CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY
Autonomous Institute | Affiliated to Osmania University

SMART PARKING SYSTEM

160123737120 Laksh Jain
160123737122 M. Vedaanth Singh
160123737125 Yogesh Choudary
IT Department, CBIT
Guide: P. Kiranmaei
Submission Date: 14-11-2024

Contents

Abstract

Parking congestion is one of the most common problems in any country in the world; it is due to the high demand made on parking that is not ample. The system comes up with a Smart Parking System that addresses these issues by using data structures for efficient parking spot allocation and reservation management.It enables the advance booking of spots based on the size of vehicles and the availability of such spots. Such a functionality decreases the search time and becomes directly helpful in managing parking in crowded areas. Although the current model includes user registration, reservation management, spot allocation, and sending notifications, the current version is at a very preliminary stage and needs further developments to attain full real-time optimization.

The use of data structures such as graphs, priority queues, dictionaries and lists is vital in enhancing the allocation process, allowing for quick spot access and path finding.Although this is one of the projects toward improving parking organization,other further works are to be done to establish a wholly automated, real-time adaptable system. This Smart Parking System not only addresses immediate parking challenges but also contributes to the broader goal of optimizing urban mobility and reducing congestion.

Keywords: Parking congestion Smart Parking System , Reservation Management,Data structures,urban mobility

Chapter 1

Introduction

1.1 Background

Parking congestion is a major urban issue, with cities like Delhi adding 1,800 vehicles daily. In Bengaluru, average traffic speeds have dropped to 18 km/h, causing commuters to spend 20% more time searching for parking. Effective space management is crucial for improving urban mobility and reducing environmental impacts.

1.2 Problem Statement

The project's work will be in making the allocation of parking space easier and would minimize search time as well. It is also a facility for users who want to find parking space fast and comes into densely populated urban areas as well. By tackling parking congestion challenges, the system seeks to enhance the broader experience that motorists encounter as often as possible while they are around high-demand locations.

1.3 Objective

The objective will be to build an intelligent parking system that utilizes effective data structures for handling real-time parking spot allocation and pre-booking for the best-in-class usage of the available space with regard to vehicular size and availability. For convenience and better efficiency, the system should include the notification facility that immediately informs users about reservation statuses..

1.4 Scope

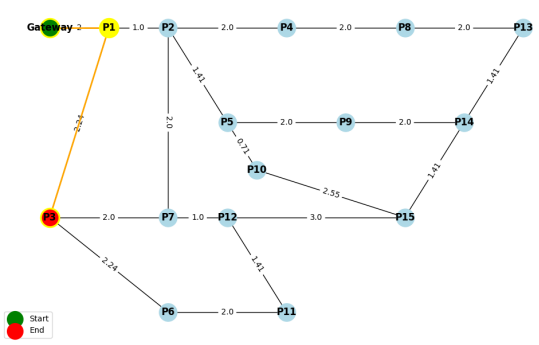
The creation of a prototype with necessary functionalities including user registration, reservation management, place allocation, and notifications is the extent of this project. However, a simple user interface and the incapacity to scale efficiently for large-scale deployment limit the existing solution. By adding real-time adaptability, sophisticated automation, and scalability to accommodate wider use cases, future improvements will try to overcome these constraints.

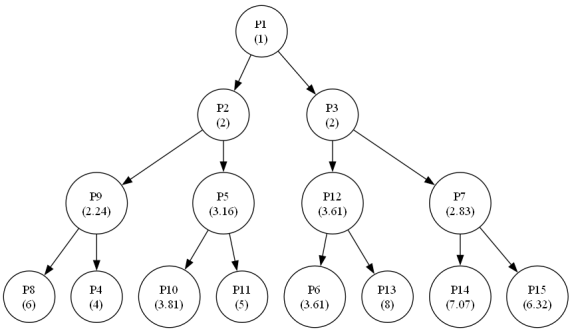
Chapter 2

Methodology

2.1 Data Structures Used

- **Graph (NetworkX Graph)**: Models the parking lot with spots as nodes and distances as edges. Used for pathfinding (Dijkstra's algorithm) to calculate shortest paths.





