VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"JNANA SANGAMA", BELAGAVI -590 018.

2024 – 2025

A Data Structure and its Applicatins (BCS304) Mini-Project Report
on

"EV Queue Management System"

Submitted in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in

CSE (AI&ML)

Submitted By

Akshith Gowda D	[1RN23CI016]
Darshan B	[1RN23CI046]
Charan Raj V	[1RN23CI042]
Aditya Ashok Patil	[1RN23CI009]

Under the guidance of
Dr. Shruthi U
Assistant Professor, Dept. of CSE (AI&ML)

Estd : 2001

DEPARTMENT OF CSE (AI&ML)

RNS INSTITUTE OF TECHNOLOGY
Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE

(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)

Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098

DEPARTMENT OF CSE (AI&ML)

Estd : 2001

CERTIFICATE

Certified that the mini-project work entitled ***"EV Queue Management System"*** carried out by Akshith Gowda [1RN23CI016], Darshan B [1RN23CI046], Charan Raj V[1RN23CI042], Aditya Ashok Patil [1RN23CI009] are bonafide students of **RNS Institute of Technology** in partial fulfilment for the award of **"BACHELOR OF ENGINEERING"** in **CSE (AI&ML)** as prescribed by **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during the academic year **2024 – 25**. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the departmental library. The mini-project report has been approved as it satisfies the academic requirements in respect of mini-project work on Data Structure and its Applications (BCS304) prescribed for the said degree.

Signature of Course Coordinator

Dept. of CSE(AI&ML), RNSIT

Abstract

The **Vehicle Management System** is an innovative solution designed to enhance the organization and management of vehicles in diverse contexts, such as parking facilities, electric vehicle (EV) charging stations, and fleet operations. This system leverages advanced data structures, including **MinHeap**, **Priority Queue**, and **Linked List Queue**, to classify and manage vehicles based on urgency, type, and battery status. Emergency and VIP vehicles are processed using the Priority Queue, ensuring swift attention to critical cases. Regular vehicles with sufficient battery levels are organized in a Linked List Queue for orderly handling, while vehicles with low battery levels are managed using a MinHeap, prioritizing their charging needs efficiently.

The system offers features for vehicle addition, categorization, display, and removal, enabling precise control and real-time tracking through unique vehicle identifiers. By integrating structured algorithms, it optimizes resource allocation and minimizes delays, particularly in high-demand environments. Key advantages include enhanced response times for emergencies, streamlined workflows, and the ability to handle large volumes of data without performance degradation. Additionally, its modular design ensures scalability, allowing seamless adaptation to growing demands or integration with other systems like automated parking sensors or IoT devices.

The Vehicle Management System represents a step forward in smart transportation management, enabling operators to improve operational efficiency, enhance customer satisfaction, and maximize resource utilization, all while reducing human error. This comprehensive approach aligns with the modern needs of urban planning and smart city initiatives, offering significant value across industries.

Table of Contents

CHAPTER 1

INTRODUCTION

EV Queue Management System:

Data structures are essential tools for organizing, managing, and storing data in a way that enables efficient access and modification. They are foundational concepts in computer science, allowing algorithms to run efficiently, especially when dealing with large amounts of data. Each data structure provides a different way of organizing data, depending on the requirements of the algorithm and the nature of the problem being solved. The primary goal of choosing the right data structure is to optimize operations like search, insertion, deletion, and traversal. Data structures can be broadly categorized into linear and non-linear structures. Linear data structures, like arrays, linked lists, stacks, and queues, store elements in a sequential order. Non-linear structures, like trees and graphs, store elements in hierarchical or interconnected arrangements. Depending on the task at hand, the correct choice of data structure can significantly impact the performance of an application.

This program is a **Vehicle Management System** that handles vehicles based on priority and specific conditions using three distinct data structures: **MinHeap**, **PriorityQueue**, and **LinkedListQueue**. It categorizes vehicles into:

1. **Low Battery Vehicles** (handled by MinHeap).
2. **VIP/Emergency Vehicles** (handled by PriorityQueue).
3. **Regular Vehicles** (handled by LinkedListQueue).

The system allows the addition, display, and removal of vehicles based on predefined rules such as battery level, type (VIP, emergency, regular), and arrival time. The goal is to simulate an efficient queue management system for a parking lot or vehicle service area, ensuring priority handling and quick processing.

QUEUE:

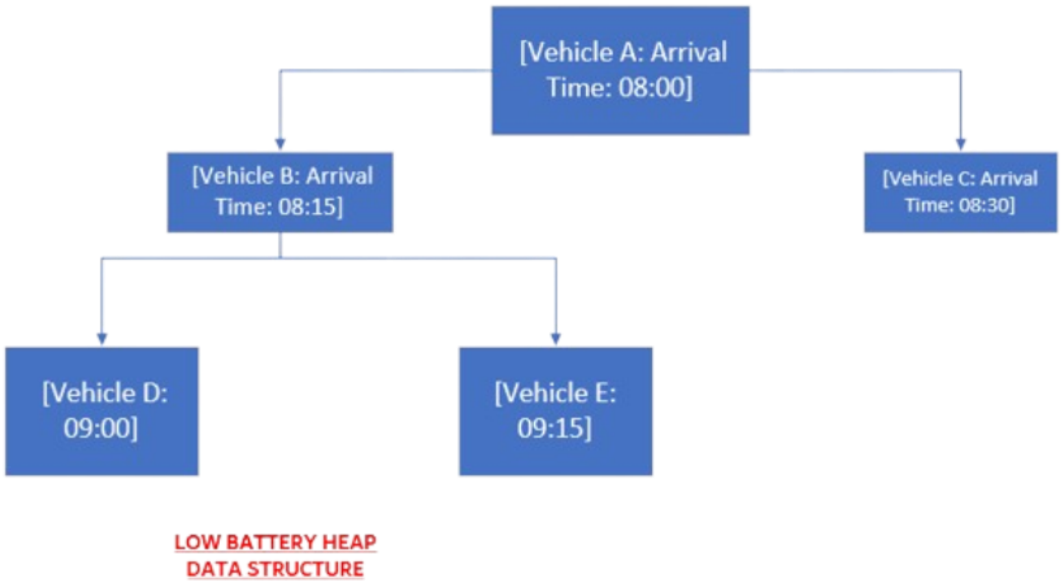
- **MinHeap**: A MinHeap is used to manage vehicles that need immediate attention due to low battery levels. The heap structure ensures that the vehicle with the lowest battery (or earliest arrival time) is processed first. The MinHeap maintains the property that each parent node has a smaller value than its children, making it efficient for priority-based processing.
- **PriorityQueue**: The PriorityQueue is used to manage vehicles that have a higher priority, such as VIP or emergency vehicles. In this system, emergency vehicles have the highest priority, followed by VIP vehicles. The priority queue ensures that the vehicle with the highest priority (smallest priority number) is always processed first.
- **LinkedListQueue**: A standard queue is used to manage regular vehicles. The queue operates on a first-in-first-out (FIFO) basis, meaning the first vehicle to arrive is the first one to be processed. This data structure is ideal for situations where the order of arrival needs to be maintained.

