wcs

**SUBMITTED BY : BISMA AZHAR**

**SUBMITTED TO: SIR JAMAL ABDUL AHAD**

**COURSE TITLE: DATA STRUCTURE**

**ROLL NO: 14641**

**SUBMISSION DATE: 25 December 2024**

**FINAL PROJECT REPORT**

Contents

[1.Introduction: 3](#_Toc186858242)

[2 System Overview: 3](#_Toc186858243)

[Key features: Adding Routes 3](#_Toc186858244)

[1. System Design. 3](#_Toc186858245)

[ Methods: 3](#_Toc186858246)

[3.Algorithm Implementation: Dijkstra's Algorithm: 4](#_Toc186858247)

[Visit Neighbors: 4](#_Toc186858248)

[Priority Queue: 4](#_Toc186858249)

[Termination: 4](#_Toc186858250)

[Path reconstruction: 4](#_Toc186858251)

[4. Code Walkthrough: 4](#_Toc186858252)

[Challenges Encountered: 5](#_Toc186858253)

[File Handling: 5](#_Toc186858254)

[Graph Representation: 5](#_Toc186858255)

[Approach Complexity: 5](#_Toc186858256)

[Sample Input Data : 5](#_Toc186858257)

[Future Enhancements: 6](#_Toc186858258)

[Conclusion : 6](#_Toc186858259)

[DIAGRAM: 6](#_Toc186858260)

[USE CASE DIAGRAM: 8](#_Toc186858261)

**Ride-Sharing System Report:**

# 1.Introduction:

The ride-sharing system aims to help users find the shortest route between two cities using Dijkstra's algorithm. Dijkstra's algorithm is a famous graph traversal algorithm that finds the shortest path between nodes in a graph, ensuring the minimal distance is selected. This system allows users to create new routes, find the quickest journey between two cities, and view the current routes accessible in the system.

## 2 System Overview:

The system is implemented in Python, with a Graph class that serves as the core structure. The class provides methods for adding routes, displaying routes, and calculating the shortest path using Dijkstra's algorithm. The graph data can be loaded from a file and saved to a file, making it easy to persist and retrieve the information.

Key features:  
Adding Routes:

Users can create new routes between cities by entering the city names and distances.  
Shortest Path Calculation**:**

Dijkstra's algorithm is used to determine the shortest path between two cities.  
Displaying Routes:

Users can view all routes that are currently saved in the system.  
File Handling:

The system supports importing graph data from a file and saving it back to the file, assuring long-term storage.

1. System Design.  
   The software is built around a class called Graph, which represents the system's structure. Here's a summary of the major components:  
     
   Graph Class:

The class stores the graph as a dictionary, with each key representing a city and the value a list of tuples representing surrounding cities and their distances.

* Methods:  
    
  **1**.**add\_edge(city1, city2, distance):** Creates a bidirectional path between two cities of specified distance.  
  **2**.**display\_routes():** This function displays all of the system's routes.  
  **3**.**load\_from\_file(file\_name):** Loads graph data from a specified file.  
  **4**.**The function save\_to\_file(file\_name)** saves the graph data to a file.  
  **5**.**shortest\_path(start, goal):** Implements Dijkstra's algorithm to find the shortest path between the start and goal cities.

3.Algorithm Implementation: Dijkstra's Algorithm:  
The shortest path between two cities is computed using Dijkstra's algorithm, which operates as follows:  
  
Initialization:  
Begin with a beginning city (the starting point).  
Set the distance to the starting city to zero and all other cities to infinite.

Visit Neighbors:  
Visit each city's neighbors and update the distances if a shorter path is discovered.

Priority Queue:  
A priority queue (based on Python's heapq) ensures that the next city with the shortest distance is processed first.

Termination:  
The algorithm runs until the destination city is reached or all potential routes have been exhausted.

Path reconstruction:  
Once the destination city is reached, the path is recreated by going backwards via the cities from the destination to the source.