- **Priority Queue (heapq Min-Heap)**: Finds the nearest available parking spot efficiently with $O(\log n)$ retrieval, prioritizing spots based on distance.
- **List (for Reservation Times)**: Stores start and end times for reservations. Ordered storage simplifies availability checks.
- **Dictionary (dict for Spot/User Data)**: Quick $O(1)$ lookup for accessing parking spots and user information.
- **Queue (List as Queue for Notifications)**: Manages notifications in FIFO order for sequential processing.

2.2 Algorithm/Approach

Dijkstra's Algorithm: The shortest route to the closest parking space is found in this project using Dijkstra's algorithm. A graph is used to depict the parking lot, with nodes standing in for parking spots and edges for pathways. The algorithm minimizes the walking distance for users by choosing the spot that is nearest to the entrance, therefore optimizing spot allocation.

Chapter 3

Implementation

3.1 Programming Language/Tools

Python, SQLite3, regex, Matplotlib, date-time module, and threading are among the programming languages and tools utilized.

3.2 Code Overview

User login and registration functions, including password verification. Integration with SQLite to store and retrieve user credentials.

3.2.1 Database Management Module

- Provides functions to connect to and interact with a SQLite database.
- Manages tables for users, reservations, and parking spots, supporting CRUD operations.

```
def create_connection(db_file):
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except sqlite3.Error as e:
        print(f"Error: {e}")
        return None
def get_parking_spot_reservations(conn):
    cursor = conn.cursor()
    cursor.execute('''
    SELECT
        ps.parking_spot_id,ps.X,ps.Y, ps.spot_size,
        GROUP_CONCAT(
        CASE WHEN r.status = 'active' THEN r.start_time || '|' || r.end_time
            ELSE NULL END, ',') AS reservation_times
    FROM
        parking_spots ps
    LEFT JOIN
        reservations r
    ON
        ps.parking_spot_id = r.parking_spot_id
    GROUP BY ps.parking_spot_id, ps.spot_size, ps.X,ps.Y ORDER BY  ps.parking_spot_id
    ''')
    results = cursor.fetchall()
    spot_data = []
    for row in results:
        parking_spot_id, X, Y, spot_size, reservation_times = row
        reservations = []
        if reservation_times:
            for res in reservation_times.split(','):
                start, end = res.split('|')
                reservations.append((datetime.fromisoformat(start), datetime.fromisoformat(end)))
        coordinates = (X, Y)
        spot_data.append((parking_spot_id, spot_size, coordinates, reservations))
    return spot_data
```

```
        spot_data.append((parking_spot_id, spot_size, coordinates, reservations))
    return spot_data
def insert_user_data(conn, license_plate, vehicle_size):
    try:
        cursor = conn.cursor()
        cursor.execute('''
        INSERT OR IGNORE INTO vehicles (license_plate, vehicle_size)
        VALUES (?, ?)
        ''', (license_plate, vehicle_size))
        conn.commit()
    except sqlite3.Error as e:
        print(f"Error inserting user data: {e}")
def insert_reservation_data(conn, reservation):
    cursor = conn.cursor()
    current_time = datetime.now().isoformat()
    cursor.execute('''
    INSERT INTO reservations (license_plate, parking_spot_id, start_time, end_time, status, username)
    VALUES ( ?, ?, ?, ?, ?,?)
    ''', (reservation['license_plate'], reservation['parking_spot_id'], reservation['start_time'], reserv
    conn.commit()
def get_parking_spot_data(conn):
    cursor = conn.cursor()
    cursor.execute("SELECT parking_spot_id, X, Y FROM parking_spots")
    spots = cursor.fetchall()
    return {spot_id: (x, y) for spot_id, x, y in spots}
# Parking Spot and Reservation Functions
```

