**A Course Based Project Report On**

# TRAFFIC SIGNAL OPERATOR

**Submitted in partial fulfillment of requirement for the completion of the**

**Data Structures Laboratory**

**I B.Tech Computer Science and Engineering of**

## VNR VJIET

**By**

|  |  |
| --- | --- |
| 1. **RITHISH** | **23071A05D7** |
| **G. SRI VIDYA** | **23071A05F8** |
| **HANVITH SAI ALLA** | **23071A05G2** |
| **K. ADVAITH** | **23071A05G5** |
| **K. KARTHIKEYA** | **23071A05G6** |

**2023-2024**



**VALLURIPALLI NAGESWARA RAO**

**VIGNANA JYOTHI INSTITUTE OF ENGINEERING &TECHNOLOGY (AUTONOMOUS INSTITUTE)**

**NAAC ACCREDITED WITH ‘A++’ GRADE**

**NBA Accreditation for B. Tech Programs**

**Vignana Jyothi Nagar, Bachupally, Nizampet (S.O), Hyderabad 500090 Phone no: 040-23042758/59/60, Fax: 040-23042761**

[**Email:postbox@vnrvjiet.ac.in**](mailto:postbox@vnrvjiet.ac.in) **Website:** [**www.vnrvjiet.ac.in**](http://www.vnrvjiet.ac.in/)

**A Project Report On**

# TRAFFIC SIGNAL OPERATOR

**Submitted in partial fulfillment of requirement for the completion of the**

**Data Structures Laboratory**

**B.Tech Computer Science and Engineering of**

## VNRVJIET 2023-2024

**Under the Guidance of**

## DR. D N VASUNDHARA

**Assistant Professor Department of CSE**





## VNR VIGNANA JYOTHI INSTITUTE OF ENGINEERING & TECHNOLOGY

**(AUTONOMUS INSTITUTE)**

**NAAC ACCREDITED WITH ‘A++’ GRADE**

## CERTIFICATE

This is to certify that the project entitled “TRAFFIC SIGNAL OPERATOR” submitted in partial fulfillment for the course of Data Structures laboratory being offered for the award of Batch (CSE-C) by VNRVJIET is a result of the bonafide work carried out by **23071A05D7**, **23071A05F8, 23071A05G2, 23071A05G5** and **23071A05G6** during the year **2023-2024**. This has not been submitted for any other certificate or course.

.

|  |  |
| --- | --- |
| **Signature of Faculty** | **Signature of Head of the Department** |

# ACKNOWLEDGEMENT

An endeavor over a long period can be successful only with the advice and support of many well-wishers. We take this opportunity to express our gratitude and appreciation to all of them.

We wish to express our profound gratitude to our honorable **Principal Dr. C.D.Naidu** and **HOD Dr.S.Nagini**, **CSE Department**, **VNR Vignana Jyothi Institute of Engineering and Technology** for their constant and dedicated support towards our career moulding and development.

With great pleasure we express our gratitude to the internal guide **DR. DN VASUNDHARA, Assistant Professor, CSE department** for his timely help, constant guidance, cooperation, support and encouragement throughout this project as it has urged us to explore many new things.

Finally, we wish to express my deep sense of gratitude and sincere thanks to our parents, friends and all our well-wishers who have technically and non-technically contributed to the successful completion of this course-based project.

# DECLARATION

We hereby declare that this Project Report titled “**TRAFFIC SIGNAL OPERATOR**” submitted by us of Computer Science & Engineering in **VNR Vignana Jyothi Institute of Engineering and Technology**, is a bonafide work undertaken by us and it is not submitted for any other certificate /Course or published any time before.

Name & Signature of the Students:

|  |  |
| --- | --- |
| 1. **RITHISH** | **23071A05D7** |
| **G. SRI VIDYA** | **23071A05F8** |
| **HANVITH SAI ALLA** | **23071A05G2** |
| **K. ADVAITH** | **23071A05G5** |
| **K. KARTHIKEYA** | **23071A05G6** |

**Date:**

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S No | Topic | Pg no. |
| 1 | Abstract | 07 |
| 2 | Introduction | 08 |
| 3 | Methodology | 09 |
| 4 | Objectives | 10 |
| 5 | Flow of execution | 11 |
| 5 | Code | 12-21 |
| 6 | Output | 22-25 |
| 7 | Conclusion | 26 |
| 8 | Future Scope | 27 |
| 9 | References | 28 |

**ABSTRACT**

Efficient traffic management is paramount for ensuring smooth urban mobility, reducing congestion, and facilitating the timely passage of emergency vehicles. As cities continue to grow, the complexity of managing traffic at intersections increases, necessitating advanced algorithms that can handle the dynamic nature of urban traffic. This paper presents a comprehensive traffic signal algorithm implemented using data structures like queues and linked lists, specifically designed to address these challenges at a four-way intersection.

The algorithm's primary objective is to manage the vehicle flow efficiently, giving priority to emergency vehicles while maintaining regular traffic operations. This dual focus is crucial for balancing the urgent needs of emergency responders with the everyday requirements of city commuters. The system is user-interactive, allowing users to create traffic data by enqueuing vehicles into respective lanes, display the current state of traffic lanes, and manage traffic flow by processing vehicles based on their priority.

To achieve this, the algorithm utilizes a queue structure, which facilitates the organized management of vehicles by ensuring that each vehicle is processed in the order of its arrival, except when an emergency vehicle is detected. Emergency vehicles are given immediate priority, interrupting the normal queue to ensure they can pass through the intersection without delay. This prioritization is crucial for reducing response times in emergency situations.