1. Code Walkthrough:  
   The application starts by loading any existing routes from a file, and then shows the user a choice of options:  
     
    1.Create new routes.  
    2.Find the shortest route between two cities.  
    3.Display the current routes.  
    4.Exit the system.

Each action invokes the associated method in the Graph class. When building a route, the add\_edge method is used to establish a bidirectional connection between two cities. When determining the shortest path, the shortest\_path method computes the path using Dijkstra's algorithm.

# Challenges Encountered:

## File Handling:

Proper file format handling and error checking were necessary to ensure accurate data reading and writing.

## Graph Representation:

Effectively managing a graph with cities as nodes and routes as edges, ensuring that no duplicate edges are created.

## Approach Complexity:

Dijkstra's approach is efficient for smaller graphs, but its efficiency degrades with larger graphs.

# Sample Input Data :

The input file requirements.txt contains the following sample data (city pairs with distances)

Karachi,Hyderabad,150.0

Karachi,Quetta,690.0

Hyderabad,Sukkur,220.0

Sukkur,Multan,395.0

Sukkur,Quetta,385.0

Multan,Lahore,340.0

Multan,Faisalabad,190.0

Multan,Quetta,100.0

Lahore,Islamabad,375.0

Lahore,Faisalabad,140.0

Islamabad,Peshawar,180.0

Peshawar,Abbottabad,120.0

1. **Results and Output :**

The user is prompted to choose one of the following options:

**Add Route:**

Allows adding a new route between two cities.

**Find Shortest Path:** Calculates the shortest path between two cities.

**Display Routes:**

Lists all available routes in the system.

**For example,** if a user selects the "Find Shortest Path" option, the program will output the shortest route along with the total distance between the start and goal cities.

# Future Enhancements:

**Graph Visualization:**

Using a graphical user interface (GUI) to improve the visual depiction of cities and routes.

**More Advanced Algorithms:**

Using other shortest path algorithms, such as A\*, for larger or more complex graphs.

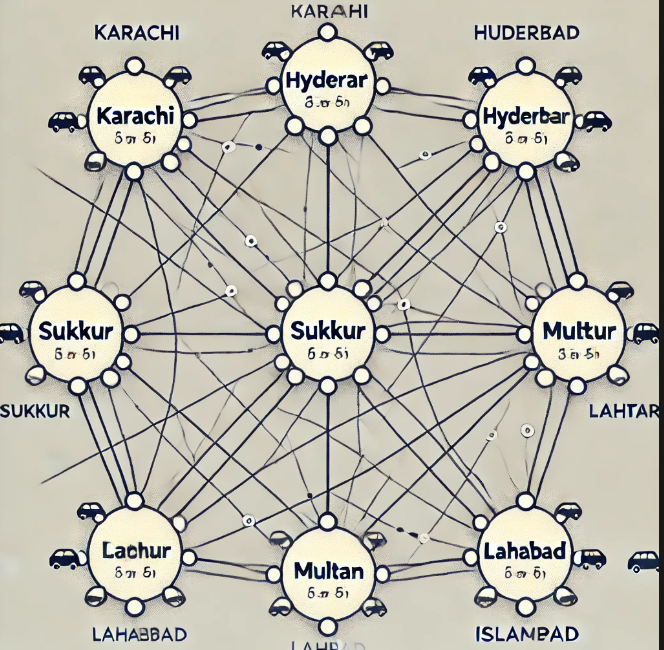
**Real-Time Traffic Updates:**

Use real-time traffic data to dynamically adjust routes.

# Conclusion :

The ride-sharing system is a simple yet effective tool for finding the shortest route between two cities. By using Dijkstra's algorithm, it efficiently calculates the minimal distance and provides an interactive interface for adding new routes and viewing existing ones. This system can be further expanded by integrating real-time data and providing a user-friendly interface.

# DIAGRAM:



# USE CASE DIAGRAM:

