CSC4202

DESIGN AND ANALYSIS OF ALGORITHMS

SEMESTER 2 YEAR 2023/2024

REPORT GROUP PROJECT

NAME	:	‘AISYAH HUMAIRA BINTI NASRUDIN (214732)
		NURFARHANNIS NABILA BINTI MOHD FAUZI (216535)
		AHGALLYAH A/P VISWANATHAN (214747)
TOPIC	:	TRAFFIC LIGHT SYSTEM
PROGRAMME	:	BACHELOR OF COMPUTER SCIENCE WITH HONORS
GROUP	:	GROUP 1
LECTURER	:	PROF. MADYA DR. NORWATI MUSTAPHA

APPENDIX

Initial Project Plan

Group Name	Master Trio
Members	<table><tr><th>Name</th><th>Email</th><th>Phone number</th></tr><tr><td>'Aisyah Humaira binti Nasrudin</td><td>214732@student.upm.edu.my</td><td>010-8316690</td></tr><tr><td>Nurfarhannis Nabila binti Mohd Fauzi</td><td>216535@student.upm.edu.my</td><td>013-6898473</td></tr><tr><td>Ahgallyah A/P Viswanathan</td><td>214747@student.upm.edu.my</td><td>012-3170694</td></tr></table>
Problem scenario description	Imagine the bustling city of Innovatia, a vibrant hub of innovation and culture, where residents and visitors alike enjoy its modern amenities, historical landmarks, and thriving business districts. However, despite its many attractions, Innovatia faces a significant challenge: severe traffic congestion, particularly during peak hours when commuters flood the streets, leading to frustratingly long travel times and increased pollution. Each day, the residents of Innovatia struggle with gridlocked roads as they attempt to get to work, drop their children off at school, or simply run errands. The daily commute, which should take no more than 30 minutes, often stretches into hours of bumper-to-bumper traffic. This congestion not only affects the quality of life but also hinders economic productivity and contributes to environmental degradation. Recognizing the urgent need to address this issue, the city council wants to design an intelligent traffic management system to optimize traffic flow and reduce congestion. The goal is to minimize the total travel time for all vehicles by dynamically adjusting traffic light timings at intersections.
Why it is important	Importance of addressing traffic congestion in Innovatia can be specified as: 1) Economic Impact i. Lost Productivity **Extended Commute Times**: Longer travel times mean that residents spend more time on the road and less time at work or engaging in productive activities. **Business Operations**: Businesses suffer from delays in employee arrivals and disruptions in the delivery of goods and services. 2) Safety i. Accident Risks **Increased Incidents**: Congested roads are more prone to accidents, including rear-end collisions and fender benders, due to frequent stopping and starting. **Emergency Response**: Traffic congestion can delay emergency response times, potentially leading to more severe outcomes in critical situations.

	3) Environmental Impact i. Air Pollution **Emissions**: Traffic congestion results in higher emissions of pollutants such as nitrogen oxides (NOx), carbon monoxide (CO), and particulate matter (PM), which degrade air quality. **Greenhouse Gases**: Increased fuel consumption contributes to higher emissions of carbon dioxide (CO2), a significant greenhouse gas, exacerbating climate change. ii. Noise Pollution **Urban Noise**: Congested traffic leads to higher levels of noise pollution, which can adversely affect the health and well-being of residents.
Problem specification	The primary focus of our issue lies at road intersections. The traffic control issue at a particular intersection involves determining the allocation of time for each traffic flow direction in order to optimise a performance measure, while ensuring the safe movement of vehicles, within a given time period T. Common metrics are the aggregate count of stops, the duration of waiting time, and the lengths of queues.
Potential solutions	Applying dynamic programming (DP) to address the traffic light timing optimization problem entails decomposing the problem into smaller, more manageable sub-problems and methodically solving them to get the best answer. Here is a methodical way to solving this problem using dynamic programming: **State Definition:** ● Denoted as S(t,i), defines the ideal performance metric (such as minimal waiting time or minimum queue length) at time t while the traffic signal is in phase i. ● The phase i may indicate several traffic signal orientations, such as north-south green and east-west green. **State Transition:** ● Analyze the process of transitioning from one phase to another by considering the timing of the traffic signals and the metrics of performance. ● Define transition functions that modify the state by taking into account the current phase and the assigned green time. **Decision Variables:** ● The decision variable represents the length of time that the traffic light remains green for each traffic direction throughout each phase. ● Let gi denote the duration of the green signal assigned to phase i. **Objective Function:** The goal is to minimise the cumulative waiting time, line lengths, and number of stops within the specified time period T.

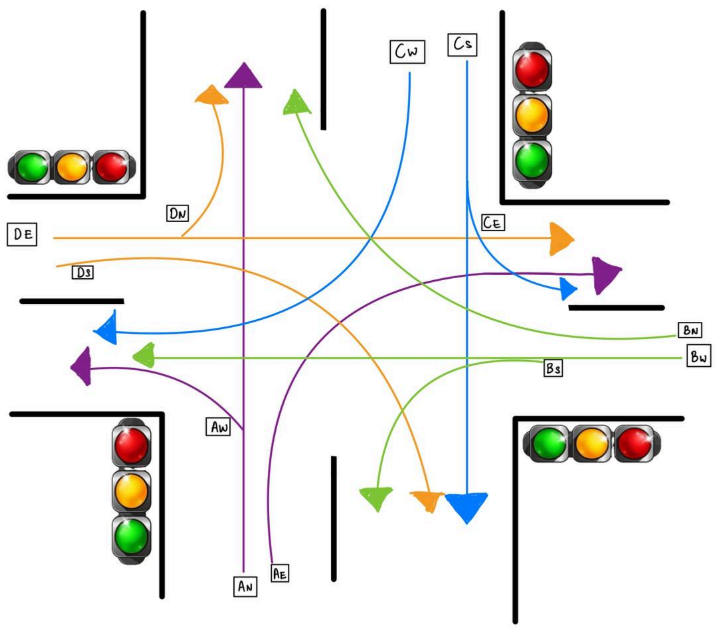
Sketch (framework, flow, interface)	**Pseudocode:**

```
# Initialize DP table
DP = [[float('inf')] * num_phases for _ in range(T+1)]
# Base case
for i in range(num_phases):
    DP[0][i] = initial_performance_measure(i)

# Fill DP table
for t in range(1, T+1):
    for i in range(num_phases):
        for g_i in possible_green_times:
            for j in range(num_phases): # Previous phase
                new_performance_measure = DP[t-1][j] + calculate_performance_measure(t, i, g_i)
                DP[t][i] = min(DP[t][i], new_performance_measure)

# Backtrack to find the optimal solution
optimal_solution = []
current_phase = min(range(num_phases), key=lambda i: DP[T][i])
for t in range(T, 0, -1):
    optimal_solution.append((t, current_phase))
    current_phase = backtrack_to_previous_phase(t, current_phase)

# Reverse the solution to get the correct order
optimal_solution.reverse()
```

Project Proposal Refinement

Group Name	Master Trio
Members	

Name	Email	Phone number
‘Aisyah Humaira binti Nasrudin	214732@student.upm.edu.my	010-8316690
Nurfarhannis Nabila binti Mohd Fauzi	216535@student.upm.edu.my	013-6898473
Ahgallyah A/P Viswanathan	214747@student.upm.edu.my	012-3170694

Problem statement	Innovatia, a bustling city known for its innovation and culture, faces a critical issue of severe traffic congestion, particularly during peak hours. This congestion leads to frustratingly long travel times, hindering economic productivity and contributing to environmental degradation. The city council seeks to address this challenge by designing an intelligent traffic management system to optimize traffic flow and reduce congestion, aiming to minimize total travel time for all vehicles by dynamically adjusting traffic light timings at intersections.
Objectives	**Objective 1** Improve business operations by optimizing traffic flow to minimize delays in employee arrivals and disruptions in the delivery of goods and services. **Objective 2** Enhance road safety by implementing measures to reduce accident risks associated with congested roads, thereby decreasing the frequency and severity of traffic incidents. **Objective 3** Improve emergency response times by mitigating traffic congestion, ensuring swift access for emergency vehicles to critical locations.
Expected output	 Cw Cs DN CE DE DS BN BW BS AW AE AN

	The diagram above illustrates a traffic light system at a four-way intersection in Innovatia. Here's an explanation of the expected output based on the diagram: 1. **Traffic Lights:** - The intersection has traffic lights on all four corners. - Each set of traffic lights controls the flow of traffic from a specific direction: North, East, South, and West. 2. **Lane Markings:** - The intersection shows multiple lanes for each direction, indicating different lanes for different turns and movements. 3. **Arrows and Colors:** - Different colored arrows represent the permitted directions for traffic flow from each lane. - **Green arrows:** 3 green arrows which indicate direction vehicles from B (East) to West, North, South. - **Orange arrows:** 3 orange arrows which indicate direction vehicles from D (West) to East, North, South. - **Blue arrows:** 3 blue arrows which indicate direction vehicles from C (North) to West, East, South. - **Purple arrows:** 3 purple arrows which indicate direction vehicles from A (South) to West, North, East. 4. **Labeling of Lanes:** - Lanes are labeled with codes of starting point and ending point such as DN, DE, DS, DW, CW, CE, etc., to denote specific traffic movements from those lanes. - The labels can correspond to specific traffic phases or signals for better traffic management. 5. **Traffic Flow:** - The arrows in the diagram illustrate how vehicles are expected to move through the intersection based on the traffic light signals. - For example: - Vehicles in lane DE can only go straight. - Vehicles in lane DN can only turn left. - Vehicles in lane CW can only turn right. 6. **Coordination of Lights:** - The traffic lights are likely coordinated to allow for smooth flow of traffic from all directions while minimizing conflicts and ensuring safety. 7. **Specific Movements:** - Each set of lights changes to green in a specific sequence to allow safe passage of vehicles from different directions. - The system ensures that when one direction has a green light, the opposite direction might have a red light to prevent collisions.

	In summary, this diagram is a detailed representation of how traffic lights manage the flow of vehicles at a busy intersection, indicating the sequence and direction of traffic movements to ensure smooth and safe driving conditions.
Problem scenario description	Imagine the bustling city of Innovatia, a vibrant hub of innovation and culture, where residents and visitors alike enjoy its modern amenities, historical landmarks, and thriving business districts. However, despite its many attractions, Innovatia faces a significant challenge: severe traffic congestion, particularly during peak hours when commuters flood the streets, leading to frustratingly long travel times and increased pollution. Each day, the residents of Innovatia struggle with gridlocked roads as they attempt to get to work, drop their children off at school, or simply run errands. The daily commute, which should take no more than 30 minutes, often stretches into hours of bumper-to-bumper traffic. This congestion not only affects the quality of life but also hinders economic productivity and contributes to environmental degradation. Recognizing the urgent need to address this issue, the city council wants to design an intelligent traffic management system to optimize traffic flow and reduce congestion. The goal is to minimize the total travel time for all vehicles by dynamically adjusting traffic light timings at intersections.
Why it is important	Importance of addressing traffic congestion in Innovatia can be specified as: 1. Economic Impact i. Lost Productivity **Extended Commute Times**: Longer travel times mean that residents spend more time on the road and less time at work or engaging in productive activities. **Business Operations**: Businesses suffer from delays in employee arrivals and disruptions in the delivery of goods and services. 2. Safety i. Accident Risks **Increased Incidents**: Congested roads are more prone to accidents, including rear-end collisions and fender benders, due to frequent stopping and starting. **Emergency Response**: Traffic congestion can delay emergency response times, potentially leading to more severe outcomes in critical situations. 3. Environmental Impact i. Air Pollution **Emissions**: Traffic congestion results in higher emissions of pollutants such as nitrogen oxides (NOx), carbon monoxide (CO), and particulate matter (PM), which degrade air quality. **Greenhouse Gases**: Increased fuel consumption contributes to higher emissions of carbon dioxide (CO2), a significant greenhouse gas, exacerbating climate change. ii. Noise Pollution **Urban Noise**: Congested traffic leads to higher levels of noise pollution, which can adversely affect the health and well-being of residents.

Problem specification	The primary focus of our issue lies at road intersections. The traffic control issue at a particular intersection involves determining the allocation of time for each traffic flow direction in order to optimise a performance measure, while ensuring the safe movement of vehicles, within a given time period T. Common metrics are the aggregate count of stops, the duration of waiting time, and the lengths of queues.
Potential solutions	Applying dynamic programming (DP) to address the traffic light timing optimization problem entails decomposing the problem into smaller, more manageable sub-problems and methodically solving them to get the best answer. Here is a methodical way to solving this problem using dynamic programming: **State Definition:** ● Denoted as S(t,i), defines the ideal performance metric (such as minimal waiting time or minimum queue length) at time t while the traffic signal is in phase i. ● The phase i may indicate several traffic signal orientations, such as north-south green and east-west green. **State Transition:** ● Analyze the process of transitioning from one phase to another by considering the timing of the traffic signals and the metrics of performance. ● Define transition functions that modify the state by taking into account the current phase and the assigned green time. **Decision Variables:** ● The decision variable represents the length of time that the traffic light remains green for each traffic direction throughout each phase. ● Let gi denote the duration of the green signal assigned to phase i. **Objective Function:** ● The goal is to minimise the cumulative waiting time, line lengths, and number of stops within the specified time period T.

Sketch (framework, flow, interface)	import matplotlib.pyplot as plt
	
	class TrafficLightPhase:
	    def __init__(self, direction, movement, duration):
	        """
	        Initialize a TrafficLightPhase object.
	