The project focuses on creating a **Vehicle Management System** that leverages various data structures to handle different types of vehicles in a parking or processing system. The goal is to organize the vehicles based on their attributes, such as arrival time, type, and battery level, using data structures like MinHeap, PriorityQueue, and LinkedListQueue. This system ensures that vehicles are processed efficiently, whether they are regular vehicles, VIP vehicles, or emergency vehicles.

CHAPTER 2

PROJECT DESIGN

Low Battery Vehicles – Min Heap Implementation :


[Vehicle A: Arrival Time: 08:00]
[Vehicle B: Arrival Time: 08:15]
[Vehicle C: Arrival Time: 08:30]
[Vehicle D: 09:00]
[Vehicle E: 09:15]
LOW BATTERY HEAP DATA STRUCTURE

The MinHeap is a data structure designed to efficiently organize and prioritize elements, ensuring the smallest element (based on arrival time) is always at the root. It is particularly useful in managing low-battery vehicles, ensuring those arriving earlier are processed first.

Key Concepts

1. **Heap Property**:
   - The parent node's value is always smaller than or equal to its child nodes.
   - The smallest element is at the root.
2. **Array Representation**:
   - **Parent**: (i−1)÷2(i-1) \div 2(i−1)÷2
   - **Left Child**: 2×i+12 \times i + 12×i+1
   - **Right Child**: 2×i+22 \times i + 22×i+2

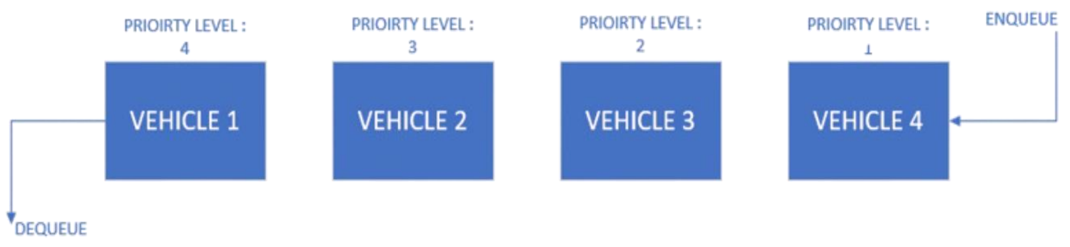
Functions

1. **__init__()**
   - **Purpose**: Initializes an empty heap.
   - **Use**: Creates the base structure to store vehicles.
2. **Helper Methods**:
   - **_parent(index)**: Returns the index of the parent node.
   - **_left_child(index)**: Returns the index of the left child.
   - **_right_child(index)**: Returns the index of the right child.
   - **_swap(i, j)**: Swaps elements at indices i and j.
3. **_heapify_up(index)**
   - **Purpose**: Restores the heap property after inserting a new element.
   - **Process**: Compares the new element with its parent and swaps if necessary, continuing upward until the heap property is restored.
4. **_heapify_down(index)**
   - **Purpose**: Restores the heap property after removing/replacing the root.
   - **Process**: Compares the current element with its children and swaps with the smallest child if necessary, continuing downward until the heap property is restored.
5. **insert(vehicle)**
   - **Purpose**: Adds a new vehicle to the heap.
   - **Process**: Appends the vehicle and uses _heapify_up to maintain the heap property.
   - **Use Case**: A new low-battery vehicle arrives.
6. **pop()**
   - **Purpose**: Removes and returns the vehicle with the earliest arrival time (root).
   - **Process**: Replaces the root with the last element, removes the last element, and uses _heapify_down to restore the heap property.
   - **Use Case**: Serve the next vehicle in line.
7. **is_empty()**
   - **Purpose**: Checks if the heap is empty.
   - **Use Case**: Prevents operations like pop() on an empty heap.
8. **display()**
   - **Purpose**: Displays all vehicles in the heap.
   - **Use Case**: Monitor the current state of vehicles awaiting service.

Advantages

1. **Efficiency**: Insertions and deletions have a time complexity of O(log n)O(\log n)O(logn).
2. **Fairness**: Prioritizes vehicles with the smallest arrival time.
3. **Scalability**: Handles large numbers of vehicles without significant performance degradation.

Emergency/VIP Vehicles – Priority Queue Implementation :



PRIORITY QUEUE DATA STRUCTURE

The PriorityQueue is a specialized data structure that serves elements based on their priority, ensuring vehicles with the highest urgency are processed first. Emergency vehicles have the highest priority (priority = 1), followed by VIP vehicles (priority = 2).

Key Concepts

1. **Priority Assignment**:
   - Each vehicle is assigned a priority value. Lower values indicate higher priority.
   - Vehicles with the highest priority are dequeued first.
2. **Heap-Based Implementation**:
   - Internally implemented as a binary heap for efficient operations.
   - Similar to a MinHeap but prioritizes based on the priority values.

Functions

1. **__init__()**
   - **Purpose**: Initializes an empty PriorityQueue.