3.2.2 Vehicle and Parking Spot Classes

- The `Vehicle` class captures vehicle details.
- The `ParkingSpotNode` class represents a parking spot, with attributes for availability and location.

```
# Class to represent a vehicle with reservation details
class Vehicle:
    def __init__(self, license_plate, entry_time, exit_time):
        self.license_plate = license_plate
        self.entry_time = entry_time
        self.fixed_entry_time = entry_time
        self.has_entered = False
        self.exit_time = exit_time
        self.has_left = False

# Class to represent ParkingSpot
class ParkingSpotNode:
    def __init__(self, spot_id, size, distance, reservations):
        self.spot_id = spot_id
        self.size = size
        self.distance = distance
        self.reservations = reservations
```

3.2.3 Reservation Management Module

- Allows users to create, view, and cancel reservations.
- Checks for spot availability, validates reservation times, and assigns spots based on vehicle size.

```python
# Reservation Management Functions
def create_reservation(conn, username, G):
    license_plate = input("Enter the vehicle's license plate: ").strip()
    if not re.match(r'^[A-Za-z0-9-]{6,8}$', license_plate):
        print("Invalid license plate format. Please enter 6-8 alphanumeric characters or hyphens only.")
        return

    try:
        vehicle_size = int(input("Enter vehicle size (0 = Small, 1 = Medium, 2 = Large): ").strip())
        if vehicle_size not in [0, 1, 2]:
            print("Invalid vehicle size. Enter 0, 1, or 2.")
            return
    except ValueError:
        print("Invalid input. Vehicle size must be an integer (0, 1, or 2).")
        return

    try:
        input_start_time = input("Enter start time (YYYY-MM-DD HH:MM): ").strip()
        input_end_time = input("Enter end time (YYYY-MM-DD HH:MM): ").strip()
        start_time = datetime.fromisoformat(input_start_time)
        end_time = datetime.fromisoformat(input_end_time)
        if end_time <= start_time:
            print("End time must be after start time.")
            return
    except ValueError:
        print("Invalid datetime format.")
        return
```

```python
    insert_user_data(conn, license_plate, vehicle_size)
    start_spot_id = 'Gateway'
    parking_spot_id = find_available_spot(vehicle_size, start_time, end_time, conn, G, start_spot_id=start_spot_id)

    if parking_spot_id is None:
        print("No available parking spots for the specified size and time range.")
        return

    reservation = {
        "license_plate": license_plate,
        "parking_spot_id": parking_spot_id,
        "start_time": start_time.isoformat(),
        "end_time": end_time.isoformat(),
        "status": 'active',
        "username": username
    }
    insert_reservation_data(conn, reservation)

    print(f"Reservation created for vehicle {license_plate} at parking spot {parking_spot_id} from {start_time} to {end_time}.")
    path = nx.shortest_path(G, source=start_spot_id, target=parking_spot_id, weight='weight')
    visualize_parking_graph(G,start_spot_id , parking_spot_id, path)
```

```python
def cancel_reservation(conn,username):
    # Get license plate and validate format
    license_plate = input("Enter the license plate of the reservation to cancel: ").strip()
    if not re.match(r'^[A-Za-z0-9-]{6,8}$', license_plate):
        print("Invalid license plate format. Please enter 6-8 alphanumeric characters or hyphens only.")
        return

    # Check if there's an active reservation for this license plate
    reservation = get_reservation_by_plate(conn, license_plate,username)
    if not reservation:
        print("No active reservation found for this license plate.")
        return

    # Confirm cancellation
    if reservation[7] == 0:
        confirm = input("Are you sure you want to cancel this reservation? (yes/no): ").strip().lower()
        if confirm == 'yes':
            update_reservation_status(conn, reservation[0], 'cancelled')
            print(f"Reservation for license plate {license_plate} has been cancelled.")
        else:
            print("Cancellation aborted.")
    else :
        print("You can't cancel; you've already entered.")
```

```python
def know_your_spot( conn,username,G):
    # Ask for the license plate number to find the reserved spot
    license_plate = input("Enter your vehicle's license plate number: ").strip()

    # Retrieve reserved spot ID from the database
    reserved_spot_id = get_reservation_by_plate(conn, license_plate, username)
    if reserved_spot_id is None:
        print("No reservation found for this license plate.")
        return
    else :
        reserved_spot_id = reserved_spot_id[2]
    # Display the reserved parking spot
    print(f"Your reserved parking spot is: {reserved_spot_id}")

    # Prompt the user for their starting position
    start_spot_id = input("Enter your current parking spot ID : ")


    # Find the shortest path from the user's starting spot to the reserved spot
    try:
        path = nx.shortest_path(G, source=start_spot_id, target=reserved_spot_id, weight="weight")
    except nx.NetworkXNoPath:
        print("No path found between your position and the reserved spot.")
        return
    # Call visualize_parking_graph to show the path
    visualize_parking_graph(G, start_spot_id, reserved_spot_id, path)
```

```python
def find_available_spot(vehicle_size, start_time, end_time, conn, G, start_spot_id=None, start_coordinates=(0, 0)):
    spots = get_parking_spot_reservations(conn)
    spots_heap = []

    for spot_id, size, coordinates, reservations in spots:
        distance = calculate_distance(G, spot_id, start_spot_id, start_coordinates)
        heapq.heappush(spots_heap, (distance, spot_id, ParkingSpotNode(spot_id, size, distance, reservations)))

    while spots_heap:
        _, spot_id, spot = heapq.heappop(spots_heap)
        if spot.size < vehicle_size:
            continue
        is_available = all(end_time <= res_start or start_time >= res_end for res_start, res_end in spot.reservations)
        if is_available:
            return spot.spot_id
    return None
```

3.2.4 Pathfinding Module (Shortest Route Calculation)

- Uses `NetworkX` to model the parking lot as a graph.
- A Dijkstra-based `calculate_distance` function finds the shortest path from entry to the reserved spot.
- Displays the path visually using `Matplotlib`.

```python
def create_graph(conn):
    positions = get_parking_spot_data(conn)
    G = nx.Graph()
    for node in positions:
        G.add_node(node, pos=positions[node])
    G.add_node('Gateway',pos = (0,6))

    edges = [
        ("Gateway",'P1'),
        ("P1", "P2"), ("P1", "P3"), ("P2", "P4"), ("P2", "P5"), ("P2", "P7"),
        ("P3", "P6"), ("P3", "P7"), ("P4", "P8"), ("P5", "P9"), ("P5", "P10"),
        ("P6", "P11"), ("P7", "P12"), ("P8", "P13"), ("P9", "P14"), ("P10", "P15"),
        ("P11", "P12"), ("P12", "P15"), ("P13", "P14"), ("P14", "P15")
    ]

    for u, v in edges[1:]:
        x1, y1 = positions[u]
        x2, y2 = positions[v]
        weight = round(math.sqrt((x2 - x1)**2 + (y2 - y1)**2), 2)
        G.add_edge(u, v, weight=weight)
    G.add_edge("Gateway","P1",weight =2)
    return G
```

Figure 3.1: Graphs for parking spots

```python
def visualize_parking_graph(G, start_spot_id, end_spot_id, path):
    # Define positions for nodes (assumes node positions are stored in 'pos' attribute in G)
    positions = nx.get_node_attributes(G, 'pos')

    # Draw the base graph
    plt.figure(figsize=(10, 6))
    nx.draw(
        G, pos=positions, with_labels=True, node_size=500, node_color="lightblue",
        font_weight="bold", font_color="black", edge_color="black"
    )

    # Draw edge weights
    edge_labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos=positions, edge_labels=edge_labels)

    # Highlight the path
    path_edges = list(zip(path, path[1:]))  # Pairs of nodes in the path for edges
    nx.draw_networkx_nodes(G, pos=positions, nodelist=path, node_color="yellow", node_size=600)
    nx.draw_networkx_edges(G, pos=positions, edgelist=path_edges, edge_color="orange", width=2)

    # Highlight start and end nodes
    nx.draw_networkx_nodes(G, pos=positions, nodelist=[start_spot_id], node_color="green", node_size=400, label="Start")
    nx.draw_networkx_nodes(G, pos=positions, nodelist=[end_spot_id], node_color="red", node_size=400, label="End")

    plt.title("Parking Graph with Highlighted Path")
    plt.legend(scatterpoints=1)
    plt.show()
```

3.2.5 Real-Time Notification System

- A background thread monitors reservation times, sending reminders and alerts for upcoming reservations.

Entry/Exit Management

- Logs entry and exit times, updating the database in real time.

```python
class ReservationSystem:
    def __init__(self, db_path, username):
        self.db_path = db_path
        self.username = username
        self.notification_queue = []

    def check_reservations(self):
        while True:
            self.send_notifications()
            time.sleep(5)  # Check every 5 seconds
```

```
def send_notifications(self):
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    now = datetime.now()

    # Check reservations that should start now
    cursor.execute(
        "SELECT reservation_id, license_plate, start_time, end_time, has_entered, has_left "
        "FROM reservations WHERE username = ? AND status = 'active'",
        (self.username,)
    )
    reservations = cursor.fetchall()

    for res_id, license_plate, start_time, end_time, has_entered, has_left in reservations:
        start_time = datetime.fromisoformat(start_time)
        end_time = datetime.fromisoformat(end_time)

        # 1. Notify if reservation time has started
        if start_time <= now < start_time + timedelta(minutes=1) and not has_entered:
            self.notification_queue.append(
                f"Notification: Reservation for {license_plate} has started. Please enter the parking lot.{now.isoformat()}"
            )

        # 2. Cancel reservation and notify if no entry within 1 minute of start time
        if now >= start_time + timedelta(minutes=1) and not has_entered:
            update_reservation_status(conn, res_id, 'cancelled')
            self.notification_queue.append(
                f"Notification: Reservation for {license_plate} has been cancelled due to no entry within 1 minute.{now.isoformat()}"
            )

        # 3. Notify if overstaying (user did not leave by the end time)
        if end_time < now and has_entered and not has_left:
            self.notification_queue.append(
                f"Notification: Reservation time for {license_plate} has ended. You are currently overstaying.{now.isoformat()}"
            )
    conn.close()
```

```
def enter_parking_lot(self, license_plate,G):
    # Mark the reservation as entered
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute("SELECT start_time,parking_spot_id FROM reservations WHERE license_plate = ? AND username = ?  AND statu
        (license_plate, self.username))
    retrieved = cursor.fetchone()
    if retrieved:
        reservation_start_time = datetime.fromisoformat(retrieved[0])
        reserved_spot_id = retrieved[1]
        if datetime.now() >reservation_start_time - timedelta(minutes =1):
            path = nx.shortest_path(G, source="Gateway", target=reserved_spot_id, weight='weight')
            visualize_parking_graph(G,"Gateway" ,reserved_spot_id , path)
            cursor.execute(
                "UPDATE reservations SET has_entered = 1 WHERE license_plate = ? AND username = ? AND status = 'active' ",
                (license_plate, self.username)
            )
            conn.commit()
            print(f"Entry recorded for vehicle with license plate {license_plate}.")
        else :
            print("You cannot enter before your reserved start time.")
    else :
        print("NO active reservation for this number plate.")
    conn.close()
```

```
    def exit_parking_lot(self, license_plate,G):
        # Mark the reservation as exited
        conn = sqlite3.connect(self.db_path)
        cursor = conn.cursor()
        cursor.execute("SELECT has_entered,parking_spot_id FROM reservations WHERE license_plate = ? AND username = ?  AND st
            (license_plate, self.username))
        retrieved = cursor.fetchone()
        if retrieved:
            has_entered = int(retrieved[0])
            start_spot_id = retrieved[1]
            if has_entered :
                path = nx.shortest_path(G, source=start_spot_id, target="Gateway", weight='weight')
                visualize_parking_graph(G,start_spot_id , "Gateway", path)
                cursor.execute(
                    "UPDATE reservations SET has_left = 1,status = 'completed' WHERE license_plate = ? AND username = ? AND status = 'a
                    (license_plate, self.username)
                )
                conn.commit()
                print(f"Exit recorded for vehicle with license plate {license_plate}.")
                self.notification_queue.clear()

            else :
                print("You have not entered to exit")
        else :
            print("NO active reservation for this number plate.")
        conn.close()
```