	        Parameters:

```
        - direction (str): Direction of the traffic light phase (e.g., 'A (South)', 'B
(East)', etc.).
        - movement (str): Movement associated with the phase (e.g., 'Straight',
'Left', 'Right').
        - duration (int): Duration of the phase in seconds.
        """
        self.direction = direction
        self.movement = movement
        self.duration = duration

def main():
    num_phases = 12
    total_time_steps = 60
    green_durations = [5] * num_phases  # Example durations for each phase in
seconds

    directions = ['A (South)', 'B (East)', 'C (North)', 'D (West)']
    movements = ['Straight', 'Left', 'Right']
    phases = [TrafficLightPhase(directions[i // 3], movements[i % 3],
green_durations[i]) for i in range(num_phases)]

    current_time_step = 0
    current_phase_index = 0
    current_phase_time_left = phases[current_phase_index].duration

    phase_changes = []

    # Simulate the traffic light phases over time steps
    while current_time_step < total_time_steps:
        phase_changes.append((current_time_step, current_phase_index))
        print(f"At time step {current_time_step}, phase is {current_phase_index +
1} ({phases[current_phase_index].direction} -
{phases[current_phase_index].movement}) for
{phases[current_phase_index].duration} seconds")

        current_time_step += 1
        current_phase_time_left -= 1

        if current_phase_time_left == 0:
            current_phase_index = (current_phase_index + 1) % num_phases
            current_phase_time_left = phases[current_phase_index].duration

    # Plot the phases over time
    plot_phase_changes(phase_changes, phases, total_time_steps)

    # Plot the time complexities (best case, worst case, average case)
    plot_time_complexities(total_time_steps)

def plot_phase_changes(phase_changes, phases, total_time_steps):
    """
    Plot the traffic light phases over time.
    Parameters:
    - phase_changes (list of tuples): List of (time_step, phase_index) tuples
representing phase changes over time.
    - phases (list of TrafficLightPhase): List of TrafficLightPhase objects
representing each phase.
    - total_time_steps (int): Total simulation time in time steps.
    """
    times = [change[0] for change in phase_changes]
    phase_indices = [change[1] for change in phase_changes]
    phase_labels = [f'{phases[i].direction} - {phases[i].movement}' for i in
range(len(phases))]

    plt.figure(figsize=(12, 6))
    plt.plot(times, phase_indices, marker='o', linestyle='-', color='b')
    plt.title('Traffic Light Phases Over Time')
    plt.xlabel('Time (seconds)')
    plt.ylabel('Phase')
    plt.xticks(range(0, total_time_steps + 1, 5))
    plt.yticks(range(len(phases)), labels=phase_labels)
    plt.grid(True)
    plt.tight_layout()

def plot_time_complexities(total_time_steps):
    """
    Plot the time complexities: best case (O(1)), worst case (O(total_time_steps)),
and average case (O(total_time_steps)).

    Parameters:
    - total_time_steps (int): Total number of time steps considered for the
complexity analysis.
    """
    x = list(range(1, total_time_steps + 1))
    y_best = [1] * total_time_steps  # Best case: O(1)
    y_worst = list(range(1, total_time_steps + 1))  # Worst case:
O(total_time_steps)
    y_average = list(range(1, total_time_steps + 1))  # Average case:
O(total_time_steps)

    # Create subplots with horizontal arrangement
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))

    # Plot best case
    ax1.plot(x, y_best, label='Best Case: O(1)', linestyle='-', color='g')
    ax1.set_title('Best Case Time Complexity')
    ax1.set_xlabel('Total Time Steps')
    ax1.set_ylabel('Operations')
    ax1.legend()
    ax1.grid(True)

    # Plot worst case
```

	```
	    ax2.plot(x, y_worst, label='Worst Case: O(total_time_steps)', linestyle='-',
	color='r')
	    ax2.set_title('Worst Case Time Complexity')
	    ax2.set_xlabel('Total Time Steps')
	    ax2.set_ylabel('Operations')
	    ax2.legend()
	    ax2.grid(True)

	    # Plot average case
	    ax3.plot(x, y_average, label='Average Case: O(total_time_steps)', linestyle='-',
	color='b')
	    ax3.set_title('Average Case Time Complexity')
	    ax3.set_xlabel('Total Time Steps')
	    ax3.set_ylabel('Operations')
	    ax3.legend()
	    ax3.grid(True)

	    plt.tight_layout()
	    plt.show()

	if __name__ == "__main__":
	    main()
	```
Methodology	see milestone table below

Milestone	Time
Project planning and research	Week 10
Algorithm Selection and Initial Implementation	Week 11
System integration and testing	Week 12
Conduct analysis of correctness and time complexity	Week 13
Online portfolio and presentation preparation	Week 14

Project Progress

Milestone 1	Project planning and research
Date (Week)	Week 10
Description/ sketch	**Problem scenario :** Imagine the bustling city of Innovatia, a vibrant hub of innovation and culture, where residents and visitors alike enjoy its modern amenities, historical landmarks, and thriving business districts. However, despite its many attractions, Innovatia faces a significant challenge: severe traffic congestion, particularly during peak hours when commuters flood the streets, leading to frustratingly long travel times and increased pollution. Each day, the residents of Innovatia struggle with gridlocked roads as they attempt to get to work, drop their children off at school, or simply run errands. The daily commute, which should take no more than 30 minutes, often stretches into hours of bumper-to-bumper traffic. This congestion not only affects the quality of life but also hinders economic productivity and contributes to environmental degradation. Recognizing the urgent need to address this issue, the city council wants to design an intelligent traffic management system to optimize traffic flow and reduce congestion. The goal is to minimize the total travel time for all vehicles by dynamically adjusting traffic light timings at intersections. Innovatia, a bustling city known for its innovation and culture, faces a critical issue of severe traffic congestion, particularly during peak hours. This congestion leads to frustratingly long travel times, hindering economic productivity and contributing to environmental degradation. The city council seeks to address this challenge by designing an intelligent traffic management system to optimize traffic flow and reduce congestion, aiming to minimize total travel time for all vehicles by dynamically adjusting traffic light timings at intersections. **Problem statement :** Innovatia, a bustling city known for its innovation and culture, faces a critical issue of severe traffic congestion, particularly during peak hours. This congestion leads to frustratingly long travel times, hindering economic productivity and contributing to environmental degradation. The city council seeks to address this challenge by designing an intelligent traffic management system to optimize traffic flow and reduce congestion, aiming to minimize total travel time for all vehicles by dynamically adjusting traffic light timings at intersections. **Problem specification :** The primary focus of our issue lies at road intersections. The traffic control issue at a particular intersection involves determining the allocation of time for each traffic flow direction in order to optimise a performance measure, while ensuring the safe movement of vehicles, within a given time period T. Common metrics are the aggregate count of stops, the duration of waiting time, and the lengths of queues.

Importance of addressing the traffic congestion in Innovatia :

1) Economic Impact

   i. Lost Productivity

   Extended Commute Times: Longer travel times mean that residents spend more time on the road and less time at work or engaging in productive activities.

   Business Operations: Businesses suffer from delays in employee arrivals and disruptions in the delivery of goods and services.

2) Safety

   i. Accident Risks

   Increased Incidents: Congested roads are more prone to accidents, including rear-end collisions and fender benders, due to frequent stopping and starting.

   Emergency Response: Traffic congestion can delay emergency response times, potentially leading to more severe outcomes in critical situations.

3) Environmental Impact

   i. Air Pollution

   Emissions: Traffic congestion results in higher emissions of pollutants such as nitrogen oxides (NOx), carbon monoxide (CO), and particulate matter (PM), which degrade air quality.

   Greenhouse Gases: Increased fuel consumption contributes to higher emissions of carbon dioxide (CO2), a significant greenhouse gas, exacerbating climate change.

   ii. Noise Pollution

   Urban Noise: Congested traffic leads to higher levels of noise pollution, which can adversely affect the health and well-being of residents.

Framework of Traffic LIght System in Innovatia :

1) **System Overview :**

   Objective: To optimize traffic flow and reduce congestion by dynamically adjusting traffic light timings at intersections.

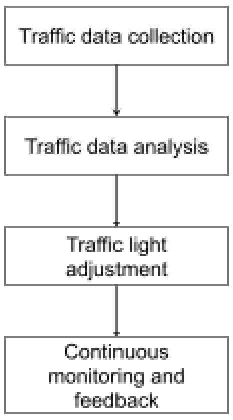
   Scope: All major intersections and roads within Innovatia.

   Stakeholders: City council, traffic management authorities, residents, commuters, businesses, environmental agencies.

2) **Data processing and analytics :**

Algorithms: Use machine learning algorithms to predict traffic patterns and adjust traffic light timings dynamically.

3) **Flow of program operations :**



Role

Aisyah Humaira	Nurfarhannis Nabila	Ahgallyah
- Find any problem that relates to the topic. - Find the suitable problem specification to focus on and potential solutions to solve the problem.	- Create the problem scenario based on the problem. - Sketch the flow on how to get the solution.	- Find why the problem is important to be solved. - Sketch the flow on how to get the solution.

Milestone 2	Algorithm Selection and Initial Implementation
Date (Week)	Week 11

Description/ sketch

Potential Solutions to solve the Traffic LIght System in Innovatia :

Applying dynamic programming (DP) to address the traffic light timing optimization problem entails decomposing the problem into smaller, more manageable sub-problems and methodically solving them to get the best answer. Here is a methodical way to solving this problem using dynamic programming:

State Definition:

- Denoted as S(t,i), defines the ideal performance metric (such as minimal waiting time or minimum queue length) at time t while the traffic signal is in phase i.
- The phase i may indicate several traffic signal orientations, such as north-south green and east-west green.

State Transition:

- Analyze the process of transitioning from one phase to another by considering the timing of the traffic signals and the metrics of performance.
- Define transition functions that modify the state by taking into account the current phase and the assigned green time.

Decision Variables:

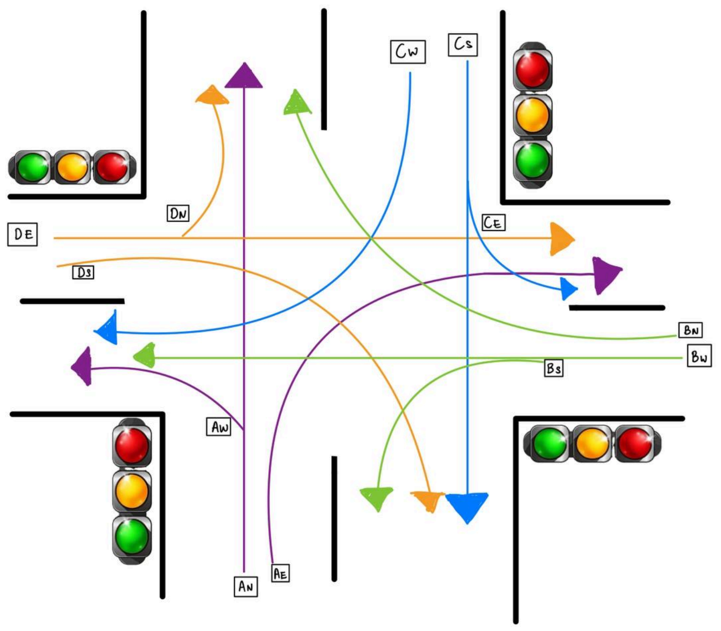
- The decision variable represents the length of time that the traffic light remains green for each traffic direction throughout each phase.
- Let gi denote the duration of the green signal assigned to phase i.

Objective Function:

- The goal is to minimise the cumulative waiting time, line lengths, and number of stops within the specified time period T.

Objectives of solving the Traffic LIght System in Innovatia :

Objective 1
Improve business operations by optimizing traffic flow to minimize delays in employee arrivals and disruptions in the delivery of goods and services.
Objective 2
Enhance road safety by implementing measures to reduce accident risks associated with congested roads, thereby decreasing the frequency and severity of traffic incidents.

	Objective 3 Improve emergency response times by mitigating traffic congestion, ensuring swift access for emergency vehicles to critical locations. **Expected output to get with the solutions :**  CW CS DN DE DS CE BN BW BS AW AE AN The diagram above illustrates a traffic light system at a four-way intersection in Innovatia. Here's an explanation of the expected output based on the diagram: 1. **Traffic Lights:** - The intersection has traffic lights on all four corners. - Each set of traffic lights controls the flow of traffic from a specific direction: North, East, South, and West. 2. **Lane Markings:** - The intersection shows multiple lanes for each direction, indicating different lanes for different turns and movements. 3. **Arrows and Colors:** - Different colored arrows represent the permitted directions for traffic flow from each lane. - **Green arrows:** 3 green arrows which indicate direction vehicles from B (East) to West, North, South. - **Orange arrows:** 3 orange arrows which indicate direction vehicles from D (West) to East, North, South. - **Blue arrows:** 3 blue arrows which indicate direction vehicles from C (North) to West, East, South. - **Purple arrows:** 3 purple arrows which indicate direction vehicles from A (South) to West, North, East.

4. **Labeling of Lanes:**
   - Lanes are labeled with codes of starting point and ending point such as DN, DE, DS, DW, CW, CE, etc., to denote specific traffic movements from those lanes.
   - The labels can correspond to specific traffic phases or signals for better traffic management.

5. **Traffic Flow:**
   - The arrows in the diagram illustrate how vehicles are expected to move through the intersection based on the traffic light signals.
   - For example:
     - Vehicles in lane DE can only go straight.
     - Vehicles in lane DN can only turn left.
     - Vehicles in lane CW can only turn right.

6. **Coordination of Lights:**
   - The traffic lights are likely coordinated to allow for smooth flow of traffic from all directions while minimizing conflicts and ensuring safety.

7. **Specific Movements:**
   - Each set of lights changes to green in a specific sequence to allow safe passage of vehicles from different directions.
   - The system ensures that when one direction has a green light, the opposite direction might have a red light to prevent collisions.

In summary, this diagram is a detailed representation of how traffic lights manage the flow of vehicles at a busy intersection, indicating the sequence and direction of traffic movements to ensure smooth and safe driving conditions.

Pseudocode of the system :

```
// Define constants and data structures
num_phases = 12  // Total number of phases (4 directions x 3 movements)
total_time_steps = 60  // Total simulation time in steps
green_durations = [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]  // Example durations for each
phase

directions = ['A (South)', 'B (East)', 'C (North)', 'D (West)']
movements = ['Straight', 'Left', 'Right']

// Initialize phase information
phases = []
for i = 0 to num_phases - 1:
   direction = directions[i // 3]
   movement = movements[i % 3]
   duration = green_durations[i]
   phases[i] = (direction, movement, duration)

// Simulation loop
current_time_step = 0
```

```
current_phase_index = 0
current_phase_time_left = phases[current_phase_index].duration

phase_changes = []

while current_time_step < total_time_steps:
   // Record current phase
   phase_changes.append((current_time_step, current_phase_index))

   // Print or log current phase details
   print("At time step", current_time_step, "phase is", current_phase_index, "(",
phases[current_phase_index].direction, "-",
phases[current_phase_index].movement, ")")

   // Update time step and remaining time in current phase
   current_time_step = current_time_step + 1
   current_phase_time_left = current_phase_time_left - 1

   // Check if current phase duration is over
   if current_phase_time_left == 0:
      // Move to the next phase
      current_phase_index = (current_phase_index + 1) % num_phases
      current_phase_time_left = phases[current_phase_index].duration

// Function to plot phase changes
function plot_phase_changes(phase_changes, num_phases, total_time_steps):
   // Extract time and phase indices for plotting
   times = [change[0] for change in phase_changes]
   phases = [change[1] for change in phase_changes]

   // Plot using a line graph
   plot(times, phases, marker='o')
   title('Traffic Light Phases Over Time')
   xlabel('Time')
   ylabel('Phase')
   xticks(range(0, total_time_steps + 1, 5))
   yticks(range(num_phases), labels=['A - Straight', 'A - Left', 'A - Right', 'B - Straight',
'B - Left', 'B - Right', 'C - Straight', 'C - Left', 'C - Right', 'D - Straight', 'D - Left', 'D -
Right'])
   grid(True)
   show()

// Main function
function main():
   // Run the simulation
   simulate_traffic_light(phases, num_phases, total_time_steps)

   // Plot the results
   plot_phase_changes(phase_changes, num_phases, total_time_steps)

// Execute main function
main()
```

Role	
	(nested table below)

Aisyah Humaira	Nurfarhannis Nabila	Ahgallyah
- Find example solutions from any resources on the internet to match the problem. - Find the objectives of solving the problem.	- Find example solutions from any resources on the internet to match the problem. - Sketch the expected output.	- Find example solutions from any resources on the internet to match the problem. - Update the pseudocode based on the solutions.