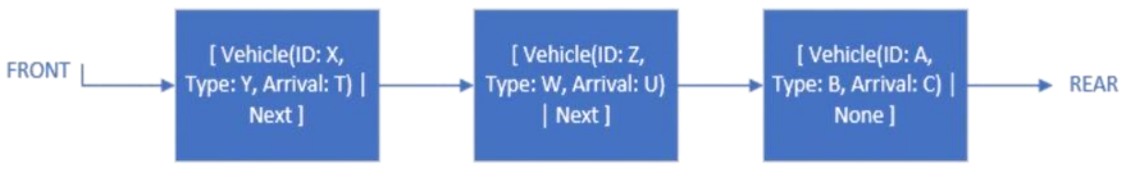
   - **Process**: Creates a list to store tuples of priority and vehicle objects.
2. **p_add_vehicle(vehicle, priority)**
   - **Purpose**: Adds a vehicle with an assigned priority.
   - **Process**:
     - Appends the tuple (priority, vehicle) to the queue.
     - Calls _bubble_up to reorder the queue and maintain priority.
   - **Use Case**: Adding a new emergency or VIP vehicle to the system.
3. **_bubble_up(index)**
   - **Purpose**: Restores the heap property after a new element is added.
   - **Process**:
     - Compares the priority of the current element with its parent.
     - Swaps if the current element has a higher priority (lower number).
     - Repeats until the heap property is restored.
   - **Example**: Moving an emergency vehicle above VIP vehicles.
4. **p_remove_vehicle()**
   - **Purpose**: Removes and returns the vehicle with the highest priority.
   - **Process**:
     - Replaces the root with the last element in the queue.
     - Removes the last element from the list.
     - Calls _bubble_down to restore the heap property.
     - If empty, returns None.
   - **Use Case**: Processing an emergency vehicle.
5. **p_bubble_down(index)**
   - **Purpose**: Adjusts the queue after removing the root element.
   - **Process**:
     - Compares the current element with its children.
     - Swaps with the child having the highest priority (lowest value).
     - Repeats until the heap property is restored.
   - **Example**: Moving a VIP vehicle to the root after an emergency vehicle is removed.
6. **p_display()**
   - **Purpose**: Displays all vehicles in the queue with their priorities.
   - **Process**: Iterates through the list and prints the priority and vehicle details.

   - **Use Case**: Monitoring the state of the PriorityQueue.
7. **p_is_empty()**
   - **Purpose**: Checks if the queue is empty.
   - **Process**: Returns True if the queue is empty, otherwise False.
   - **Use Case**: Ensures operations like p_remove_vehicle() are valid.

Advantages

1. **Efficient Prioritization**: Ensures critical vehicles (e.g., emergency) are served first.
2. **Heap Optimization**: Insertion and removal operations are efficient with O(log⁡n)O(\log n)O(logn) complexity.
3. **Fairness and Order**: Guarantees urgent vehicles are processed before others.

Regular Vehicles - LinkedListQueue Representation :


FRONT
[Vehicle(ID: X, Type: Y, Arrival: T) | Next]
[Vehicle(ID: Z, Type: W, Arrival: U) | Next]
[Vehicle(ID: A, Type: B, Arrival: C) | None]
REAR

LINKEDLIST QUEUE DATA STRUCTURE

A **LinkedListQueue** is a queue implemented using a **singly linked list**. In this structure, each node contains data and a pointer to the next node. The queue operates on the **FIFO (First In, First Out)** principle, where the first element added is the first to be removed.

Purpose of LinkedListQueue in the Program

In the vehicle management system, the **LinkedListQueue** is used to handle **regular vehicles**. This ensures that vehicles without urgent requirements are processed in the order they arrive.

Key Concepts in LinkedListQueue

1. **Node Structure**:
    - Each node contains:
        - data: Stores the vehicle object.
        - next: Points to the next node in the list.
    - The linked list allows dynamic memory allocation, making it efficient for queues that grow and shrink frequently.
2. **Pointers**:
    - front: Points to the first node in the queue.
    - rear: Points to the last node in the queue.
3. **Operations**:
    - **Enqueue**: Add a new element at the end of the queue.
    - **Dequeue**: Remove the element at the front of the queue.
    - **Display**: Traverse the list and print all elements.

LinkedListQueue Functions

1. __init__

- **Purpose**: Initializes an empty queue.
- **Explanation**:
    - Sets both front and rear pointers to None.
    - Indicates that the queue is initially empty.

---

2. lqenqueue(item)

- **Purpose**: Adds a new vehicle to the end of the queue.
- **Explanation**:
    - Creates a new Node with the given item (vehicle).
    - If the queue is empty (rear is None):
        - Sets both front and rear to the new node.
    - If the queue is not empty:
        - Sets the next pointer of the current rear to the new node.
        - Updates the rear pointer to the new node.

UseCase:

A regular vehicle arrives and needs to be added to the queue.

---

3. lqdequeue()

- **Purpose**: Removes and returns the vehicle at the front of the queue.
- **Explanation**:
  - If the queue is empty (front is None), returns "Queue is empty".
  - Temporarily stores the current front node.
  - Moves the front pointer to the next node in the list.
  - If the queue becomes empty (front becomes None), sets rear to None.
  - Returns the data of the removed node.

Use Case: The first vehicle in the regular queue is processed and removed.

---

4. lqis_empty()

- **Purpose**: Checks if the queue is empty.
- **Explanation**:
  - Returns True if front is None; otherwise, returns False.
  - Prevents operations like lqdequeue() or lqpeek() on an empty queue.

Use Case: Before removing or peeking, this function ensures the queue has elements to operate on.

---

5. lqpeek()

- **Purpose**: Returns the vehicle at the front of the queue without removing it.
- **Explanation**:
  - If the queue is empty (front is None), returns "Queue is empty".
  - Otherwise, returns the data of the front node.

Use Case: To check which vehicle is next to be processed without modifying the queue.

---

6. lqdisplay()

- **Purpose**: Displays all vehicles in the queue.
- **Explanation**:
  - Traverses the linked list starting from the front.
  - Collects the data from each node into a list for display.
  - Prints the contents of the queue.

Use Case: To view all regular vehicles currently waiting in the queue.

Advantages of LinkedListQueue in the Program

1. **Dynamic Size**: The queue grows and shrinks dynamically, without the need for resizing.
2. **Efficient Memory Use**: Memory is allocated only for nodes that contain data.
3. **FIFO Order**: Ensures vehicles are processed in the order they arrive.

CHAPTER 3

IMPLEMENTATION

Main Code:

```
from datetime import datetime
from priorq import PriorityQueue
from queue_1 import LinkedListQueue
from minheap import MinHeap

class Vehicle:
    def __init__(self, vehicle_id, vehicle_type, arrival_time):
        self.vehicle_id = vehicle_id
        self.vehicle_type = vehicle_type
        self.arrival_time = arrival_time

    def __lt__(self, other):
        return self.arrival_time < other.arrival_time

    def __repr__(self):
        return f"Vehicle(ID: {self.vehicle_id}, Type: {self.vehicle_type}, Arrival:
{self.arrival_time})"

def main():
    min_heap = MinHeap()
```

```
priority_queue = PriorityQueue()
linked_list_queue = LinkedListQueue()

while True:
    print("\n--- Vehicle Management System ---")
    print("1. Add a vehicle")
    print("2. Display all vehicles in MinHeap (Low Battery)")
    print("3. Display all vehicles in PriorityQueue (VIP/Emergency)")
    print("4. Display all vehicles in LinkedListQueue (Regular)")
    print("5. Remove a vehicle by ID")
    print("6. Exit")

    choice = input("Enter your choice: ").strip()

    if choice == "1":
        vehicle_id = input("Enter Vehicle ID (NUMBER PLATE ID): ")

        valid_types = ["regular", "vip", "emergency"]
        while True:
            vehicle_type = input("Enter Vehicle Type (Regular, VIP, Emergency): ").strip().lower()
            if vehicle_type in valid_types:
                break
            print("Invalid vehicle type. Please enter Regular, VIP, or Emergency.")

        while True:
            arrival_time = input("Enter Arrival Time (HH:MM format): ")
            try:
                arrival_time = datetime.strptime(arrival_time, "%H:%M")
                break
            except ValueError:
                print("Invalid time format. Please use HH:MM.")

        while True:
            try:
                battery_level = int(input("Enter Battery Level (in percentage [0-100]): "))
                if 0 <= battery_level <= 100:
                    break
                else:
                    print("Battery level must be between 0 and 100.")
            except ValueError:
                print("Invalid input. Please enter a number between 0 and 100.")

        vehicle = Vehicle(vehicle_id, vehicle_type, arrival_time)

        if vehicle_type in ["vip", "emergency"]:
            priority = 1 if vehicle_type == "emergency" else 2
            priority_queue.p_add_vehicle(vehicle, priority)
            print(f"Vehicle {vehicle_id} added to PriorityQueue.")
        elif vehicle_type == "regular" and battery_level > 40:
            linked_list_queue.lqenqueue(vehicle)
            print(f"Vehicle {vehicle_id} added to LinkedListQueue.")
        elif battery_level <= 40:
            min_heap.insert(vehicle)
            print(f"Vehicle {vehicle_id} added to MinHeap.")
        else:
            print("Invalid vehicle type or conditions. Vehicle not added.")

    elif choice == "2":
        print("\n--- MinHeap: Vehicles with Low Battery ---")
        min_heap.display()

    elif choice == "3":
        print("\n--- PriorityQueue: VIP/Emergency Vehicles ---")
        priority_queue.p_display()

    elif choice == "4":
        print("\n--- LinkedListQueue: Regular Vehicles ---")
        linked_list_queue.lqdisplay()

    elif choice == "5":
        vehicle_id = input("Enter Vehicle ID to remove: ")
        removed_from_min_heap = min_heap.remove_by_id(vehicle_id)
        if removed_from_min_heap:
            print(f"Vehicle {vehicle_id} removed from MinHeap.")
        else:
            removed_from_priority_queue = priority_queue.remove_by_id(vehicle_id)
            if removed_from_priority_queue:
                print(f"Vehicle {vehicle_id} removed from PriorityQueue.")
            else:
                removed_from_linked_list = linked_list_queue.remove_by_id(vehicle_id)
                if removed_from_linked_list:
                    print(f"Vehicle {vehicle_id} removed from LinkedListQueue.")
                else:
                    print(f"Vehicle {vehicle_id} not found in any queue.")

    elif choice == "6":
        print("Exiting the system.")
        break
    else:
        print("Invalid choice. Please try again.")

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

Min Heap - Low Battery Vehicles :

```
class Vehicle:
    def __init__(self, vehicle_id, vehicle_type, arrival_time):
        self.vehicle_id = vehicle_id
        self.vehicle_type = vehicle_type
        self.arrival_time = arrival_time

    def __lt__(self, other):
        return self.arrival_time < other.arrival_time

    def __repr__(self):
        return f"Vehicle(ID: {self.vehicle_id}, Type: {self.vehicle_type}, Arrival:
{self.arrival_time})"

class MinHeap:
    def __init__(self):
        self.heap = []

    def _parent(self, index):
        return (index - 1) // 2

    def _left_child(self, index):
        return 2 * index + 1

    def _right_child(self, index):
        return 2 * index + 2

    def _swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]

    def _heapify_up(self, index):
        parent = self._parent(index)
        if index > 0 and self.heap[index] < self.heap[parent]:
            self._swap(index, parent)
            self._heapify_up(parent)

    def _heapify_down(self, index):
        smallest = index
        left = self._left_child(index)
        right = self._right_child(index)

        if left < len(self.heap) and self.heap[left] < self.heap[smallest]:
            smallest = left
        if right < len(self.heap) and self.heap[right] < self.heap[smallest]:
            smallest = right

        if smallest != index:
            self._swap(index, smallest)
            self._heapify_down(smallest)

    def insert(self, vehicle):
        self.heap.append(vehicle)
        self._heapify_up(len(self.heap) - 1)

    def pop(self):
        if self.is_empty():
            print("Heap is empty")
            return None
        if len(self.heap) == 1:
            return self.heap.pop()

        root = self.heap[0]
        self.heap[0] = self.heap.pop()
        self._heapify_down(0)
        return root
    def is_empty(self):
        return len(self.heap) == 0

    def display(self):
        if self.is_empty():
            print("Heap is empty")
            return
        for vehicle in self.heap:
            print(vehicle)
```

Priority Queue – Emergency/VIP Vehicles :

