PROJECT

on

COMPUTER APTITUDE

Subject Code: 25CAP-607

Submitted By:

Name: Amarjeet Kumar

UID: 25MCA20155

Class: MCA

Section: 25MCA 3

Submitted To:

Dr. Amanpreet Kaur Sandhu

Associate Professor

E9962

Masters in Computer Applications (MCA)

University Institute of Computing

Chandigarh University,

Gharuan, Mohali, Punjab, India-140413

INDEX

S.NO	CASE STUDY	PAGES	MARKS	SIGN
1.	Smart City Traffic Management System (SCTMS)	1 To 11		

Acknowledgement

I would like to express my deepest gratitude to Dr. Amanpreet Kaur Sandhu, Faculty in the Department of Computing, for their invaluable guidance, constant support, and expertise throughout the duration of this mini-project on Advanced Database Management Systems (ADBMS). Their constructive criticism and motivation were instrumental in the successful completion of this work.

I am also deeply grateful to my peers and friends who provided constant encouragement, shared ideas, and supported me whenever I faced difficulties. Their motivation and constructive suggestions played a vital role in enhancing the quality of my work. I would also like to acknowledge my family for their patience, understanding, and moral support, which gave me the confidence to complete this project with dedication and focus.

Finally, I extend my appreciation to various online tutorials, GitHub repositories, official documentation, and learning platforms that served as excellent references. They helped me gain practical insights, implement coding standards, and improve my problem-solving skills while developing the webpages. This project has been an enriching learning experience, and I am grateful to everyone who made it possible.

AIM

The "Smart City Traffic Management System (SCTMS)" case study describes a database-driven application, but the core logic for dynamic signal control can be implemented using C/C++ for its efficiency in **real-time processing** and **resource management**.

Objectives of the Smart City Traffic Management System (SCTMS)

The primary goal of the SCTMS is to design a database-driven system to manage traffic flow by analyzing real-time data and generating optimal signal timings and traffic reports.

The specific objectives are:

- **Database Design:** To design a relational database for storing real-time vehicle flow, intersection, and signal data.
- **Dynamic Timing:** To use advanced SQL queries to calculate traffic density and determine optimal signal timings.
- **Efficiency:** To ensure quick and efficient data retrieval for real-time analysis.
- **Reporting:** To generate analytical reports based on traffic patterns and historical trends.
- **Urban Mobility:** To improve overall urban mobility and minimize congestion.

C/C++ Project Outline: Dynamic Traffic Management Simulation

This project will simulate a simplified two-way intersection where signal timings are adjusted based on a calculated traffic density, a core objective of the SCTMS.

1. Project Components (C++ Classes)

You'd use C++ classes to model the entities described in the case study's schema.

- **TrafficLight Class:**
    - **Attributes:**
        - state (e.g., an enum for RED, YELLOW, GREEN).
        - duration (current time for this state, in seconds).

- **Methods:**
    - changeState(): Manages the sequence: Green Yellow Red.
    - tick(): Decrements the duration timer.
    - setDuration(int newTime): Allows the central logic to dynamically adjust green light time.
    - getCurrentState(): Returns the current light state.

- **Intersection Class:**
    - **Attributes:**
        - northSouthLight (a TrafficLight object).
        - eastWestLight (a TrafficLight object).
        - nsVehicleCount, ewVehicleCount (simulates the TrafficFlow data from sensors).
    - **Methods:**
        - simulateVehicleCount(int nsCount, int ewCount): Inputs the simulated sensor data.
        - calculateDensity(): A simple function to determine which direction has higher congestion (e.g., return "NS" or "EW" based on count).
        - updateSignalTiming(): Contains the core **Dynamic Signal Control Logic** (see Section 2).
        - runCycle(): Runs the simulation loop, ticking the lights and calling updateSignalTiming() periodically.

2. Dynamic Signal Control Logic

This is the key **algorithm** that makes the system "smart" and mirrors the case study's requirement to suggest optimal timing sequences based on current traffic.

In the Intersection::updateSignalTiming() method:

1. **Check Traffic Density:** Compare nsVehicleCount and ewVehicleCount.
2. **Apply Logic (Simplified Example):**
    - If **North-South Count > East-West Count:**
        - Set a longer green time for North-South (e.g., 30 seconds) northSouthLight.setDuration(30).