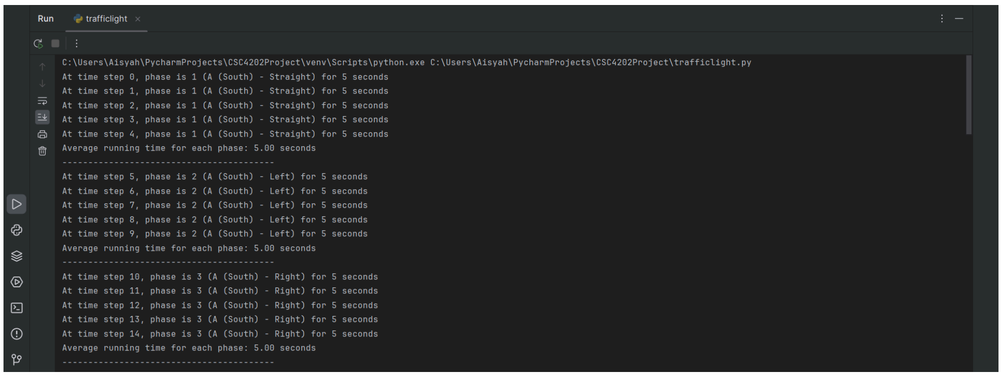
Milestone 3	System integration and testing
Date (Week)	Week 12
Description/ sketch	**Implement the system using Python code:**

```python
import matplotlib.pyplot as plt

class TrafficLightPhase:
   def __init__(self, direction, movement, duration):
       """
       Initialize a TrafficLightPhase object.

       Parameters:
       - direction (str): Direction of the traffic light phase (e.g., 'A (South)', 'B (East)', etc.).
       - movement (str): Movement associated with the phase (e.g., 'Straight', 'Left', 'Right').
       - duration (int): Duration of the phase in seconds.
       """
       self.direction = direction
       self.movement = movement
       self.duration = duration

def main():
   num_phases = 12
   total_time_steps = 60
   green_durations = [5] * num_phases  # Example durations for each phase in seconds

   directions = ['A (South)', 'B (East)', 'C (North)', 'D (West)']
   movements = ['Straight', 'Left', 'Right']
   phases = [TrafficLightPhase(directions[i // 3], movements[i % 3], green_durations[i]) for i in range(num_phases)]

   current_time_step = 0
   current_phase_index = 0
   current_phase_time_left = phases[current_phase_index].duration

   phase_changes = []

   # Simulate the traffic light phases over time steps
   while current_time_step < total_time_steps:
       phase_changes.append((current_time_step, current_phase_index))
       print(f"At time step {current_time_step}, phase is {current_phase_index + 1} ({phases[current_phase_index].direction} - {phases[current_phase_index].movement}) for {phases[current_phase_index].duration} seconds")

       current_time_step += 1
       current_phase_time_left -= 1

       if current_phase_time_left == 0:
           current_phase_index = (current_phase_index + 1) % num_phases
```

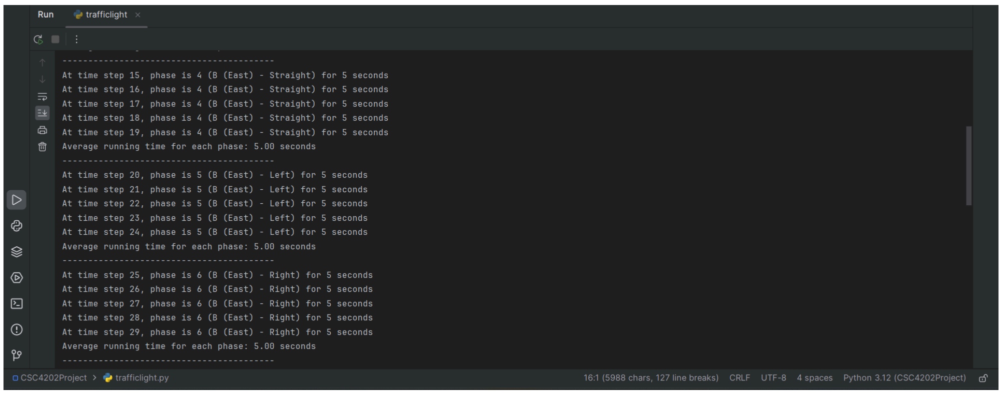
```
            current_phase_time_left =
phases[current_phase_index].duration

            # Calculate average running time for each phase
            average_duration = sum(phase.duration for phase in
phases) / len(phases)
            print(f"Average running time for each phase:
{average_duration:.2f} seconds")
            print("------------------------------------------")

    # Plot the time complexities (best case, worst case,
average case)
    plot_time_complexities(total_time_steps, phases)  # Pass
phases as an argument

    # Plot the phase changes over time
    plot_phase_changes(phase_changes, phases, total_time_steps)

def plot_time_complexities(total_time_steps, phases):
    """
    Plot the time complexities: best case (O(1)), worst case
(O(total_time_steps)),
    average case (O(total_time_steps)).

    Parameters:
    - total_time_steps (int): Total number of time steps
considered for the complexity analysis.
    - phases (list of TrafficLightPhase): List of
TrafficLightPhase objects representing each phase.
    """
    x = list(range(1, total_time_steps + 1))
    num_phases = len(phases)

    # Create y-values based on the phases
    y_best = [i % num_phases + 1 for i in
range(total_time_steps)]  # Best case: rotating phases
    y_worst = [(i + num_phases // 2) % num_phases + 1 for i in
range(total_time_steps)]  # Worst case: rotating phases
    y_average = [(i + num_phases // 4) % num_phases + 1 for i
in range(total_time_steps)]  # Average case: rotating phases

    # Create subplots with horizontal arrangement
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))

    # Plot best case
    ax1.plot(x, y_best, linestyle='-', color='g')
    ax1.set_title('Best Case Time Complexity')
    ax1.set_xlabel('Total Time Steps')
    ax1.set_ylabel('Phase')
    ax1.text(0.5, 0.9, r'$O(1)$', horizontalalignment='center',
verticalalignment='center', transform=ax1.transAxes,
fontsize=14, bbox=dict(facecolor='white', edgecolor='none',
pad=5))
    ax1.grid(True)

    # Plot worst case
    ax2.plot(x, y_worst, linestyle='-', color='r')
    ax2.set_title('Worst Case Time Complexity')
    ax2.set_xlabel('Total Time Steps')
    ax2.set_ylabel('Phase')
```

```
   ax2.text(0.5, 0.9, r'$O(total\_time\_steps)$',
horizontalalignment='center', verticalalignment='center',
transform=ax2.transAxes, fontsize=14,
bbox=dict(facecolor='white', edgecolor='none', pad=5))
   ax2.grid(True)

   # Plot average case
   ax3.plot(x, y_average, linestyle='-', color='b')
   ax3.set_title('Average Case Time Complexity')
   ax3.set_xlabel('Total Time Steps')
   ax3.set_ylabel('Phase')
   ax3.text(0.5, 0.9, r'$O(total\_time\_steps)$',
horizontalalignment='center', verticalalignment='center',
transform=ax3.transAxes, fontsize=14,
bbox=dict(facecolor='white', edgecolor='none', pad=5))
   ax3.grid(True)

   plt.tight_layout()
   plt.show()

def plot_phase_changes(phase_changes, phases,
total_time_steps):
   """
   Plot the traffic light phases over time.

   Parameters:
   - phase_changes (list of tuples): List of tuples (time
step, phase index) representing phase changes over time.
   - phases (list of TrafficLightPhase): List of
TrafficLightPhase objects representing each phase.
   - total_time_steps (int): Total number of time steps
considered for the simulation.
   """
   times = [change[0] for change in phase_changes]  # Extract
time steps from phase changes
   phase_indices = [change[1] for change in phase_changes]  #
Extract phase indices from phase changes
   phase_labels = [f'{phases[i].direction} -
{phases[i].movement}' for i in range(len(phases))]  # Create
labels for each phase

   plt.figure(figsize=(12, 6))  # Set the figure size for the
plot
   plt.plot(times, phase_indices, marker='o', linestyle='-',
color='b')  # Plot the phase changes over time
   plt.title('Traffic Light Phases Over Time')  # Set the
title of the plot
   plt.xlabel('Time (seconds)')  # Label the x-axis
   plt.ylabel('Phase')  # Label the y-axis
   plt.xticks(range(0, total_time_steps + 1, 5))  # Set the
x-axis ticks
   plt.yticks(range(len(phases)), labels=phase_labels)  # Set
the y-axis ticks and labels
   plt.grid(True)  # Enable the grid
   plt.tight_layout()  # Adjust the layout
   plt.show()  # Display the plot

if __name__ == "__main__":
   main()
```