```
class PriorityQueue:
    def __init__(self):
        self.queue = []

    def p_add_vehicle(self, vehicle, priority):
        self.queue.append((priority, vehicle))
        self._bubble_up(len(self.queue) - 1)

    def _bubble_up(self, index):
        parent = (index - 1) // 2
        while index > 0 and self.queue[index][0] < self.queue[parent][0]:
            self.queue[index], self.queue[parent] = self.queue[parent], self.queue[index]
            index = parent
            parent = (index - 1) // 2

    def p_bubble_up(self, index):
        while index > 0:
            parent_index = (index - 1) // 2
            if self.queue[index][0] < self.queue[parent_index][0]:
                self.queue[index], self.queue[parent_index] = self.queue[parent_index],
self.queue[index]
                index = parent_index
            else:
                break

    def p_remove_vehicle(self):
        if not self.queue:
            return "No vehicles in the priority queue"

        highest_priority_vehicle = self.queue[0][1]
        self.queue[0] = self.queue[-1]
        self.queue.pop()
        self._bubble_down(0)
        return highest_priority_vehicle

    def p_bubble_down(self, index):
        length = len(self.queue)
        while index < length:
            left_child_index = 2 * index + 1
            right_child_index = 2 * index + 2
            smallest = index

            if left_child_index < length and self.queue[left_child_index][0] < self.queue[smallest][0]:
                smallest = left_child_index

            if right_child_index < length and self.queue[right_child_index][0] <
self.queue[smallest][0]:
                smallest = right_child_index
           if smallest != index:
                self.queue[index], self.queue[smallest] = self.queue[smallest], self.queue[index]
                index = smallest
            else:
                break
     def p_display(self):
        print("Priority Queue:")
        for priority, vehicle in self.queue:
            print(f"Priority {priority}: {vehicle}")
   def p_is_empty(self):
        return len(self.queue) == 0
```

```
if __name__ == "__main__":
    priority_queue = PriorityQueue()
    priority_queue.add_vehicle("Ambulance", 1)
    priority_queue.add_vehicle("Diplomat Car", 2)
    priority_queue.add_vehicle("Fire Truck", 1)
    priority_queue.add_vehicle("Regular Car", 5)

    priority_queue.display()
    print("Serving:", priority_queue.remove_vehicle())
    print("Serving:", priority_queue.remove_vehicle())
    priority_queue.display()
```

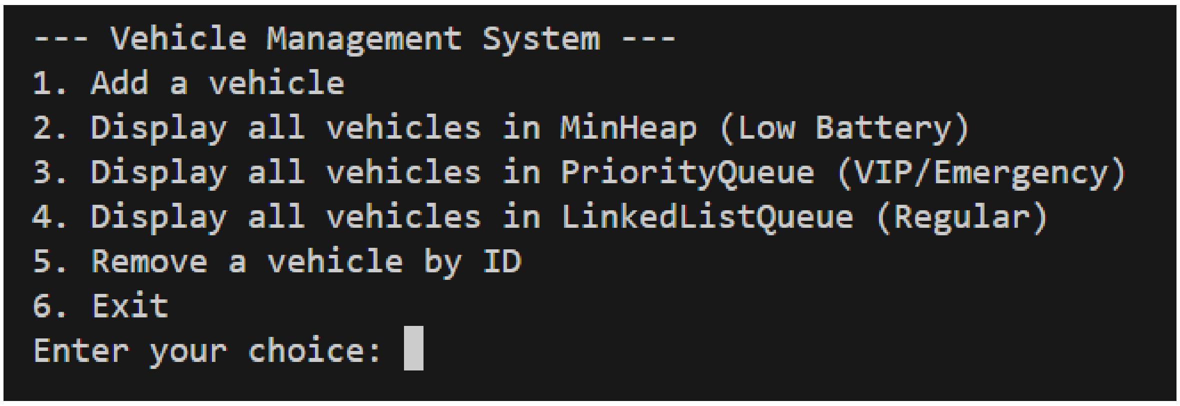
Linked List – Regular Vehicles :

```
class LinkedListQueue:
    def __init__(self):
        self.front = None
        self.rear = None

    def lqis_empty(self):
        return self.front is None

    def lqdisplay(self):
        if self.lqis_empty():
            print("Queue is empty.")
            return
        current = self.front
        while current:
            print(current.data)
            current = current.next


class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedListQueue:
    def __init__(self):
        self.front = None
        self.rear = None

    def lqenqueue(self, item):
        new_node = Node(item)
        if self.rear is None:
            self.front = self.rear = new_node
            return
        self.rear.next = new_node
        self.rear = new_node

    def lqdequeue(self):
        if self.front is None:
            return "Queue is empty"
        temp = self.front
        self.front = temp.next
        if self.front is None:
            self.rear = None
        return temp.data

    def lqis_empty(self):
        return self.front is None

    def lqpeek(self):
        if self.front is not None:
            return self.front.data
        return "Queue is empty"

    def lqdisplay(self):
        if self.lqis_empty():
            print("Queue is empty")
        else:
            current = self.front
            queue_list = []
            while current:
                queue_list.append(current.data)
                current = current.next
           print("Queue:", queue_list)
```

CHAPTER 4

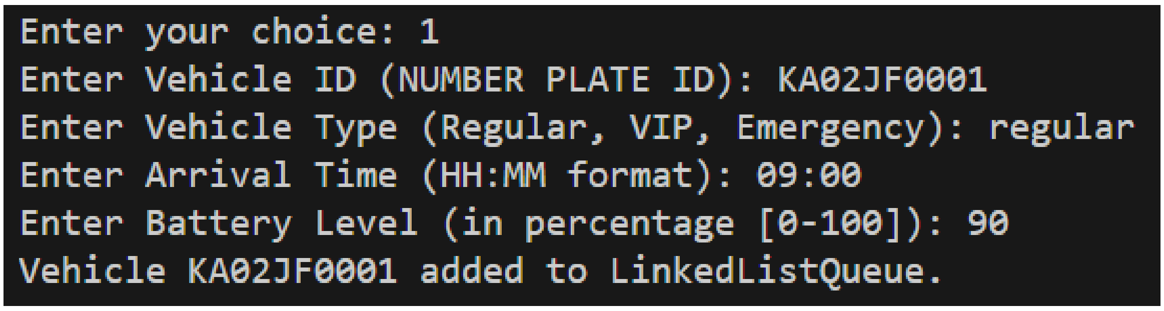
RESULTS

MENU:



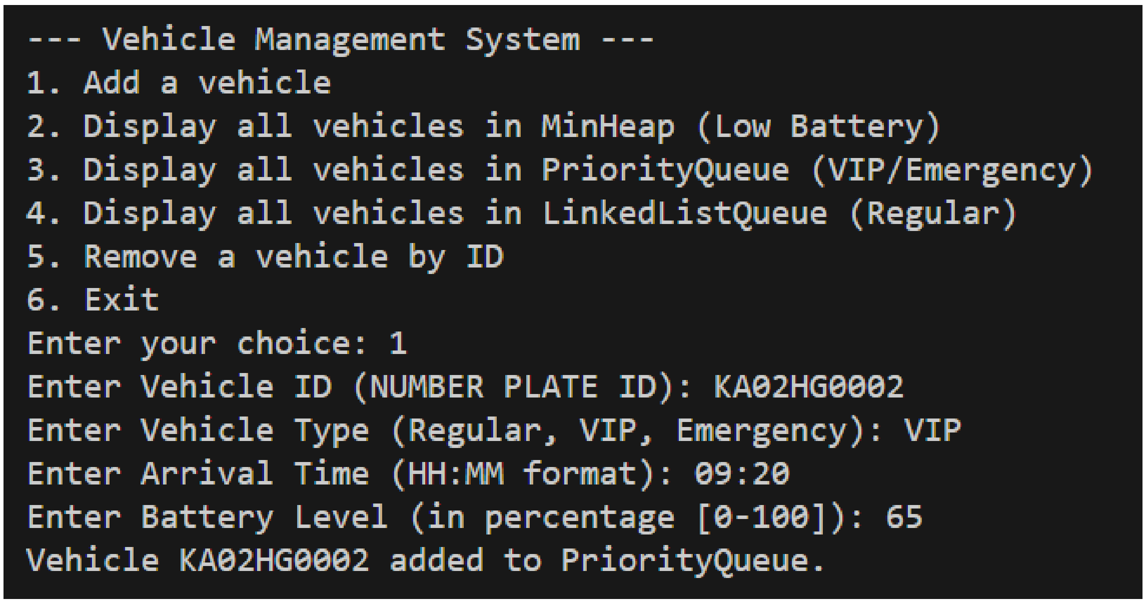
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 
```

Figure1: Menu displaying operations and accepting a choice



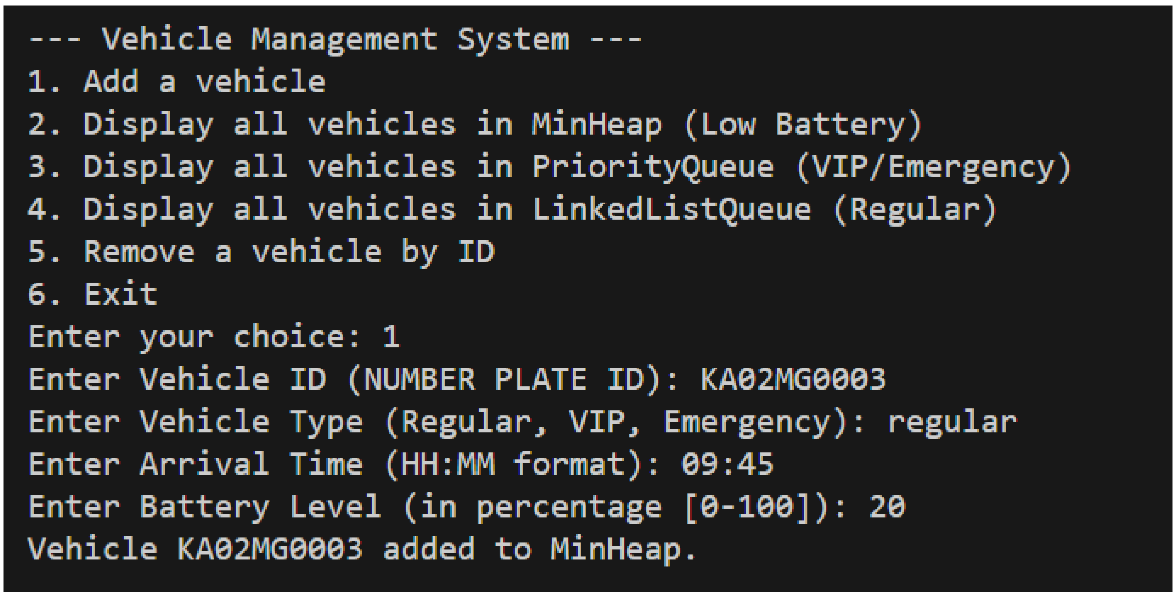
```
Enter your choice: 1
Enter Vehicle ID (NUMBER PLATE ID): KA02JF0001
Enter Vehicle Type (Regular, VIP, Emergency): regular
Enter Arrival Time (HH:MM format): 09:00
Enter Battery Level (in percentage [0-100]): 90
Vehicle KA02JF0001 added to LinkedListQueue.
```

Figure2: Adding a Regular Vehicle into the LinkedListqueue and also give it pertaining details of the vehicle which is the queue for regular vehicles



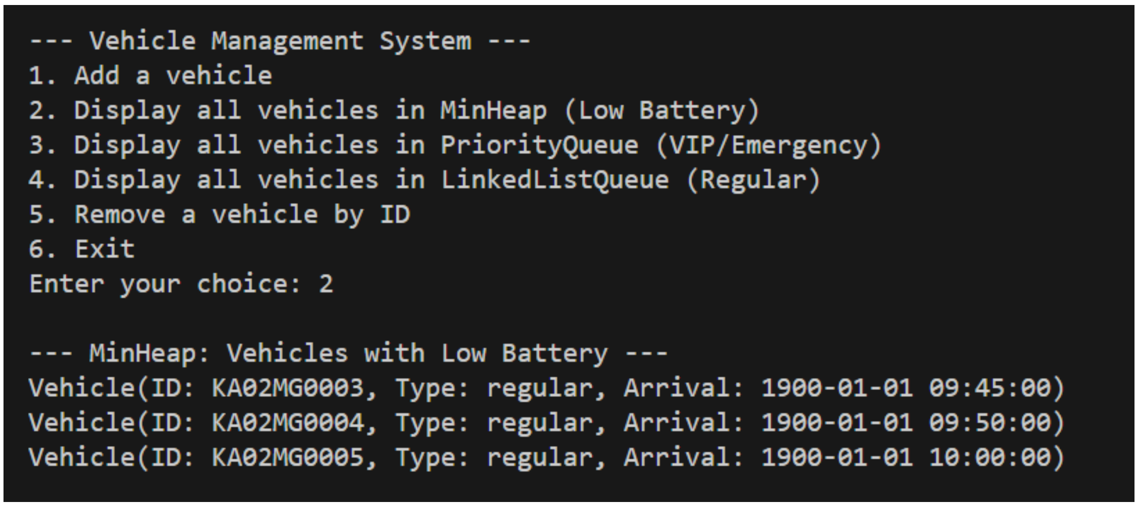
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 1
Enter Vehicle ID (NUMBER PLATE ID): KA02HG0002
Enter Vehicle Type (Regular, VIP, Emergency): VIP
Enter Arrival Time (HH:MM format): 09:20
Enter Battery Level (in percentage [0-100]): 65
Vehicle KA02HG0002 added to PriorityQueue.
```

Figure3: Adding VIP vehicle and giving its details into the VIP/EMERGENCY vehicles queue(Priority Queue Implementation)



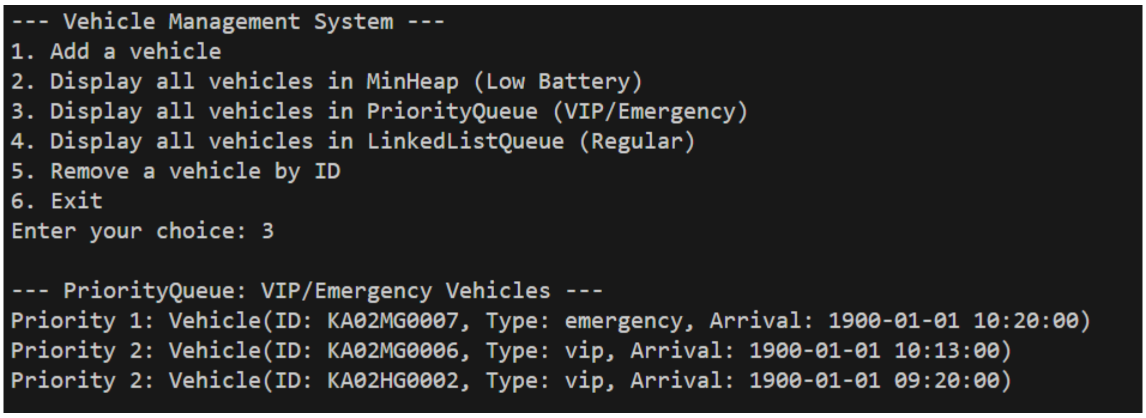
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 1
Enter Vehicle ID (NUMBER PLATE ID): KA02MG0003
Enter Vehicle Type (Regular, VIP, Emergency): regular
Enter Arrival Time (HH:MM format): 09:45
Enter Battery Level (in percentage [0-100]): 20
Vehicle KA02MG0003 added to MinHeap.
```

Figure4: Adding a Vehicle with low battery Into the Low battery queue (Min heap implementation)



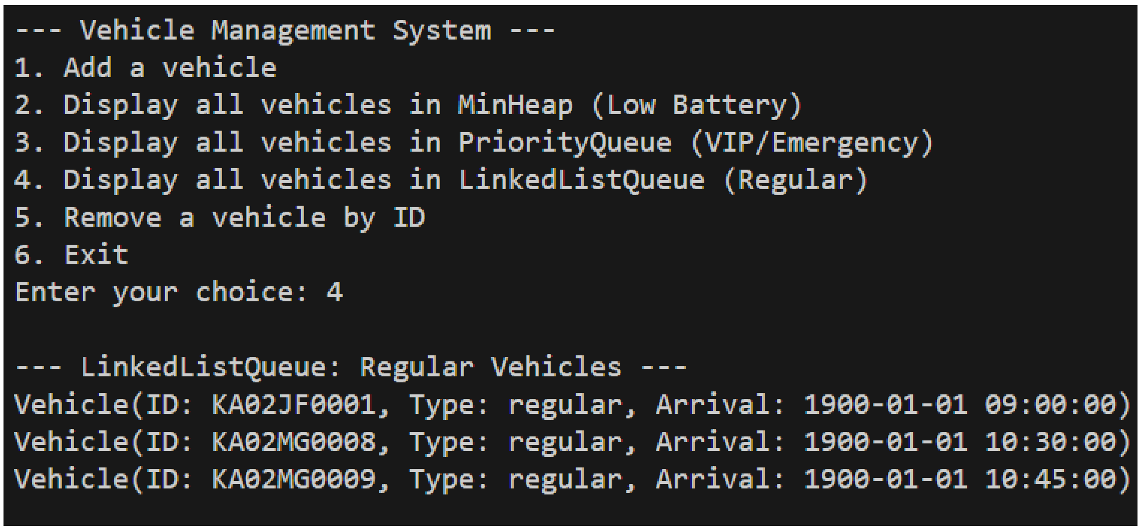
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 2