In addition to managing real-time traffic flow, the algorithm maintains a log of all processed vehicles using a linked list structure. This log serves multiple purposes: it provides a record for monitoring and auditing traffic patterns, helps in analyzing the efficiency of the traffic management system, and offers data for future improvements. By keeping an accurate record of vehicle movements, urban planners and traffic engineers can gain insights into traffic trends and identify potential bottlenecks.

The program simulates various traffic conditions, providing a robust framework for real-world traffic signal management systems. It allows for the dynamic adjustment of signal timings based on real-time traffic data, ensuring that the intersection operates at optimal efficiency. This capability is particularly valuable in urban environments where traffic conditions can change rapidly due to factors such as accidents, road works, or sudden increases in traffic volume.

Through the implementation of this algorithm, urban planners and traffic engineers can explore and test different strategies to optimize intersection management. By prioritizing emergency vehicles and balancing the flow of regular traffic, the system aims to improve overall traffic efficiency, reduce congestion, and enhance the safety and mobility of all road users.

In conclusion, the traffic signal algorithm presented in this paper offers a sophisticated approach to managing urban intersections. By leveraging data structures like queues and linked lists, the system effectively handles the complexities of real-time traffic management. This algorithm not only prioritizes emergency vehicles but also ensures a smooth flow of regular traffic, providing a valuable tool for urban planners and traffic engineers seeking to enhance traffic efficiency and safety in growing cities.

# INTRODUCTION

Traffic congestion is a persistent issue in urban areas, often leading to significant delays, increased fuel consumption, and heightened emissions. Effective traffic management at intersections is crucial to alleviating these problems, as intersections are often the critical points where traffic flow bottlenecks occur. Efficiently managing traffic flow not only improves transportation systems but also ensures the swift passage of emergency vehicles, thereby enhancing public safety and response times. This paper presents a program that implements a sophisticated traffic signal algorithm using fundamental data structures such as queues and linked lists to manage the flow of vehicles at a four-way intersection. This approach not only addresses congestion but also contributes to smoother and more predictable traffic patterns.

The primary objective of the algorithm is to prioritize emergency vehicles, ensuring their quick passage through the intersection while simultaneously maintaining the regular flow of traffic. This balance is essential to minimize delays and optimize the overall efficiency of the traffic system. The system is designed to allow for the creation, display, and management of vehicle queues in different directions: North, East, South, and West. Each direction has its own queue to manage the incoming and outgoing traffic, ensuring an organized and systematic approach to vehicle movement. By maintaining distinct queues, the system can effectively handle varying traffic densities and reduce the waiting time for vehicles.

In addition to managing the current traffic flow, the algorithm maintains a detailed log of all processed vehicles using a linked list structure. This log serves multiple important functions: it provides a historical record for monitoring and auditing purposes, helps analyze traffic patterns and efficiency, and offers valuable data for future improvements and optimizations.

By implementing and testing this traffic signal algorithm, urban planners and traffic engineers can explore and develop strategies to optimize intersection management. The algorithm's ability to prioritize emergency vehicles while maintaining a smooth flow of regular traffic offers a significant advantage in reducing congestion and enhancing public safety. The insights gained from simulating traffic management through this algorithm can contribute to the development of more efficient and responsive traffic control systems in urban areas.

# METHODOLOGY

The methodology for developing the traffic signal algorithm for managing a four-way intersection focuses on ensuring efficient traffic flow and prioritizing emergency vehicles. This systematic approach involves several key stages: initialization and input handling, vehicle queue management, traffic flow control, and vehicle logging. Each phase leverages fundamental data structures like queues and linked lists to achieve effective traffic management.

1. **Initialization and Input Handling**: The initial phase involves setting up the system to handle user inputs and initialize the necessary data structures. The system prompts the user to input the number of vehicles and their respective directions (North, East, South, and West). These inputs are used to enqueue vehicles into their respective lanes, with emergency vehicles given a special priority flag.
2. **Vehicle Queue Management**: In this phase, the algorithm manages the vehicle queues for each direction. The queue structure ensures that vehicles are processed in the order they arrive, maintaining an organized flow. For emergency vehicles, the system interrupts the regular queue, allowing these vehicles to move to the front. This is crucial for minimizing the response time of emergency services. The queues are implemented using linked lists to facilitate dynamic and efficient insertion and deletion of vehicles.
3. **Traffic Flow Control**: The core of the algorithm involves controlling the traffic flow based on the current state of the queues and the presence of emergency vehicles. The system simulates traffic signals, allowing vehicles to proceed in a controlled manner. When an emergency vehicle is detected, the algorithm temporarily overrides normal traffic rules to give it immediate passage. This phase ensures a balance between prioritizing emergency vehicles and maintaining regular traffic flow.
4. **Vehicle Logging**: This phase involves maintaining a detailed log of all processed vehicles. The linked list structure is used to record each vehicle's entry and exit from the intersection. This log serves multiple purposes: it provides a record for monitoring and auditing traffic patterns, helps analyze the efficiency of the traffic management system, and offers data for future improvements. The log ensures transparency and accountability in the traffic management process.

Overall, the methodology emphasizes a user-centric design, efficient implementation of data structures, and iterative refinement to ensure the delivery of a high-quality traffic management system. By prioritizing emergency vehicles and maintaining regular traffic flow, the system aims to enhance the efficiency, safety, and mobility of urban intersections.