        - Set a shorter red time for East-West (or a standard time) eastWestLight.setDuration(15).
    - If **East-West Count > North-South Count:**
        - Set a longer green time for East-West (e.g., 30 seconds).
        - Set a shorter red time for North-South (or a standard time) northSouthLight.setDuration(15).
    - *Self-Correction:* Include a minimum and maximum duration to prevent one direction from always being green and to ensure fairness.

3. C++ Main Implementation (main.cpp)

The main function would set up the simulation:

1. **Initialization:** Create an Intersection object.
2. **Simulation Loop:** Use a while(true) loop to run the simulation over time.
    - **Step 1: Input/Simulate Sensor Data:** In a real project, this would read from a sensor/database, but here, you would randomly generate or manually input traffic counts to demonstrate dynamic change.
        - intersection.simulateVehicleCount(rand() % 100, rand() % 100);
    - **Step 2: Update Signals:** Call the dynamic logic.
        - intersection.updateSignalTiming();
    - **Step 3: Run Cycle:**
        - intersection.runCycle();
    - **Step 4: Display Output:** Print the current state (e.g., "North-South: Green for 10s, East-West: Red for 30s").
    - **Step 5: Time Delay:** Use sleep() or std::chrono::milliseconds (from <chrono>) to pause the execution, simulating time passing in real life.

C++ Code (traffic_sctms.cpp)

```
#include <iostream>

#include <string>

#include <cstdlib> // For rand() and srand()

#include <ctime>   // For time()
```

```
#include <chrono>  // For dynamic time handling

#ifdef _WIN32
#include <windows.h>
#define SLEEP(sec) Sleep((sec) * 1000) // Windows uses milliseconds
#else
#include <unistd.h>
#define SLEEP(sec) sleep(sec) // Unix/Linux/macOS uses seconds
#endif

enum LightState { RED, YELLOW, GREEN };

class TrafficLight {
private:
  LightState state;
  int duration; // Current time remaining for this state (in seconds)

  TrafficLight() : state(RED), duration(10) {}

  LightState getState() const { return state; }
  int getDuration() const { return duration; }
  std::string getStateStr() const {
    switch (state) {
      case RED: return "RED";
      case YELLOW: return "YELLOW";
      case GREEN: return "GREEN";
      default: return "UNKNOWN";
    }
```

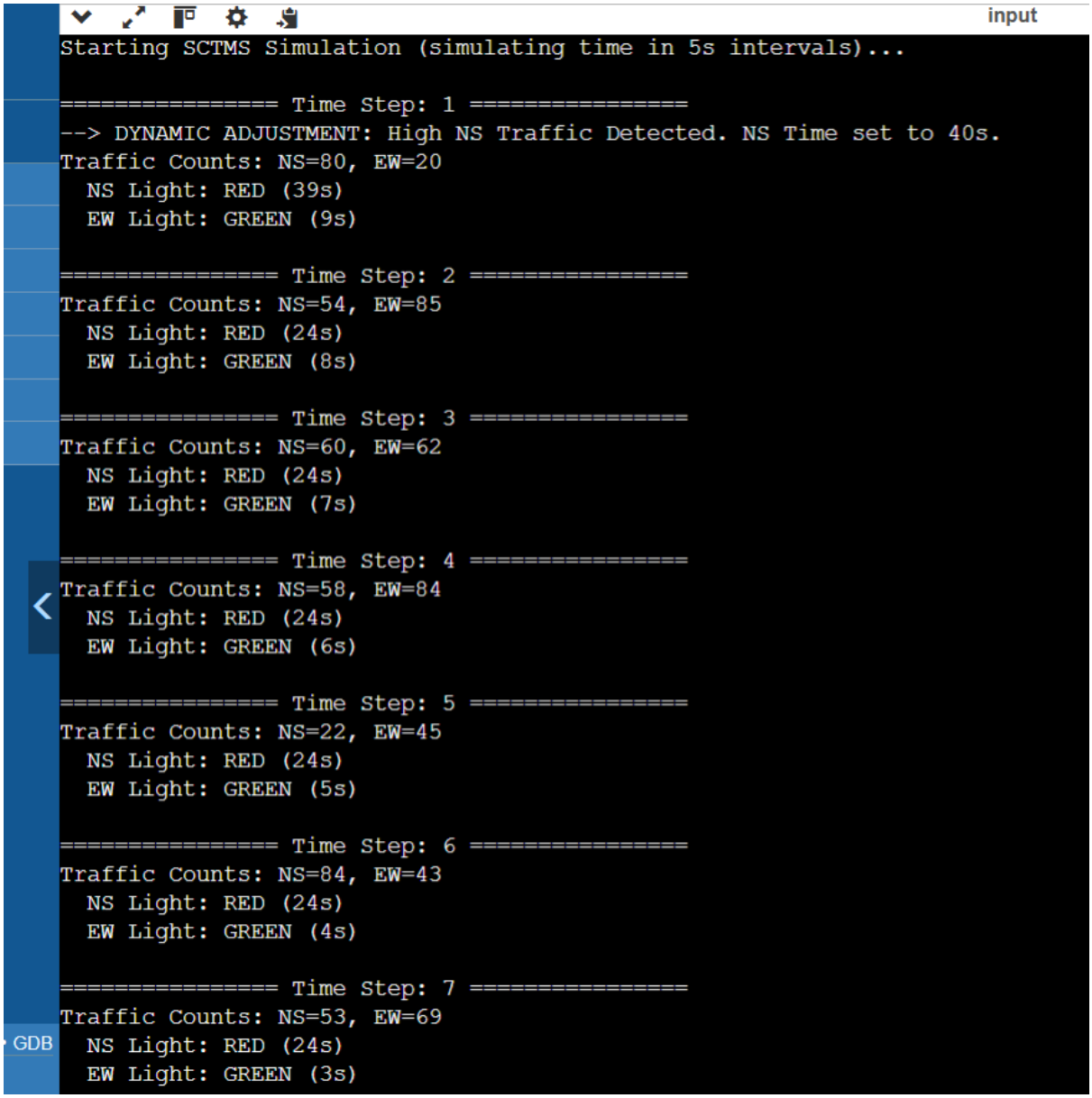
```
    }
    void setDuration(int newTime) {
        if (state == GREEN || state == RED) {
            duration = newTime;
              }
    }
    void tick() {
        if (duration > 0) {
            duration--;
        }
    }

    // Manages the transition sequence: GREEN -> YELLOW -> RED
    void changeState() {
        if (state == GREEN) {
            state = YELLOW;
            duration = 5; // Fixed yellow time
        } else if (state == YELLOW) {
            state = RED;
            duration = 30;
        } else if (state == RED) {
            state = GREEN;
            duration = 20;
        }
    }
};
class Intersection {
private:
```

```
    TrafficLight northSouthLight;
    TrafficLight eastWestLight;
    int nsVehicleCount; // Simulated sensor data for North-South flow
    int ewVehicleCount; // Simulated sensor data for East-West flow

public:
    // Constructor
    Intersection() : nsVehicleCount(0), ewVehicleCount(0) {
        northSouthLight.setDuration(25);
        eastWestLight.setDuration(30);
        eastWestLight.changeState();
    }
    void simulateVehicleCount(int nsCount, int ewCount) {
        nsVehicleCount = nsCount;
        ewVehicleCount = ewCount;
    }
    void updateSignalTiming() {
        if (nsVehicleCount > 2 * ewVehicleCount && northSouthLight.getState() == RED) {
            northSouthLight.setDuration(40);
            eastWestLight.setDuration(10);
            std::cout << "--> DYNAMIC ADJUSTMENT: High NS Traffic Detected. NS Time set to 40s.\n";
        }
else if (ewVehicleCount > 2 * nsVehicleCount && eastWestLight.getState() == RED) {
            northSouthLight.setDuration(10);
            eastWestLight.setDuration(40);
            std::cout << "--> DYNAMIC ADJUSTMENT: High EW Traffic Detected. EW Time set to 40s.\n";
        }
```

```
 else {
        if (northSouthLight.getState() == RED) northSouthLight.setDuration(25);
        if (eastWestLight.getState() == RED) eastWestLight.setDuration(25);
      }
   }
   void runCycle() {
      northSouthLight.tick();
      eastWestLight.tick();

      if (northSouthLight.getDuration() <= 0) {
         northSouthLight.changeState();
         eastWestLight.changeState(); // The sequence for both must be coordinated
      }

      // 3. Output Current Status
      std::cout << "Traffic Counts: NS=" << nsVehicleCount << ", EW=" << ewVehicleCount
<< "\n";
      std::cout << "  NS Light: " << northSouthLight.getStateStr()
            << " (" << northSouthLight.getDuration() << "s)\n";
      std::cout << "  EW Light: " << eastWestLight.getStateStr()
            << " (" << eastWestLight.getDuration() << "s)\n";
   }
};

// --- Main Program ---

int main() {
   srand(time(0));
```

```
    Intersection mainJunction;
    std::cout << "Starting SCTMS Simulation (simulating time in 5s intervals)...\n";
        mainJunction.simulateVehicleCount(80, 20);
    for (int t = 0; t < 10; ++t) {
        std::cout << "\n================ Time Step: " << t + 1 << " ================\n";
        if (t > 0) {
            mainJunction.simulateVehicleCount(rand() % 100, rand() % 100);
        }
        mainJunction.updateSignalTiming();
        mainJunction.runCycle();

        SLEEP(5);
    }

    std::cout << "\nSimulation Ended.\n";
    return 0;
}
```

