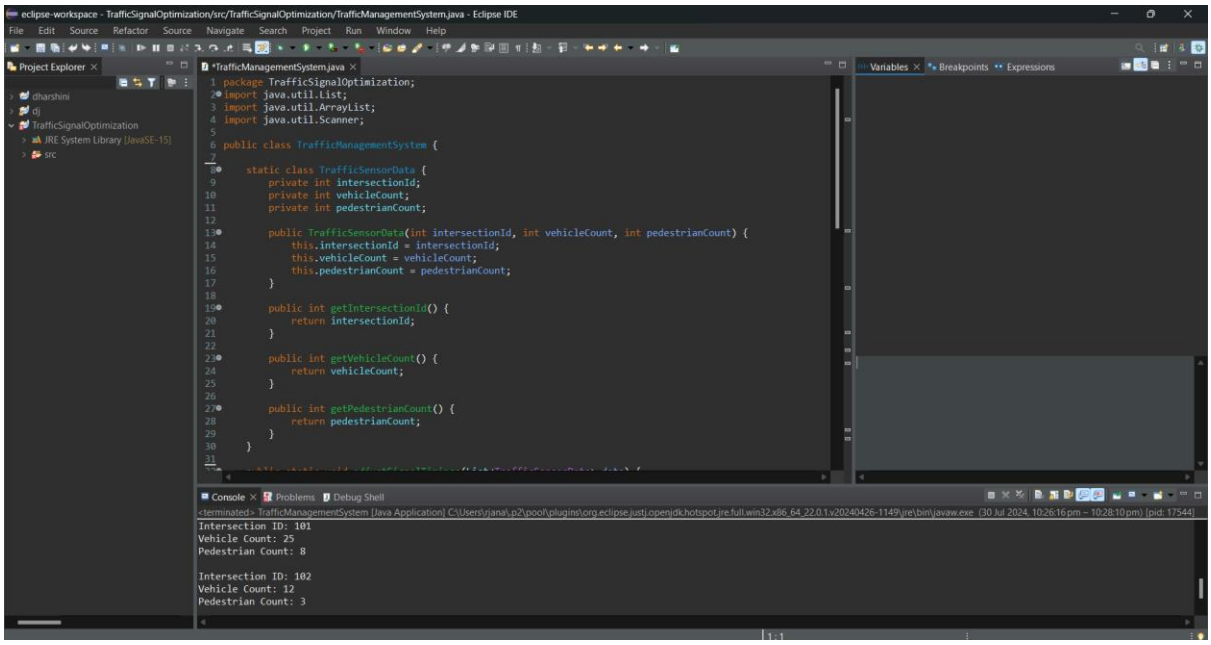
```
import java.util.ArrayList;
import java.util.Scanner;

public class TrafficManagementSystem {

    // Inner class to model traffic sensor data
    static class TrafficSensorData {
        private int intersectionId;
        private int vehicleCount;
        private int pedestrianCount;

        public TrafficSensorData(int intersectionId, int vehicleCount, int
pedestrianCount) {
            this.intersectionId = intersectionId;
            this.vehicleCount = vehicleCount;
            this.pedestrianCount = pedestrianCount;
        }

        public int getIntersectionId() {
            return intersectionId;
        }

        public int getVehicleCount() {
            return vehicleCount;
        }

        public int getPedestrianCount() {
```

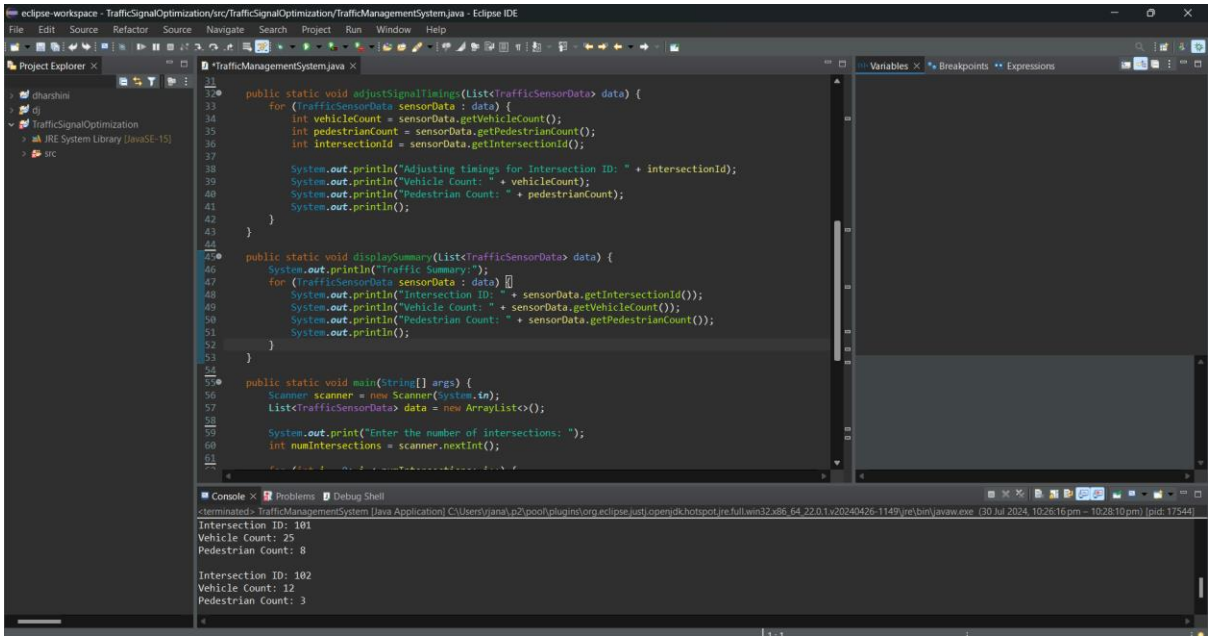
```
            return pedestrianCount;
        }
    }

    // Method to adjust signal timings based on sensor data
    public static void adjustSignalTimings(List<TrafficSensorData> data) {
        for (TrafficSensorData sensorData : data) {
            int vehicleCount = sensorData.getVehicleCount();
            int pedestrianCount = sensorData.getPedestrianCount();
            int intersectionId = sensorData.getIntersectionId();

            System.out.println("Adjusting timings for Intersection ID: " +
intersectionId);
            System.out.println("Vehicle Count: " + vehicleCount);
            System.out.println("Pedestrian Count: " + pedestrianCount);
            System.out.println();
        }
    }

    // Method to display a summary of traffic data
    public static void displaySummary(List<TrafficSensorData> data) {
        System.out.println("Traffic Summary:");
        for (TrafficSensorData sensorData : data) {
            System.out.println("Intersection ID: " + sensorData.getIntersectionId());
            System.out.println("Vehicle Count: " + sensorData.getVehicleCount());
            System.out.println("Pedestrian Count: " +
sensorData.getPedestrianCount());
```

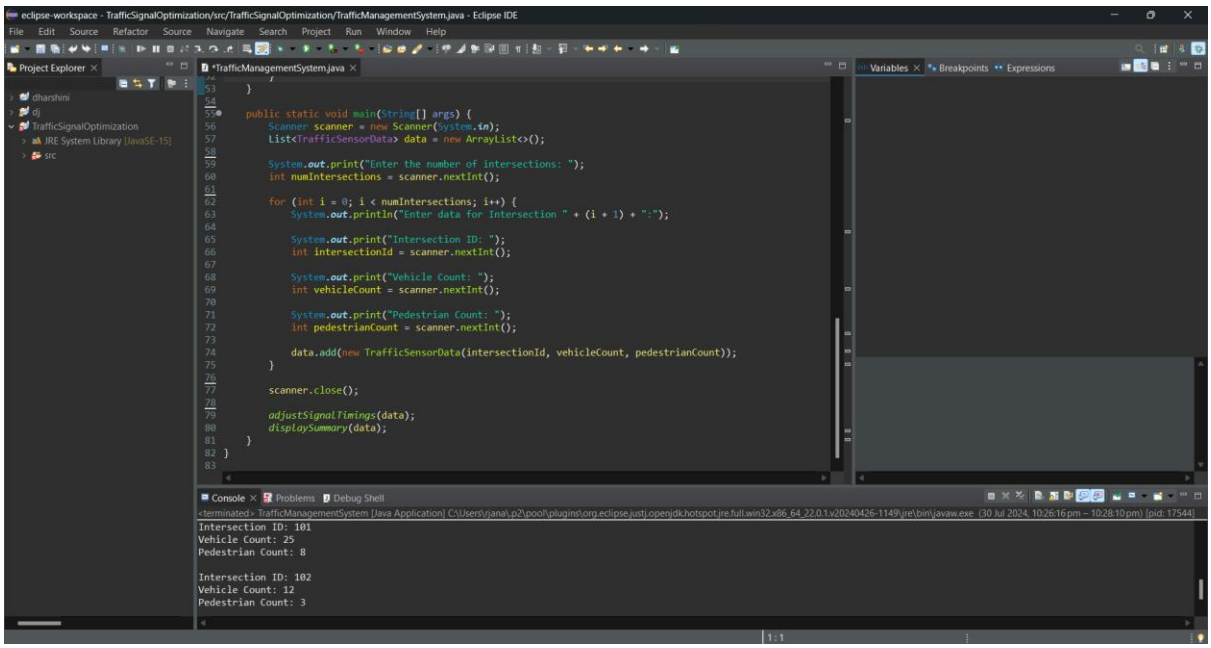
```
        System.out.println();
    }
}

// Main method to run the application
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    List<TrafficSensorData> data = new ArrayList<>();

    // Get number of intersections from the user
    System.out.print("Enter the number of intersections: ");
    int numIntersections = scanner.nextInt();

    // Input data for each intersection
    for (int i = 0; i < numIntersections; i++) {
        System.out.println("Enter data for Intersection " + (i + 1) + ":");

        System.out.print("Intersection ID: ");
        int intersectionId = scanner.nextInt();

        System.out.print("Vehicle Count: ");
        int vehicleCount = scanner.nextInt();

        System.out.print("Pedestrian Count: ");
        int pedestrianCount = scanner.nextInt();
```