3.2.6 User Interface (CLI)

- Command-line interface for login, reservation management, and notifications.

```
def login(conn):
    cursor = conn.cursor()
    username = input("Enter your username: ")
    password = input("Enter your password: ")
    cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))
    user = cursor.fetchone()

    if user:
        print("Login successful!")
        return True, username
    else:
        print("Invalid username or password.")
        return False, None

def signup(conn):
    cursor = conn.cursor()
    username = input("Choose a username: ")
    password = input("Choose a password: ")
    cursor.execute("SELECT * FROM users WHERE username = ?", (username,))
    if cursor.fetchone():
        print("Username already taken. Please choose another username.")
        return False

    cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, password))
    conn.commit()
    print("Signup successful!")
    return True
```

```
def main_menu(conn, username, graph):
    db_path = "./parking_system_new.db"
    reservation_system = ReservationSystem(db_path, username)

    # Start reservation checking in a separate thread
    checker_thread = threading.Thread(target=reservation_system.check_reservations)
    checker_thread.daemon = True
    checker_thread.start()

    try:
        while True:
            print("\n--- Parking Reservation System ---")
            print("1. Create a Parking Reservation")
            print("2. Cancel a Reservation")
            print("3. Know Your Spot")
            print("4. Show Notifications")
            print("5. Enter Parking Lot")  # New option to mark entry
```

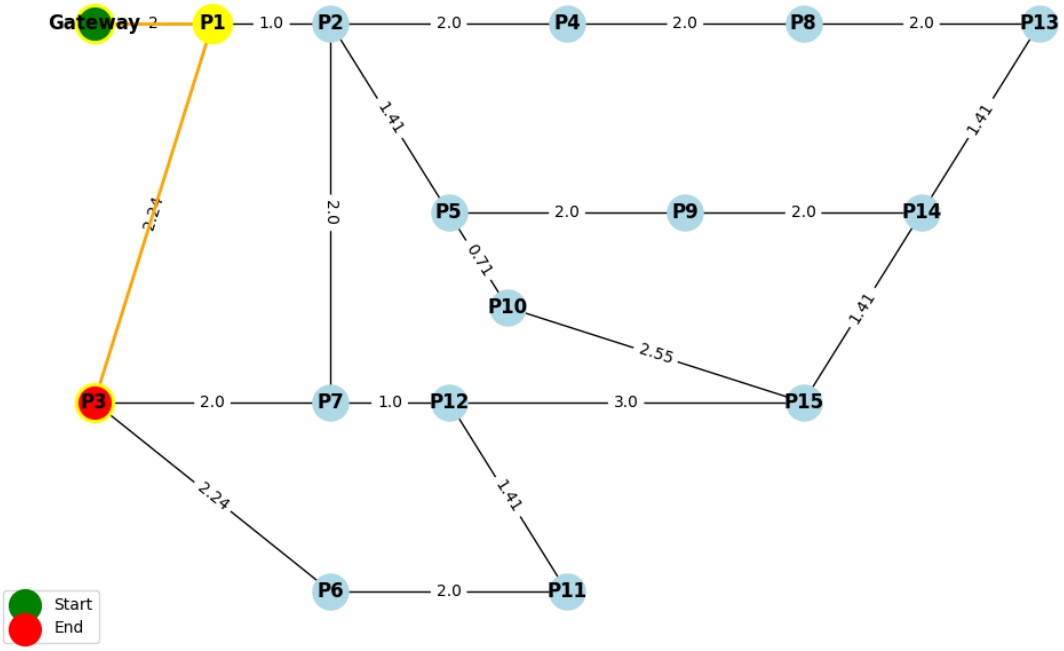
```
            print("6. Exit Parking Lot")  # New option to mark exit
            print("7. Exit Program")

            choice = input("Please select an option (1-7): ")

            if choice == '1':
                create_reservation(conn, username, graph)
            elif choice == '2':
                cancel_reservation(conn, username)
            elif choice == '3':
                know_your_spot(conn, username, graph)
            elif choice == '4':
                show_notifications(reservation_system)
            elif choice == '5':  # User entering the parking lot
                license_plate = input("Enter your vehicle's license plate to mark entry: ").strip()
                reservation_system.enter_parking_lot(license_plate,graph)
            elif choice == '6':  # User exiting the parking lot
                license_plate = input("Enter your vehicle's license plate to mark exit: ").strip()
                reservation_system.exit_parking_lot(license_plate,graph)
```

```
            elif choice == '7':
                print("Exiting the Parking Reservation System.")
                break
            else:
                print("Invalid option. Please select a number between 1 and 7.")
        except KeyboardInterrupt:
            checker_thread.join()
```

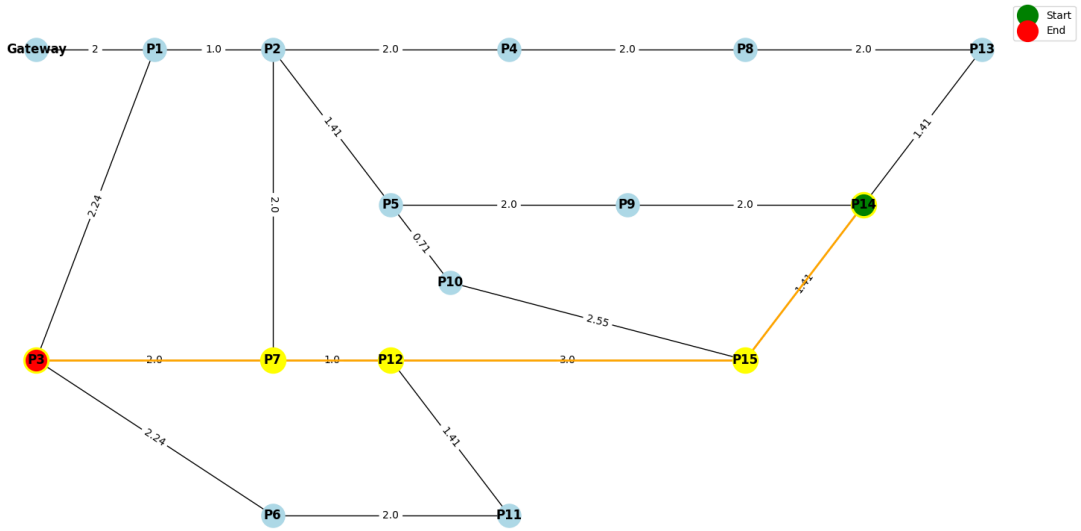
```
def main():
    print("Welcome to the Parking Reservation System")
    database = "./parking_system_new.db"
    with create_connection(database) as conn :
        graph = create_graph(conn)
        try :
            while True:
                print("\nPlease choose an option:")
                print("1. Login")
                print("2. Signup")

                choice = input("Enter 1 or 2: ")

                if choice == '1':
                    Flag,username =login(conn)
                    if Flag:
                        main_menu(conn,username,graph)
                        break
                    else:
                        print("Login failed. Please try again.")
                elif choice == '2':
                    if signup(conn):
                        print("Signup successful! Please log in to continue.")
                        continue  # Prompt user to log in after successful signup
                    else:
                        print("Signup failed. Please try again.")
                else:
```