# OBJECTIVES

The primary objective of the traffic signal algorithm implemented in this code is to efficiently manage the flow of vehicles at a four-way intersection, with a special emphasis on prioritizing emergency vehicles. This system aims to achieve the following specific objectives:

1. **Efficient Traffic Flow Management**:
   * To organize and manage the movement of vehicles at an intersection in a systematic manner.
   * To ensure that vehicles are processed in the order they arrive, maintaining an orderly flow.
2. **Prioritization of Emergency Vehicles**:
   * To identify and prioritize emergency vehicles, ensuring they receive immediate passage through the intersection.
   * To minimize the response time for emergency services by interrupting the regular traffic queue when necessary.
3. **User Interaction and Input Handling**:
   * To allow users to input traffic data, including the number of vehicles and their respective directions.
   * To dynamically enqueue vehicles into their respective lanes based on user inputs.
4. **Simulation of Traffic Conditions**:
   * To simulate real-world traffic conditions and traffic signal operations at a four-way intersection.
   * To dynamically adjust the signal timings and vehicle movements based on the current state of the traffic queues.
5. **Vehicle Logging and Monitoring**:
   * To maintain a detailed log of all vehicles processed through the intersection for monitoring and auditing purposes.
   * To provide data for analyzing traffic patterns and improving future traffic management strategies.
6. **Balancing Regular Traffic and Emergency Situations**:
   * To balance the flow of regular traffic while ensuring that emergency vehicles are not delayed.
   * To enhance the overall safety and efficiency of traffic management at urban intersections.

By achieving these objectives, the algorithm aims to provide a robust framework for real-world traffic signal management systems, contributing to smoother urban mobility, reduced congestion, and enhanced public safety.

**FLOW OF EXECUTION**

**Traffic Signal Algorithm Program Flow**

**1. Start Program**

**|**

**V**

**2. Input Number of Vehicles (N) and Directions**

**|**

**V**

**3. Initialize Traffic Queues for North, East, South, West Directions**

**|**

**V**

**4. User Interaction:**

**- Enqueue Vehicles into Respective Directional Queues**

**- Handle Emergency Vehicles with Priority**

**|**

**V**

**5. Traffic Signal Management:**

**- Process Vehicles from Queues**

**- Prioritize Emergency Vehicles for Immediate Passage**

**|**

**V**

**6. Vehicle Processing:**

**- Dequeue Vehicles in FIFO Order for Regular Traffic**

**- Log Processed Vehicles**

**|**

**V**

**7. Display Current Traffic State:**

**- Show Vehicles Waiting in Each Direction**

**- Indicate Current Traffic Signal Status**

**|**

**V**

**8. End Program**

**IMPLEMENTATION OF PROGRAM**

**Code:**

// TRAFFIC SIGNAL ALGORITHM

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// Vehicle structure to hold vehicle details

typedef struct Vehicle {

    int id;             // Vehicle ID

    int priority;       // 0 for regular, 1 for emergency

    char direction[4];  // "N", "E", "S", "W"

    struct Vehicle \*next;

} Vehicle;

// Queue structure for lane queues

typedef struct {

    Vehicle \*front;     // Pointer to the first vehicle in the queue

    Vehicle \*rear;      // Pointer to the last vehicle in the queue

} Queue;

// Log structure to log processed vehicles

typedef struct Log {

    int id;             // Vehicle ID

    int priority;       // 0 for regular, 1 for emergency

    char direction[4];  // "N", "E", "S", "W"

    struct Log \*next;   // Pointer to the next log entry

} Log;

Queue lanes[4];                  // Array of queues for four lanes: 0: North, 1: East, 2: South, 3: West

Log \*traffic\_log = NULL;         // Head of the log linked list

int emergency\_vehicle\_count = 0; // Count of emergency vehicles in the queues

// Initialize the lanes as empty queues

void create\_lanes() {

    for (int i = 0; i < 4; i++) {

        lanes[i].front = NULL;  // Initialize front of each lane as NULL

        lanes[i].rear = NULL;   // Initialize rear of each lane as NULL

    }

}

// Check if a queue is empty

int is\_empty(Queue \*queue) {

    return queue->front == NULL;  // Returns 1 if the queue is empty

}

// Get the lane index from the direction

int get\_lane\_index(char direction[]) {

    if (strcmp(direction, "N") == 0) return 0;  // North lane

    if (strcmp(direction, "E") == 0) return 1;  // East lane

    if (strcmp(direction, "S") == 0) return 2;  // South lane

    if (strcmp(direction, "W") == 0) return 3;  // West lane

    return -1;  // Invalid direction

}

// Check if a vehicle with a given ID exists in any lane

int vehicle\_exists(int id) {

    for (int i = 0; i < 4; i++) {  // Iterate over each lane

        Vehicle \*temp = lanes[i].front;

        while (temp != NULL) {

            if (temp->id == id) return 1;  // Vehicle exists

            temp = temp->next;

        }

    }

    return 0;  // Vehicle does not exist

}

// Add a vehicle to the appropriate lane

void enqueue\_vehicle(int id, int priority, char direction[]) {

    int lane\_index = get\_lane\_index(direction);  // Get the lane index- 0: North, 1: East, 2: South, 3: West

    if (lane\_index == -1) {

        printf("\033[1;30mWARNING: Invalid direction. Use N, S, E, W only.\n\033[0m");

        return;

    }

    Vehicle \*new\_vehicle = (Vehicle \*)malloc(sizeof(Vehicle));

    if (new\_vehicle == NULL) {

        printf("\033[1;30mMemory allocation failed.\n\033[0m");

        exit(EXIT\_FAILURE);

    }

    new\_vehicle->id = id;

    new\_vehicle->priority = priority;

    strcpy(new\_vehicle->direction, direction);

    new\_vehicle->next = NULL;

    if (is\_empty(&lanes[lane\_index])) {

        lanes[lane\_index].front = new\_vehicle;

        lanes[lane\_index].rear = new\_vehicle;

    } else {

        lanes[lane\_index].rear->next = new\_vehicle;

        lanes[lane\_index].rear = new\_vehicle;