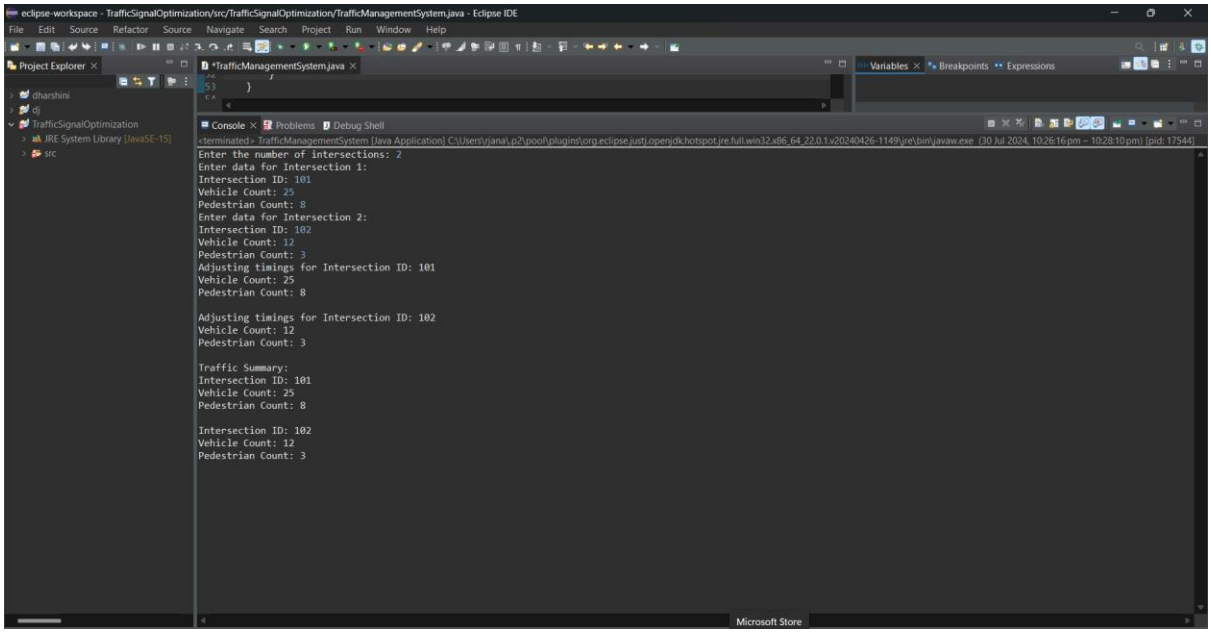
```
            data.add(new TrafficSensorData(intersectionId, vehicleCount,
pedestrianCount));
        }

        // Close the scanner
        scanner.close();

        // Adjust signal timings and display the summary
        adjustSignalTimings(data);
        displaySummary(data);
    }
}
```

EXECUTION OF THE CODE:




eclipse-workspace - TrafficSignalOptimization/src/TrafficSignalOptimization/TrafficManagementSystem.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Project Explorer
dharshini
dj
TrafficSignalOptimization
JRE System Library [JavaSE-15]
src
*TrafficManagementSystem.java
Variables
Breakpoints
Expressions
Console
Problems
Debug Shell
<terminated> TrafficManagementSystem [Java Application] C:\Users\rjana\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (30 Jul 2024, 10:26:16 pm – 10:28:10 pm) [pid: 17544]
Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8
Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3


eclipse-workspace - TrafficSignalOptimization/src/TrafficSignalOptimization/TrafficManagementSystem.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Project Explorer
dharshini
dj
TrafficSignalOptimization
JRE System Library [JavaSE-15]
src
*TrafficManagementSystem.java
Variables
Breakpoints
Expressions
Console
Problems
Debug Shell
<terminated> TrafficManagementSystem [Java Application] C:\Users\rjana\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (30 Jul 2024, 10:26:16 pm – 10:28:10 pm) [pid: 17544]
Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8
Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3


eclipse-workspace - TrafficSignalOptimization/src/TrafficSignalOptimization/TrafficManagementSystem.java - Eclipse IDE
File Edit Source Refactor Source Navigate Search Project Run Window Help
Project Explorer
dharshini
dj
TrafficSignalOptimization
JRE System Library [JavaSE-15]
src
*TrafficManagementSystem.java
}
Variables
Breakpoints
Expressions
Console
Problems
Debug Shell
<terminated> TrafficManagementSystem [Java Application] C:\Users\rjana\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (30 Jul 2024, 10:26:16 pm – 10:28:10 pm) [pid: 17544]
Enter the number of intersections: 2
Enter data for Intersection 1:
Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8
Enter data for Intersection 2:
Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3
Adjusting timings for Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8
Adjusting timings for Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3
Traffic Summary:
Intersection ID: 101
Vehicle Count: 25
Pedestrian Count: 8
Intersection ID: 102
Vehicle Count: 12
Pedestrian Count: 3
Microsoft Store