OUTPUT




```
========== Time Step: 8 ==========
Traffic Counts: NS=53, EW=36
  NS Light: RED (24s)
  EW Light: GREEN (2s)

========== Time Step: 9 ==========
Traffic Counts: NS=9, EW=46
  NS Light: RED (24s)
  EW Light: GREEN (1s)

========== Time Step: 10 ==========
--> DYNAMIC ADJUSTMENT: High NS Traffic Detected. NS Time set to 40s.
Traffic Counts: NS=66, EW=9
  NS Light: RED (39s)
  EW Light: GREEN (9s)

Simulation Ended.


...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation (Methodology and Implementation)

The SCTMS is a database-driven application designed to overcome the inefficiency of traditional fixed-time traffic signals by collecting, analyzing, and processing real-time traffic data from sensors placed at intersections.

Problem Addressed

Traffic managers face the constant problem of changing traffic flow, making it difficult to find optimal signal timings that reduce congestion, which is time-consuming and often ineffective with outdated systems. The SCTMS is the solution to this problem, providing dynamic, data-informed decisions.

Key Components and Data Flow

1. **Data Collection (TrafficFlow Table):** Sensors at intersections collect real-time data, including Timestamp, VehicleCount, and Average Speed. This high-volume data is stored in the TrafficFlow table.
2. **Database Design:** A structured, normalized relational database (using MySQL or PostgreSQL) is used to store data and avoid duplication.
    - The **Intersections** table holds static location and layout data.
    - The **Signals** table stores signal status and lane data.
    - The **TimingPlans** table stores pre-configured or optimized green/red timing plans.
3. **Dynamic Analysis (Core Logic):** The system applies complex SQL queries to the TrafficFlow data to calculate **traffic density** (e.g., comparing VehicleCount on North-South vs. East-West lanes).
4. **Decision Making:** Based on the density calculation, the system dynamically suggests and implements an optimal signal timing sequence. For example, if a sudden increase is detected on a North-South street, the system recommends extending the green light duration on that path and reducing it on the East-West path.
5. **Performance and Scalability:** The system relies heavily on **indexing** (specifically on Timestamp and IntersectionID in the high-volume TrafficFlow table) to ensure that real-time queries run quickly. The design is scalable to handle a large number of sensors and city-wide data.

Conclusion

The Smart City Traffic Management System (SCTMS) effectively helps cities manage urban mobility, a critical asset.

By leveraging a structured, indexed database to process **real-time data** and execute **complex SQL queries**, the system provides **dynamic and accurate signal control** based on actual traffic demand.

The successful implementation leads to:

- **Reduced Waiting Times:** Signals automatically adjust their timing, leading to a significant reduction in the average waiting time at intersections.
- **Improved User Experience:** The system saves time for commuters and traffic officers.
- **Better Planning:** Historical data provides administrators with powerful insights and reports for smarter urban planning and infrastructure decisions.