Solution :

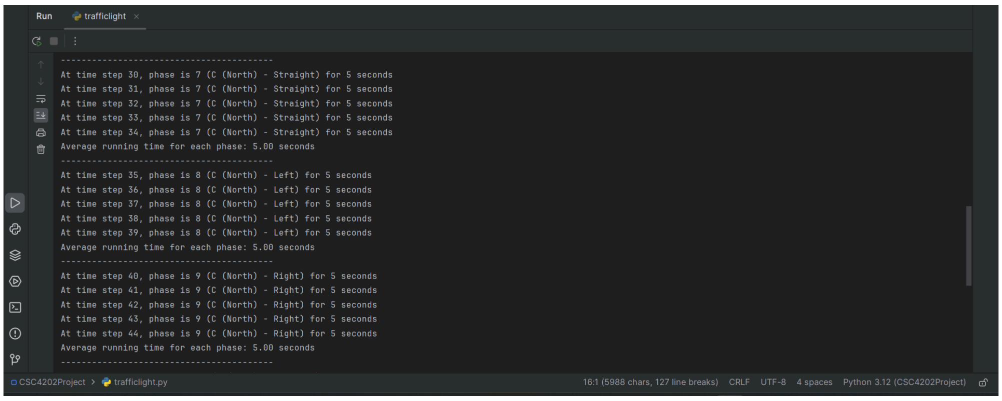


```
C:\Users\Aisyah\PycharmProjects\CSC4202Project\venv\Scripts\python.exe C:\Users\Aisyah\PycharmProjects\CSC4202Project\trafficlight.py
At time step 0, phase is 1 (A (South) - Straight) for 5 seconds
At time step 1, phase is 1 (A (South) - Straight) for 5 seconds
At time step 2, phase is 1 (A (South) - Straight) for 5 seconds
At time step 3, phase is 1 (A (South) - Straight) for 5 seconds
At time step 4, phase is 1 (A (South) - Straight) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 5, phase is 2 (A (South) - Left) for 5 seconds
At time step 6, phase is 2 (A (South) - Left) for 5 seconds
At time step 7, phase is 2 (A (South) - Left) for 5 seconds
At time step 8, phase is 2 (A (South) - Left) for 5 seconds
At time step 9, phase is 2 (A (South) - Left) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 10, phase is 3 (A (South) - Right) for 5 seconds
At time step 11, phase is 3 (A (South) - Right) for 5 seconds
At time step 12, phase is 3 (A (South) - Right) for 5 seconds
At time step 13, phase is 3 (A (South) - Right) for 5 seconds
At time step 14, phase is 3 (A (South) - Right) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
```



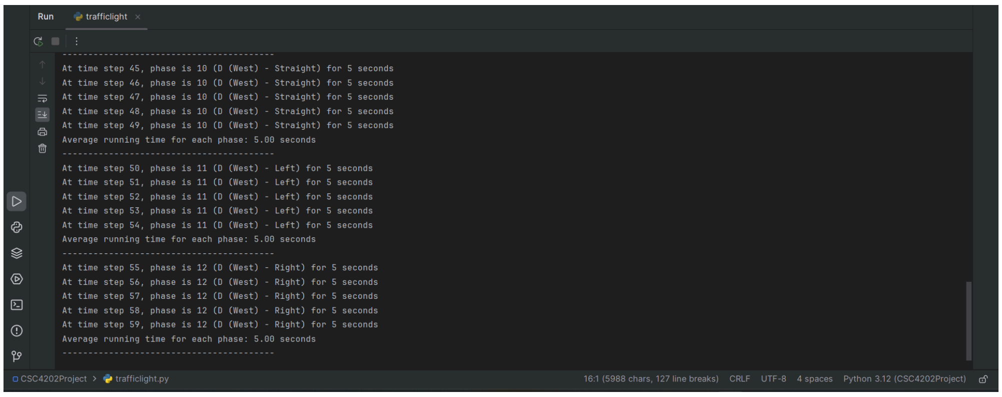
```
----------------------------------------
At time step 15, phase is 4 (B (East) - Straight) for 5 seconds
At time step 16, phase is 4 (B (East) - Straight) for 5 seconds
At time step 17, phase is 4 (B (East) - Straight) for 5 seconds
At time step 18, phase is 4 (B (East) - Straight) for 5 seconds
At time step 19, phase is 4 (B (East) - Straight) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 20, phase is 5 (B (East) - Left) for 5 seconds
At time step 21, phase is 5 (B (East) - Left) for 5 seconds
At time step 22, phase is 5 (B (East) - Left) for 5 seconds
At time step 23, phase is 5 (B (East) - Left) for 5 seconds
At time step 24, phase is 5 (B (East) - Left) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 25, phase is 6 (B (East) - Right) for 5 seconds
At time step 26, phase is 6 (B (East) - Right) for 5 seconds
At time step 27, phase is 6 (B (East) - Right) for 5 seconds
At time step 28, phase is 6 (B (East) - Right) for 5 seconds
At time step 29, phase is 6 (B (East) - Right) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
```

CSC4202Project > trafficlight.py   16:1 (5988 chars, 127 line breaks) CRLF UTF-8 4 spaces Python 3.12 (CSC4202Project)



```
----------------------------------------
At time step 30, phase is 7 (C (North) - Straight) for 5 seconds
At time step 31, phase is 7 (C (North) - Straight) for 5 seconds
At time step 32, phase is 7 (C (North) - Straight) for 5 seconds
At time step 33, phase is 7 (C (North) - Straight) for 5 seconds
At time step 34, phase is 7 (C (North) - Straight) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 35, phase is 8 (C (North) - Left) for 5 seconds
At time step 36, phase is 8 (C (North) - Left) for 5 seconds
At time step 37, phase is 8 (C (North) - Left) for 5 seconds
At time step 38, phase is 8 (C (North) - Left) for 5 seconds
At time step 39, phase is 8 (C (North) - Left) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 40, phase is 9 (C (North) - Right) for 5 seconds
At time step 41, phase is 9 (C (North) - Right) for 5 seconds
At time step 42, phase is 9 (C (North) - Right) for 5 seconds
At time step 43, phase is 9 (C (North) - Right) for 5 seconds
At time step 44, phase is 9 (C (North) - Right) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
```