--- MinHeap: Vehicles with Low Battery ---
Vehicle(ID: KA02MG0003, Type: regular, Arrival: 1900-01-01 09:45:00)
Vehicle(ID: KA02MG0004, Type: regular, Arrival: 1900-01-01 09:50:00)
Vehicle(ID: KA02MG0005, Type: regular, Arrival: 1900-01-01 10:00:00)
```

Figure5:Displaying low battery vehicles which is stored in the Low battery queue (Minheap)



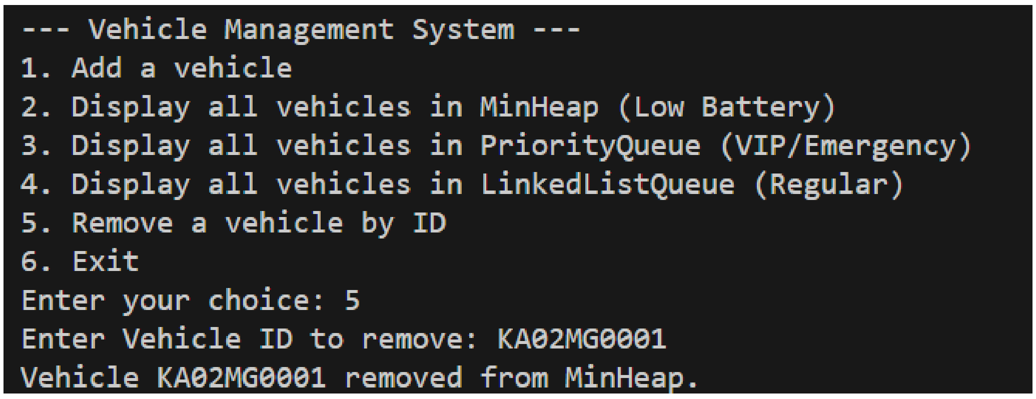
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 3

--- PriorityQueue: VIP/Emergency Vehicles ---
Priority 1: Vehicle(ID: KA02MG0007, Type: emergency, Arrival: 1900-01-01 10:20:00)
Priority 2: Vehicle(ID: KA02MG0006, Type: vip, Arrival: 1900-01-01 10:13:00)
Priority 2: Vehicle(ID: KA02HG0002, Type: vip, Arrival: 1900-01-01 09:20:00)
```

Figure6: Displaying VIP/EMERGENCY vehicles (priority queue) stored in VIP/EMERGENCY vehicle queue



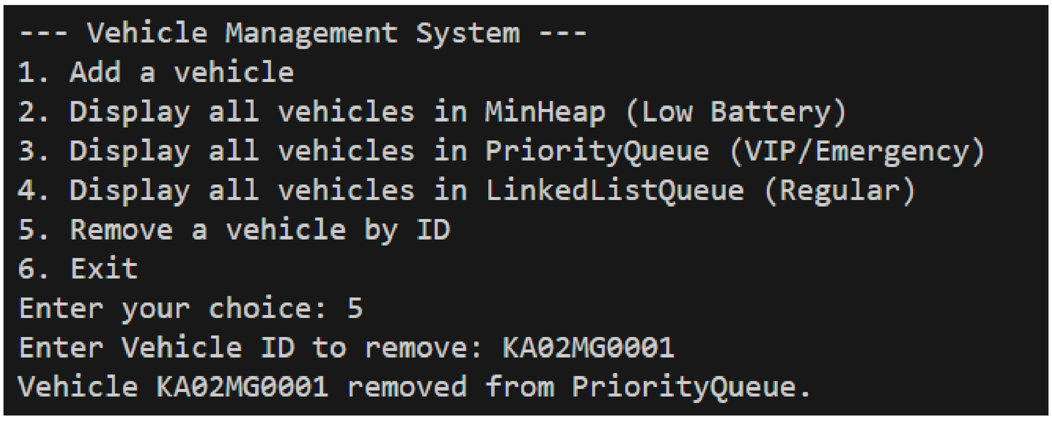
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 4

--- LinkedListQueue: Regular Vehicles ---
Vehicle(ID: KA02JF0001, Type: regular, Arrival: 1900-01-01 09:00:00)
Vehicle(ID: KA02MG0008, Type: regular, Arrival: 1900-01-01 10:30:00)
Vehicle(ID: KA02MG0009, Type: regular, Arrival: 1900-01-01 10:45:00)
```

Figure7:Displaying Regular Vehicles(Linked list queue) in the regular queue



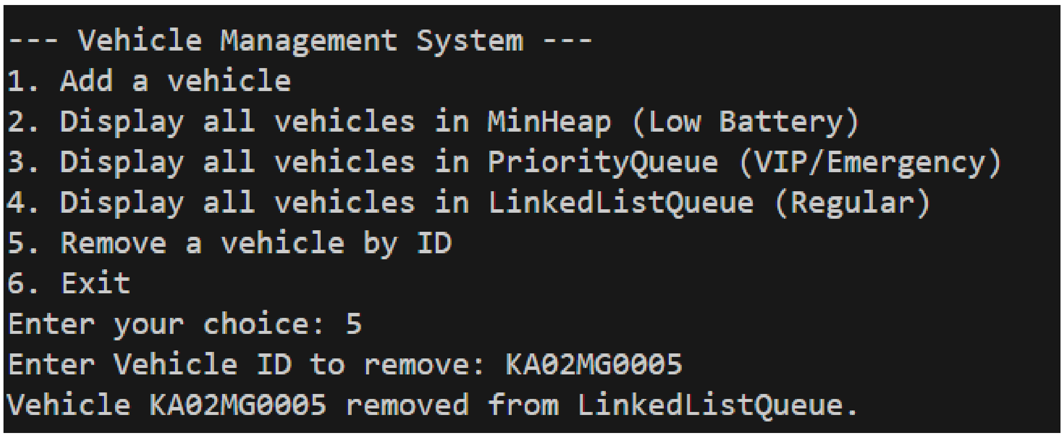
```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 5
Enter Vehicle ID to remove: KA02MG0001
Vehicle KA02MG0001 removed from MinHeap.
```

Figure8: Removing a Vehicle from the Low Battery Queue



```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 5
Enter Vehicle ID to remove: KA02MG0001
Vehicle KA02MG0001 removed from PriorityQueue.
```

Figure9: Removing a Vehicle from the Vip/Emergency Queue



```
--- Vehicle Management System ---
1. Add a vehicle
2. Display all vehicles in MinHeap (Low Battery)
3. Display all vehicles in PriorityQueue (VIP/Emergency)
4. Display all vehicles in LinkedListQueue (Regular)
5. Remove a vehicle by ID
6. Exit
Enter your choice: 5
Enter Vehicle ID to remove: KA02MG0005
Vehicle KA02MG0005 removed from LinkedListQueue.
```

Figure 10: Removing a Vehicle from the Regular Vehicles Queue

References

[1] https://www.geeksforgeeks.org/courses/Data-Structures-With-Python

[2] "Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles" by Narasimha Karumanchi

[3] https://www.w3schools.com/dsa/dsa_intro.php