    }

    if (priority == 1) emergency\_vehicle\_count++;

}

// Create traffic data by enqueuing vehicles

void create() {

    while (1) {

        char input[10];         // Input buffer for vehicle ID

        int id, priority;       // Vehicle ID and priority

        char direction[4];      // Vehicle direction

        printf("Enter Vehicle ID (3-digit), Priority (0 or 1), and Direction (N, E, S, W) or '$' to stop: ");

        scanf("%s", input);

        if (strcmp(input, "$") == 0) break;

        id = atoi(input);       // Convert input to integer (ASCII to integer = atoi)

        scanf("%d %s", &priority, direction);

        for (int i = 0; direction[i] != '\0'; i++)

            direction[i] = toupper(direction[i]);  // Convert direction to uppercase

        if (id < 100 || id > 999) {

            printf("\033[1;30mWARNING: Vehicle ID must be a 3-digit number.\n\033[0m");

            continue;

        }

        if (priority != 0 && priority != 1) {

            printf("\033[1;30mWARNING: Priority must be 0 or 1.\n\033[0m");

            continue;

        }

        if (vehicle\_exists(id)) {

            printf("\033[1;30mWARNING: Vehicle with ID %d already exists. Enter a different ID.\n\033[0m", id);

            continue;

        }

        enqueue\_vehicle(id, priority, direction);  // Enqueue the vehicle

    }

}

// Display all vehicles in all lanes

void display() {

    const char \*directions[4] = {"North", "East", "South", "West"};

    for (int i = 0; i < 4; i++) {

        printf("%s Lane:\n", directions[i]);

        if (is\_empty(&lanes[i])) {

            printf("\033[1;30mNO VEHICLES IN THIS LANE.\n\033[0m");

        } else {

            Vehicle \*temp = lanes[i].front;

            while (temp != NULL) {

                printf("  Vehicle ID: %d, Priority: %s\n", temp->id, temp->priority ? "Emergency" : "Regular");

                temp = temp->next;

            }

        }

    }

}

// Count the number of vehicles in a given lane

int count\_vehicles\_in\_lane(int lane\_index) {

    int count = 0;              // Vehicle count of that particular lane

    Vehicle \*temp = lanes[lane\_index].front;

    while (temp != NULL) {

        count++;

        temp = temp->next;

    }

    return count;

}

// Add a vehicle to the log of processed vehicles

void add\_to\_log(int id, int priority, char direction[]) {

    Log \*new\_log = (Log \*)malloc(sizeof(Log));

    if (new\_log == NULL) {

        printf("\033[1;30mMemory allocation failed.\n\033[0m");

        exit(EXIT\_FAILURE);

    }

    new\_log->id = id;

    new\_log->priority = priority;

    strcpy(new\_log->direction, direction);

    new\_log->next = NULL;

    if (traffic\_log == NULL) {

        traffic\_log = new\_log;

    } else {

        Log \*temp = traffic\_log;

        while (temp->next != NULL)

            temp = temp->next;

        temp->next = new\_log;

    }

}

// Free the memory allocated for vehicles in a lane

void free\_vehicles(Queue \*lane) {

    Vehicle \*temp;

    while (lane->front != NULL) {

        temp = lane->front;

        lane->front = lane->front->next;

        free(temp);

    }

}

// Process and dequeue emergency vehicles first

void process\_emergency\_vehicles(const char \*directions[]) {

    while (emergency\_vehicle\_count > 0) {

        int emergency\_count[4] = {0};  // Array to store emergency vehicle counts for each lane

        for (int i = 0; i < 4; i++) {

            Vehicle \*temp = lanes[i].front;

            while (temp != NULL) {

                if (temp->priority == 1)

                    emergency\_count[i]++;

                temp = temp->next;

            }

        }

        int total\_emergency\_vehicles = 0;  // Total count of emergency vehicles

        for (int i = 0; i < 4; i++)

            total\_emergency\_vehicles += emergency\_count[i];

        if (total\_emergency\_vehicles == 0) break;

        int lane\_order[4] = {0, 1, 2, 3};  // Lane order array for prioritizing lanes using Bubble Sort

        for (int i = 0; i < 4 - 1; i++) {

            for (int j = 0; j < 4 - i - 1; j++) {

                if (emergency\_count[lane\_order[j]] == emergency\_count[lane\_order[j + 1]]) {

                    if (lane\_order[j] > lane\_order[j + 1]) {

                        int temp = lane\_order[j];

                        lane\_order[j] = lane\_order[j + 1];

                        lane\_order[j + 1] = temp;

                    }

                } else if (emergency\_count[lane\_order[j]] < emergency\_count[lane\_order[j + 1]]) {

                    int temp = lane\_order[j];

                    lane\_order[j] = lane\_order[j + 1];

                    lane\_order[j + 1] = temp;

                }

            }

        }

        for (int i = 0; i < 4; i++) {

            int lane\_index = lane\_order[i];  // Lane index to process

            if (emergency\_count[lane\_index] > 0) {

                Vehicle \*temp = lanes[lane\_index].front;

                Vehicle \*farthest\_emergency = NULL;

                while (temp != NULL) {

                    if (temp->priority == 1)

                        farthest\_emergency = temp;

                    temp = temp->next;

                }

                printf("\033[1;32mGreen light for %s lane\n\033[0m", directions[lane\_index]);

                while (lanes[lane\_index].front != NULL) {

                    Vehicle \*dequeue\_vehicle = lanes[lane\_index].front;

                    lanes[lane\_index].front = lanes[lane\_index].front->next;

                    printf("  Vehicle ID: %d, Priority: %s\n", dequeue\_vehicle->id, dequeue\_vehicle->priority ? "Emergency" : "Regular");

                    add\_to\_log(dequeue\_vehicle->id, dequeue\_vehicle->priority, dequeue\_vehicle->direction);

                    if (dequeue\_vehicle->priority == 1)

                        emergency\_vehicle\_count--;

                    free(dequeue\_vehicle);

                    if (dequeue\_vehicle == farthest\_emergency)

                        break;

                }

                if (lanes[lane\_index].front == NULL)

                    lanes[lane\_index].rear = NULL;

                break;

            }

        }

    }

}

// Process and dequeue regular vehicles

void process\_regular\_vehicles(const char \*directions[], int total\_vehicles) {

    for (int i = 0; i < 4; i++) {

        if (!is\_empty(&lanes[i]))

            printf("\033[1;32mGreen light for %s lane\n\033[0m", directions[i]);

        int regular\_vehicle\_count = 0;  // Count of regular vehicles in the current lane

        Vehicle \*temp = lanes[i].front;

        while (temp != NULL) {

            if (temp->priority == 0)

                regular\_vehicle\_count++;

            temp = temp->next;

        }

        int other\_lanes\_have\_vehicles = 0;  // Check if other lanes have vehicles

        for (int j = 0; j < 4; j++) {

            if (j != i && !is\_empty(&lanes[j])) {

                other\_lanes\_have\_vehicles = 1;

                break;

            }

        }

        int allowed\_to\_dequeue = regular\_vehicle\_count;

        if (regular\_vehicle\_count > 1 && other\_lanes\_have\_vehicles)

            allowed\_to\_dequeue = (int)(regular\_vehicle\_count \* 0.7); // 70% of regular vehicles can be dequeued

        int dequeued\_count = 0;  // Count of dequeued vehicles

        while (lanes[i].front != NULL && dequeued\_count < allowed\_to\_dequeue) {

            Vehicle \*dequeue\_vehicle = lanes[i].front;

            lanes[i].front = lanes[i].front->next;

            printf("  Vehicle ID: %d, Priority: %s\n", dequeue\_vehicle->id, dequeue\_vehicle->priority ? "Emergency" : "Regular");

            add\_to\_log(dequeue\_vehicle->id, dequeue\_vehicle->priority, dequeue\_vehicle->direction);

            free(dequeue\_vehicle);

            dequeued\_count++;

        }

        if (lanes[i].front == NULL) {

            lanes[i].rear = NULL;

        } else {

            printf("\033[1;33mRemaining vehicles in %s lane will be dequeued in the next turn.\n\033[0m", directions[i]);

        }

    }

}

// Manage overall traffic by coordinating emergency and regular vehicle processing

void manage\_traffic() {

    int total\_vehicles = 0;  // Total count of vehicles in all lanes

    const char \*directions[4] = {"North", "East", "South", "West"};

    for (int i = 0; i < 4; i++)

        total\_vehicles += count\_vehicles\_in\_lane(i);

    if (total\_vehicles == 0) {

        printf("\033[1;30mNO VEHICLES IN ANY LANE.\n\033[0m");

        return;

    } else {

        printf("\033[1;31mALL LANES ARE RED\n\033[0m");

    }

    process\_emergency\_vehicles(directions);

    process\_regular\_vehicles(directions, total\_vehicles);

    for (int i = 0; i < 4; i++) {

        if (!is\_empty(&lanes[i])) {

            printf("\033[1;32mGreen light for %s lane\n\033[0m", directions[i]);

            while (lanes[i].front != NULL) {

                Vehicle \*dequeue\_vehicle = lanes[i].front;

                lanes[i].front = lanes[i].front->next;

                printf("  Vehicle ID: %d, Priority: %s\n", dequeue\_vehicle->id, dequeue\_vehicle->priority ? "Emergency" : "Regular");

                add\_to\_log(dequeue\_vehicle->id, dequeue\_vehicle->priority, dequeue\_vehicle->direction);

                free(dequeue\_vehicle);

            }

            if (lanes[i].front == NULL)

                lanes[i].rear = NULL;

        }

    }

    for (int i = 0; i < 4; i++)

        free\_vehicles(&lanes[i]);

}

// Print the log of processed vehicles

void print\_traffic\_log() {

    if (traffic\_log == NULL) {

        printf("\033[1;30mNO VEHICLES IN TRAFFIC LOG.\n\033[0m");

    } else {

        Log \*temp = traffic\_log;

        while (temp != NULL) {

            printf("Vehicle ID: %d, Direction: %s, Priority: %s\n", temp->id, temp->direction, temp->priority ? "Emergency" : "Regular");

            temp = temp->next;

        }

    }

}

// Free the memory allocated for the log

void free\_log() {

    Log \*temp;

    while (traffic\_log != NULL) {

        temp = traffic\_log;

        traffic\_log = traffic\_log->next;

        free(temp);

    }

}

// Main function: provides menu for user interaction

int main() {

    create\_lanes();

    while (1) {

        int choice;      // User's menu choice

        char input[10];  // Input buffer for user's choice

        printf("\033[1;34m\nMENU:\n");

        printf("1. Create Traffic\n");

        printf("2. Display All Vehicles\n");

        printf("3. Manage Traffic\n");

        printf("4. Traffic Log\n");

        printf("Press $ to exit.\n");

        printf("Enter your choice: \033[0m");

        scanf("%s", input);

        if (strcmp(input, "$") == 0) {

            printf("\033[1;30mEXITING...\n\n\033[0m");

            break;

        }

        choice = atoi(input);

        switch (choice) {

        case 1:

            create();

            break;

        case 2:

            display();

            break;

        case 3:

            manage\_traffic();

            break;

        case 4:

            print\_traffic\_log();

            break;

        default:

            printf("\033[1;30mInvalid choice. Please try again.\n\033[0m");

            break;

        }

    }

    for (int i = 0; i < 4; i++)

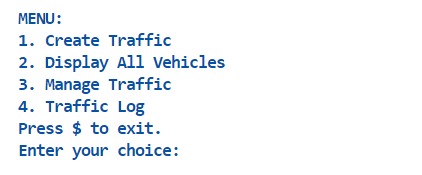
        free\_vehicles(&lanes[i]);

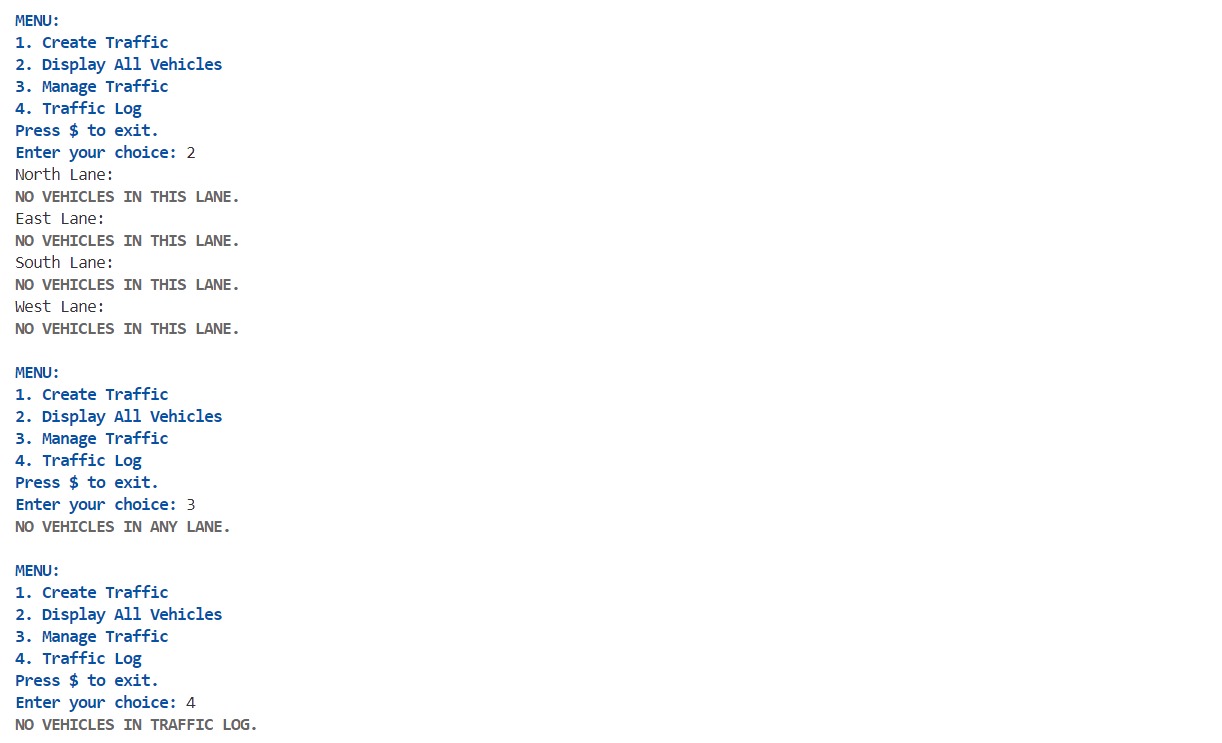
    free\_log();

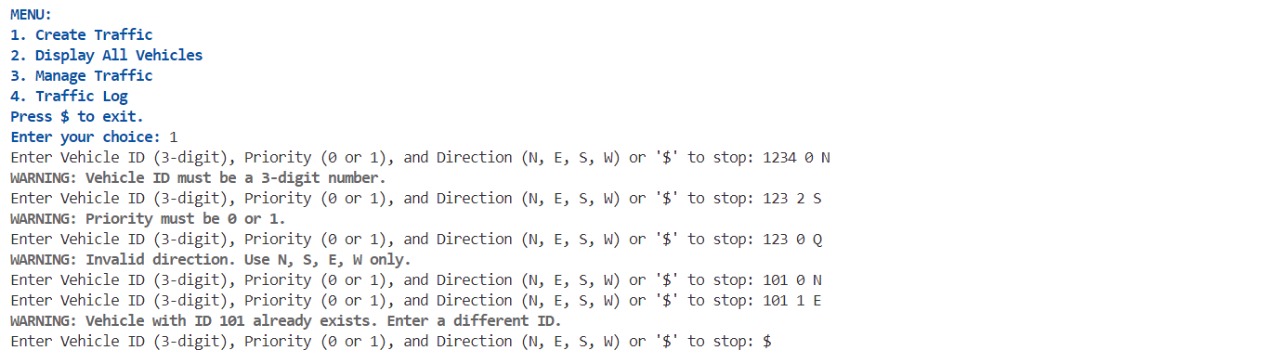
    return 0;

}

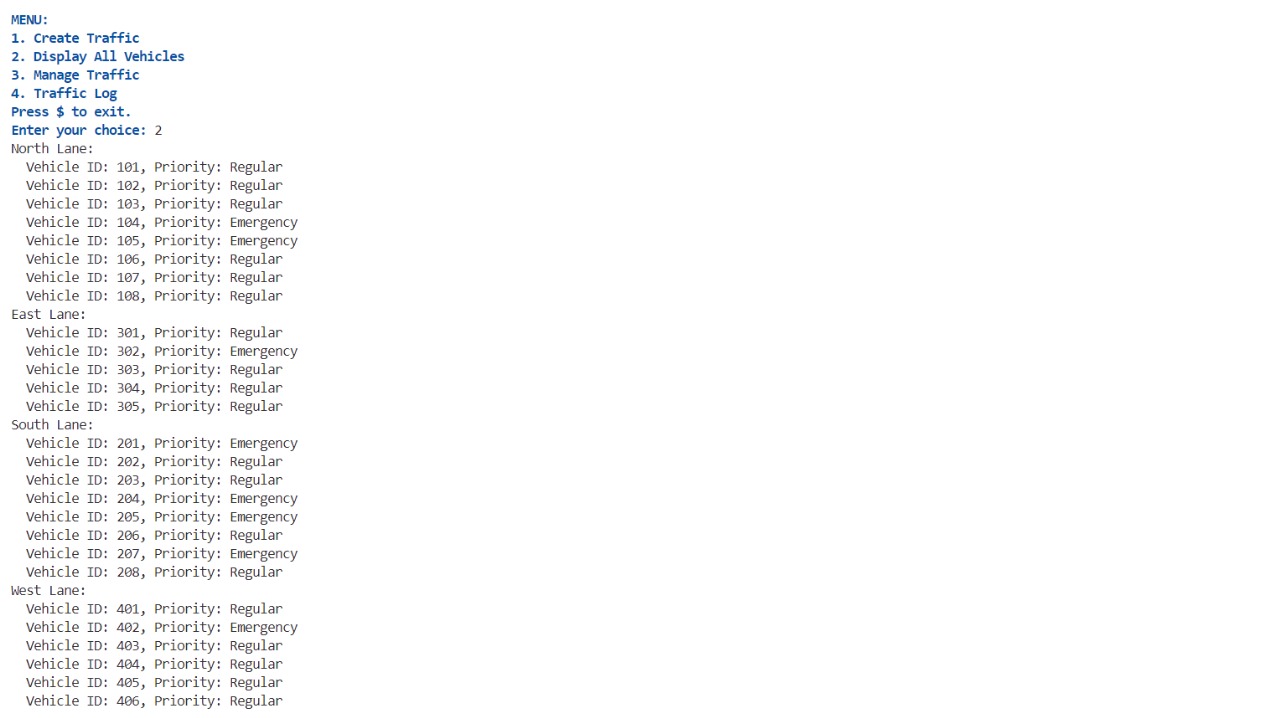
# Output:

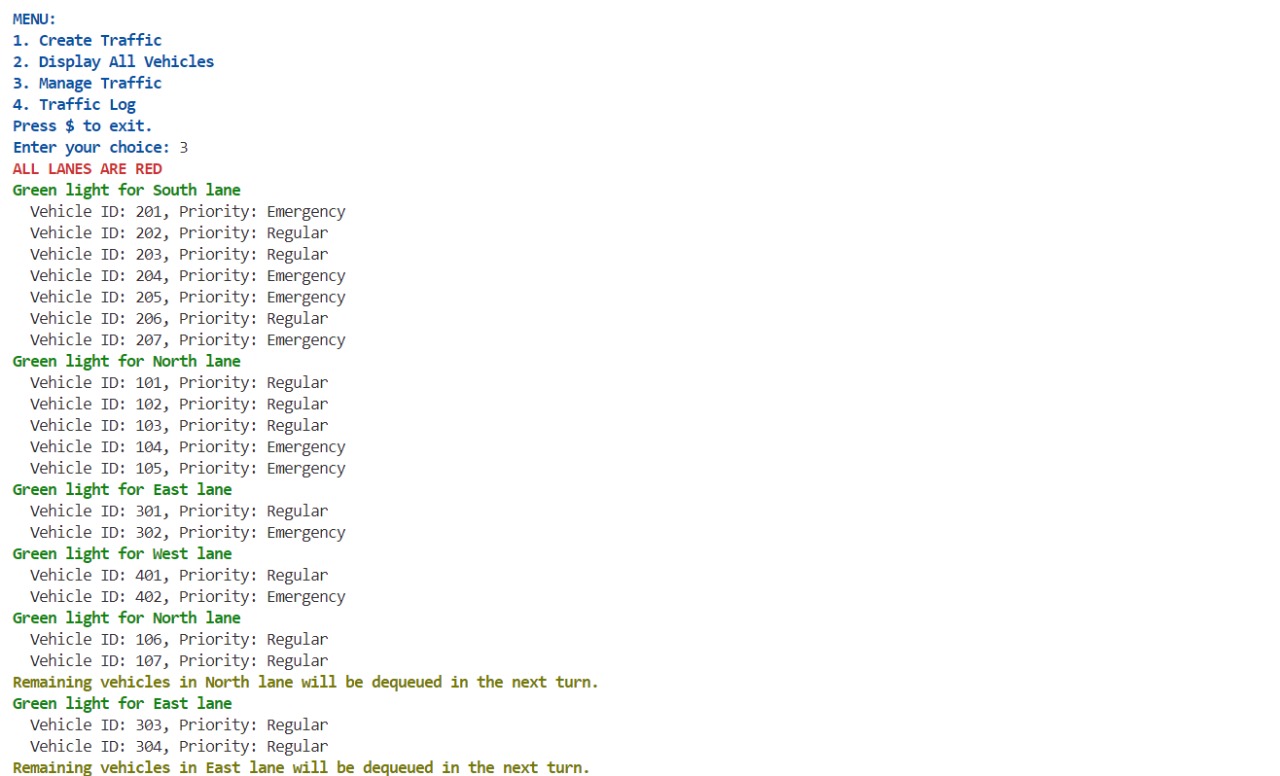
****

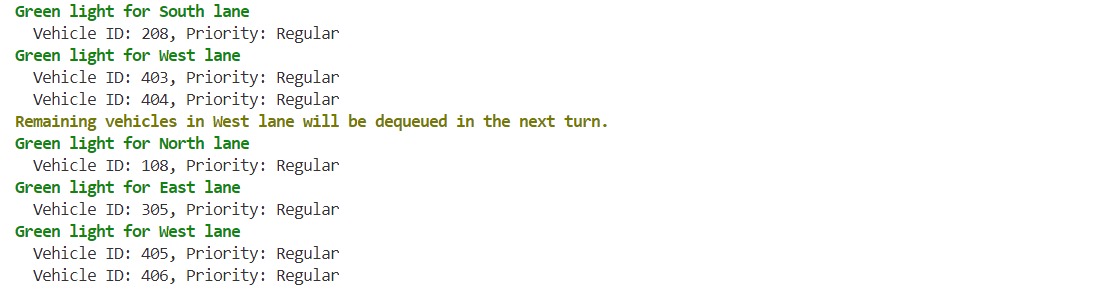
****







****

****

****

****

**CONCLUSION**

In conclusion, the traffic signal algorithm developed in this project represents a significant advancement in urban traffic management, addressing critical issues such as congestion, delays, and emergency vehicle prioritization. By efficiently managing the flow of vehicles at a four-way intersection, the algorithm enhances overall traffic efficiency while ensuring the timely passage of emergency responders.

This dual objective is pivotal in urban environments where rapid response times can make a life-saving difference. By prioritizing emergency vehicles without compromising regular traffic flow, the algorithm strikes a balance that supports both safety and operational efficiency.

Moreover, the integration of advanced data structures like queues and linked lists plays a pivotal role in the algorithm's functionality. These structures enable systematic vehicle management and comprehensive logging of traffic movements, facilitating real-time monitoring and analysis.

The ability to maintain accurate records of vehicle flow not only aids in optimizing signal timings but also provides valuable insights for urban planners and traffic engineers. This data-driven approach empowers decision-making processes aimed at enhancing intersection efficiency, reducing environmental impact, and improving overall urban mobility.

Furthermore, the iterative development process employed in this project underscores its adaptability and responsiveness to user needs and operational realities. Through continuous testing, feedback incorporation, and refinement, the algorithm has evolved into a robust solution that meets diverse stakeholder requirements.

This iterative approach ensures that the algorithm not only meets functional specifications but also aligns with evolving traffic patterns and urban dynamics. As cities grow and face increasing traffic challenges, the implementation of intelligent traffic management systems becomes ever more critical. The traffic signal algorithm presented here sets a precedent for future innovations in urban infrastructure, aiming to create smarter, safer, and more resilient cities worldwide.

# FUTURISTIC SCOPE

Looking ahead, the traffic signal algorithm showcased in this project has promising avenues for future development and integration into smart city frameworks. As cities worldwide grapple with increasing urbanization, the need for intelligent transportation systems (ITS) becomes more critical. Here are several futuristic possibilities for the algorithm:

1. **IoT and AI Integration**: By incorporating Internet of Things (IoT) devices and Artificial Intelligence (AI), the algorithm can dynamically adjust signal timings based on real-time traffic data. This integration can predict traffic patterns, optimize intersections, and enhance safety by responding to changing conditions autonomously.
2. **V2X Communication**: Vehicle-to-Everything (V2X) communication technology can enable direct communication between vehicles, infrastructure, and traffic signals. This capability allows for coordinated traffic management, prioritization of emergency vehicles, and safer interactions between different road users.
3. **Predictive Analytics**: Implementing predictive analytics can forecast traffic demand and preemptively adjust signal timings. This proactive approach helps mitigate congestion during peak hours and special events, improving overall traffic flow efficiency.
4. **Environmental Considerations**: Future iterations can focus on reducing emissions and energy consumption by minimizing idle times at intersections. Optimizing traffic flow contributes to lowering carbon footprints and enhancing air quality in urban environments.
5. **Multi-modal Integration**: As cities embrace diverse transportation modes like bicycles, scooters, and public transit, the algorithm can evolve to accommodate these options. Integrating multi-modal data enables holistic traffic management strategies that promote sustainable urban mobility.
6. **Smart City Integration**: Serving as a core component of smart city ecosystems, the algorithm can interface with other urban systems like smart grids and public safety networks. This integration supports comprehensive city management approaches aimed at improving livability and resilience.
7. **Global Urbanization Trends**: Scalability and adaptability across different urban landscapes and demographic contexts will be crucial. The algorithm's versatility can cater to the evolving needs of megacities and developing regions, contributing to more efficient and inclusive urban transportation solutions.

In summary, the algorithm's future lies in its ability to evolve into a sophisticated, adaptive system that optimizes traffic management, enhances safety, and supports sustainable urban development. By leveraging technological advancements and collaborative urban planning, this project can contribute to creating smarter, more liveable cities globally.

# REFERENCES

* + [**https://github.com/kanavsethi/Music-Player-Using-Data-Structures**](https://github.com/kanavsethi/Music-Player-Using-Data-Structures)
  + [**https://ds.cs.rutgers.edu/assignment-music-playlist/**](https://ds.cs.rutgers.edu/assignment-music-playlist/)
  + [**https://cplusplus.com/forum/beginner/120630/**](https://cplusplus.com/forum/beginner/120630/)
  + [**http://ijarse.com/images/fullpdf/1493545969\_C740ijarse.pdf**](http://ijarse.com/images/fullpdf/1493545969_C740ijarse.pdf)
  + [**https://convertio.co/mp3-wav/**](https://convertio.co/mp3-wav/)
  + [**https://www.embarcadero.com/free-tools/dev-cpp**](https://www.embarcadero.com/free-tools/dev-cpp)