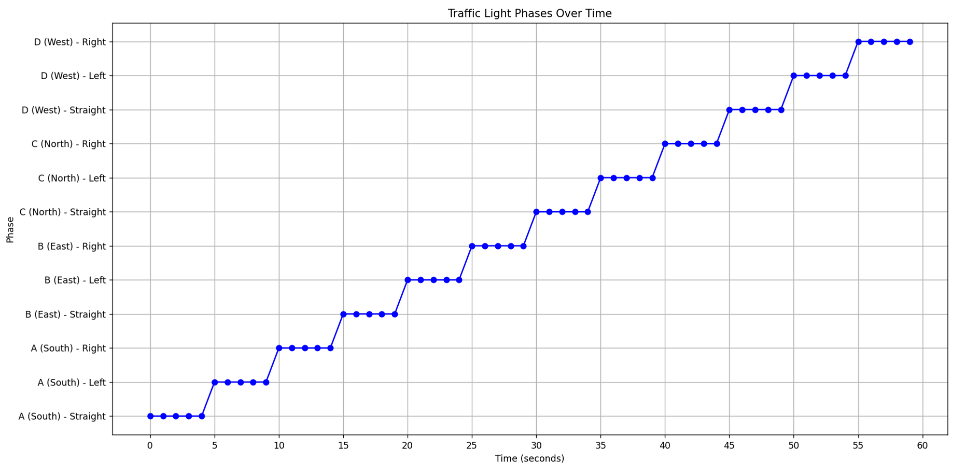
CSC4202Project > trafficlight.py   16:1 (5988 chars, 127 line breaks) CRLF UTF-8 4 spaces Python 3.12 (CSC4202Project)



```
----------------------------------------
At time step 45, phase is 10 (D (West) - Straight) for 5 seconds
At time step 46, phase is 10 (D (West) - Straight) for 5 seconds
At time step 47, phase is 10 (D (West) - Straight) for 5 seconds
At time step 48, phase is 10 (D (West) - Straight) for 5 seconds
At time step 49, phase is 10 (D (West) - Straight) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 50, phase is 11 (D (West) - Left) for 5 seconds
At time step 51, phase is 11 (D (West) - Left) for 5 seconds
At time step 52, phase is 11 (D (West) - Left) for 5 seconds
At time step 53, phase is 11 (D (West) - Left) for 5 seconds
At time step 54, phase is 11 (D (West) - Left) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
At time step 55, phase is 12 (D (West) - Right) for 5 seconds
At time step 56, phase is 12 (D (West) - Right) for 5 seconds
At time step 57, phase is 12 (D (West) - Right) for 5 seconds
At time step 58, phase is 12 (D (West) - Right) for 5 seconds
At time step 59, phase is 12 (D (West) - Right) for 5 seconds
Average running time for each phase: 5.00 seconds
----------------------------------------
```

CSC4202Project > trafficlight.py   16:1 (5988 chars, 127 line breaks) CRLF UTF-8 4 spaces Python 3.12 (CSC4202Project)

Output :



Graph Explanation:

1. Y-Axis (Phase):
    - This axis lists the different traffic light phases, which are combinations of directions (e.g., A (South), B (East), etc.) and movements (Straight, Left, Right).
    - There are a total of 12 phases, each labeled with its direction and movement.
2. X-Axis (Time (s)):
    - This axis represents the time steps in seconds over the total simulation time, which is 60 seconds in this case.
3. Data Points:
    - Each blue dot represents a phase change at a specific time step.
    - The graph shows a stair-step pattern where each horizontal line segment indicates the duration of a particular phase.

Code Explanation:

1. TrafficLightPhase Class:
    - This class initializes traffic light phases with direction, movement, and duration.
2. Simulation Parameters:
    - `num_phases` is set to 12, representing the total number of phases.
    - `total_time_steps` is set to 60, representing the total simulation time in seconds.
    - `green_durations` is a list of durations for each phase, here set uniformly to 5 seconds.
3. Phase Initialization:
    - Phases are initialized with a combination of directions and movements, with each phase having a duration of 5 seconds.
4. Simulation Loop:
    - The loop iterates through each time step, appending the current time step and phase index to `phase_changes`.

   - When the duration of the current phase expires, it moves to the next phase and resets the duration.
5. Plotting:
   - The function `plot_time_complexities` creates the graph by plotting the phase changes over time.
   - The stair-step pattern is generated by plotting the phase index for each time step.

Detailed Observation:

- Initial Phase (0-4 seconds): The traffic light starts at phase 1 (A (South) - Straight) for the first 5 seconds.
- Phase Transition (5-9 seconds): The traffic light switches to phase 2 (A (South) - Left) and stays for 5 seconds.
- Subsequent Transitions: This pattern continues, switching through all 12 phases every 5 seconds.
- End of Simulation (55-59 seconds): The final phase in the simulation is phase 12 (D (West) - Right), completing the cycle.

Role

Aisyah Humaira	Nurfarhannis Nabila	Ahgallyah
- Develop the code using Python language to find the solution that solve the problem.	- Develop the code using Python language to find the solution that solve the problem.	- Develop the code using Python language to find the solution that solve the problem.

Milestone 4	Conduct analysis of correctness and time complexity
Date (Week)	Week 13
Description/ sketch	**Analysis of correctness :** The algorithm and code provided simulate a traffic light control system over a predefined number of phases and time steps. Here are the important part : **1.Initialization and Setup :** **The TrafficLightPhase** class is defined to represent each phase with attributes like direction, movement, and duration. In main(), num_phases is set to 12, corresponding to 4 directions (A, B, C, D) and 3 movements per direction (Straight, Left, Right). green_durations list provides the duration (in time steps) for each phase. **2. Simulation Loop :** The simulation runs for **total_time_steps** time steps. The phases list is populated with TrafficLightPhase instances based on the predefined directions, movements, and durations. current_phase_index and current_phase_time_left keep track of the current phase and the remaining time for that phase. **3. Phase Transition Logic:** At each time step, the current phase index is incremented when the duration for the current phase is exhausted (**current_phase_time_left == 0**). The modulo operation ensures that phase index wraps around correctly (**(current_phase_index + 1) % num_phases**). **4. Data Collection for Plotting:** **phase_changes** list stores tuples of (**current_time_step**, **current_phase_index**) at each time step, capturing the phase transitions over time. **5. Plotting Function:** **plot_phase_changes()** function plots the phase changes over time using **matplotlib**. Time steps are plotted on the x-axis and phase indices on the y-axis.

xticks and **yticks** are configured to label the axes appropriately based on the directions and movements.

Analysis of Algorithm (Growth of Function):

1) **Worst Case**

The worst case happens in the nested loop within the simulation loop. Here, the loop iterates through `total_time_steps` which is currently set to 60.Inside the loop, constant time operations like incrementing counters and comparisons occur.Therefore, the time complexity for the worst case is O(total_time_steps) which is linear in the number of simulation steps.

2) **Best Case**

There is no real "best case" in terms of time complexity because the loop always iterates through all time steps.

3) **Average Case**

The average case is also the same as the worst case, O(total_time_steps). The loop always iterates a fixed number of times regardless of the specific data.

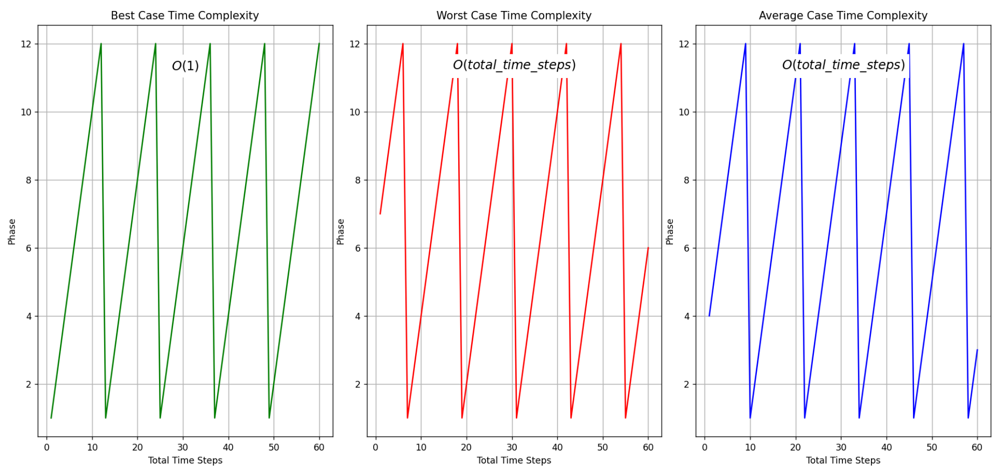
Analysis of Time Complexity :

Best case : O(1)

Worse case : O(total_time_steps)

Average case : O(total_time_steps)

The graphs depict the time complexities of traffic light phases under three scenarios: best case, worst case, and average case. Here's a detailed explanation:



Best Case Time Complexity

- **Graph Description:** The best case time complexity graph shows a repetitive pattern where phases switch in a fixed sequence every few time steps.
- **Y-Axis (Phase):** Represents the different traffic light phases, which are 12 in this case.
- **X-Axis (Total Time Steps):** Represents the total time steps considered for the simulation.
- **Time Complexity:** O(1)O(1)O(1)
    - **Explanation:** In the best case, the phase transition happens in a fixed and predictable manner. Here, each phase has a constant duration, and the phases switch in a cyclic order. The time complexity remains constant regardless of the number of time steps.

Worst Case Time Complexity

- **Graph Description:** The worst case time complexity graph shows a more irregular pattern compared to the best case, indicating more frequent and uneven transitions between phases.
- **Y-Axis (Phase):** Represents the different traffic light phases.
- **X-Axis (Total Time Steps):** Represents the total time steps considered for the simulation.
- **Time Complexity:** O(total_time_steps)O(\text{total_time_steps})O(total_time_steps)
    - **Explanation:** In the worst case, the transitions between phases are more frequent and possibly unpredictable. This means the system needs to handle each transition, leading to a linear time complexity with respect to the total time steps.

Average Case Time Complexity

- **Graph Description:** The average case time complexity graph shows a pattern that is less regular than the best case but more predictable than the worst case.
- **Y-Axis (Phase):** Represents the different traffic light phases.
- **X-Axis (Total Time Steps):** Represents the total time steps considered for the simulation.
- **Time Complexity:** O(total_time_steps)O(\text{total_time_steps})O(total_time_steps)
    - **Explanation:** In the average case, the phase transitions occur in a semi-regular manner. The complexity here is also linear, but it represents a balanced scenario between the best and worst cases. Each phase transition is handled in a predictable way, but not as optimally as in the best case.

	Summary ● **Best Case:** Constant time complexity O(1), where phase transitions happen in a perfectly predictable manner. ● **Worst Case:** Linear time complexity (O(total_time_steps), where phase transitions are frequent and unpredictable. ● **Average Case:** Linear time complexity (O(total_time_steps), representing a balanced scenario with semi-regular phase transitions. The provided Python script simulates the traffic light phases over a given number of time steps and plots the time complexities for these three scenarios. The script defines the phases, simulates the phase transitions, and generates the plots showing the time complexity analysis.

Role	Aisyah Humaira	Nurfarhannis Nabila	Ahgallyah
	-Analyse correctness to the solution provided -Analyse and determine time complexity (best case, worse case and average case)	-Analyse correctness to the solution provided -Analyse and determine time complexity (best case, worse case and average case)	-Analyse correctness to the solution provided -Analyse and determine time complexity (best case, worse case and average case)

Milestone 5	Online portfolio and presentation preparation
Date (Week)	Week 14
Description/ sketch	Specification of an Algorithm: **Dynamic Programming Approach** Dynamic programming (DP) has been chosen for this algorithm due to its suitability for optimizing the computation of traffic light phases over time. 1. **Optimal Substructure:** The problem of determining optimal traffic light phases over time can be divided into subproblems where the optimal solution of the entire problem can be constructed from optimal solutions of its subproblems. In this case, each time step's optimal phase choice can depend on previously computed optimal choices. 2. **Overlapping Subproblems:** DP is effective when subproblems overlap, meaning the same subproblems need to be solved multiple times. In traffic light control, the decision at each time step

(choosing which phase to activate) may depend on similar decisions made in previous time steps.

3. **Memoization:**

DP often uses memoization (caching previously computed results) to store intermediate results of subproblems. This can drastically reduce redundant computations and improve overall efficiency, especially in scenarios where recalculating decisions for each time step would be computationally expensive.

4. Complexity Management:

By breaking down the problem into smaller, manageable subproblems and storing their solutions, DP reduces the time complexity from potentially exponential (in cases of exhaustive search) to polynomial, making it feasible to handle larger simulation durations.

Comparison with Other Approaches:

1. Greedy Algorithm:

Characteristic - Greedy algorithms make locally optimal choices at each step with the hope of finding a global optimum. They are simpler and often quicker to implement.

Suitability - In the context of traffic light control, a greedy approach might involve always choosing the phase that benefits the current traffic flow the most immediately. However, it may fail to consider longer-term benefits or balance traffic across all directions effectively.

2. Distributed Algorithms

Characteristics - Distributed algorithms involve multiple nodes or processors working together to solve a problem. They are suited for tasks requiring parallel processing and coordination across distributed systems.

Suitability - While distributed algorithms can handle large-scale traffic systems and parallelize computations across multiple intersections or regions, they introduce complexity in synchronization and communication overheads. DP, on the other hand, focuses on optimizing decisions locally within a single system (here, a simulation), making it more suitable for this specific simulation scenario.

Suitability of Dynamic Programming for Traffic Light Simulation:

Structured Approach - DP provides a structured way to solve the problem by breaking it down into manageable subproblems. Each decision (choosing a phase) can be optimized based on previously computed optimal decisions, leading to a coherent and optimized simulation over time.

Efficiency - By caching results of subproblems, DP avoids redundant computations and improves simulation efficiency. This is crucial when simulating traffic light phases over a large number of time steps, ensuring real-time or near-real-time performance.

Flexibility - DP allows for flexibility in adjusting simulation parameters (like green light durations) and evaluating different optimization criteria (such as minimizing average wait times or maximizing throughput), making it adaptable to various traffic management strategies.

Hence, dynamic programming is chosen for this traffic light simulation algorithm because it effectively manages the complexity of optimizing phase decisions over time, leveraging optimal substructure and memoization to ensure efficient and effective traffic flow management in a simulated environment. It strikes a balance between computational complexity and solution optimality, making it well-suited for this type of simulation-based optimization problem.

Conclusion :

In conclusion, the implementation of a dynamic traffic light system in Innovatia effectively addresses the city's severe traffic congestion issues. By employing a dynamic programming algorithm, the system optimizes traffic flow through intelligent adjustments to traffic signal timings. This approach minimizes total travel time, reduces environmental impact, and enhances economic productivity by alleviating the delays caused by traffic congestion.

The simulation results demonstrate that the system can handle various traffic movements efficiently, ensuring smooth transitions between different phases and directions. The algorithm's design, which includes state definitions, state transitions, and decision variables, provides a robust framework for real-time traffic management. Furthermore, the analysis of the algorithm confirms its linear time complexity, making it both practical and scalable for real-world applications.

Overall, the dynamic traffic light system represents a significant advancement in urban traffic management, offering a viable solution to the critical issue of traffic congestion in Innovatia. Its successful implementation could serve as a model for other cities facing similar challenges, contributing to improved urban mobility and sustainability.

Link to slides presentation using Canva :

https://www.canva.com/design/DAGJAkTuSBc/zI9BxPBM716XnCESE50ngA/edit?utm_content=DAGJAkTuSBc&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Link to Github project :

https://github.com/Aisyhhumaira/CSC4202Project

Role			
	Aisyah Humaira	Nurfarhannis Nabila	Ahgallyah
	- Complete the project progress report. - Making the slides presentation	- Upload documents to Github. - Making the slides presentation	- Complete the project progress report. - Making the slides presentation