3.3 Outputs

```
*IDLE Shell 3.12.1*
File  Edit  Shell  Debug  Options  Window  Help
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: E:\DS project\Parking_system\final\parking_system.py
Welcome to the Parking Reservation System

Please choose an option:
1. Login
2. Signup
Enter 1 or 2: 1
Enter your username: Sandeep
Enter your password: 23737123
Login successful!

--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 1
Enter the vehicle's license plate: SD23MJ
Enter vehicle size (0 = Small, 1 = Medium, 2 = Large): 2
Enter start time (YYYY-MM-DD HH:MM): 2024-11-15 19:33
Enter end time (YYYY-MM-DD HH:MM): 2024-11-15 19:35
Reservation created for vehicle SD23MJ at parking spot P3 from 2024-11-15 19:33:00 to 2024-11-15 19:35:00.
```



```
*IDLE Shell 3.12.1*
File  Edit  Shell  Debug  Options  Window  Help

--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 4
Notification: Reservation for SD23MJ has started. Please enter the parking lot.2024-11-15T19:33:04.892767

--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 5
Enter your vehicle's license plate to mark entry: SD23MJ
Entry recorded for vehicle with license plate SD23MJ.
```




```
*IDLE Shell 3.12.1*
File  Edit  Shell  Debug  Options  Window  Help

--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 3
Enter your vehicle's license plate number: SD23MJ
Your reserved parking spot is: P3
Enter your current parking spot ID : P14

--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 4
Notification: Reservation time for SD23MJ has ended. You are currently overstaying.2024-11-15T19:35:09.916024
```

```
--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 6
Enter your vehicle's license plate to mark exit: SD23MJ
Exit recorded for vehicle with license plate SD23MJ.
```

```
--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 4
Notification: Reservation for SD23MJ has been cancelled due to no entry within 1 minute.2024-11-15T19:43:00.000449
```

```
--- Parking Reservation System ---
1. Create a Parking Reservation
2. Cancel a Reservation
3. Know Your Spot
4. Show Notifications
5. Enter Parking Lot
6. Exit Parking Lot
7. Exit Program
Please select an option (1-7): 2
Enter the license plate of the reservation to cancel: SD23MJ
Are you sure you want to cancel this reservation? (yes/no): yes
Reservation for license plate SD23MJ has been cancelled.
```

Chapter 4

Results and Analysis

4.1 Performance Analysis

Performance is analyzed for each function with time and space complexities.

4.1.1 Time Complexity

- **User Authentication:** O(1) per query due to direct database lookups.
- **Reservation Functions:** The time complexity for finding an available spot is **O(n log n)** because of the heap operations and the need to calculate distances for each parking spot.
- **Distance Calculation and Pathfinding:** The time complexity is **O(E log V)**, where E is the number of edges (connections between spots) and V is the number of parking spots (nodes).
- **Notification System:** The time complexity is **O(m)**, where m is the number of active reservations being monitored.

4.1.2 Space Complexity

- **User Authentication:** O(1) for storing login credentials and performing checks.
- **Reservation Functions: O(n)** to store reservation information.
- **Distance Calculation and Pathfinding**: **O(V + E)**, as it holds the graph structure and distance weights.
- **Notification System**: **O(m)** for storing active notifications.

4.1.3 Total Complexity

- **Total Time Complexity:** Dominated by $O(n \log n)$ for parking spot selection and pathfinding.
- **Total Space Complexity:** $O(n + V + E)$, where n is the number of reservations, and V, E refer to the graph structure.

Chapter 5

Conclusion

This system effectively handles parking spot reservations by combining real-time space availability, vehicle monitoring, and user authentication. It uses graph theory for enhancing pathfinding to reserved places. It thus offers state-of-the-art, user-friendly solutions toward maximizing parking efficiency and convenience with notifications, reservation changes, and smooth user interaction The system uses Dijkstra's algorithm to find the nearest spot, minimizing walking distance and improving spot allocation efficiency.

5.1 Future Work

Future work includes enhancing real-time management and scalability for large-scale settings and adapting for real-time application.