SUMMER TRAINING/INTERNSHIP

PROJECT REPORT

(Term June–July 2025)

AIRPORT ROUTE PLANNER

(A project to find the least time and cost route between airports using Dijkstra's Algorithm)

Submitted by : **Aushika Gupta**
Registration Number: **12318277**

Course Code: **PETV82**

Under the Guidance of: **Dr. Prabhjeet Kaur**

School of Computer Science and Engineering
Lovely Professional University

Acknowledgement

I would like to express my deepest gratitude to **Mrs. Prabhjeet Kaur**, Course Coordinator, School of Computer Science and Engineering, Lovely Professional University, for their **continuous guidance, motivation, and valuable feedback** throughout the course of this summer training project.

Working on the project "**Airport Route Planner**" has been a transformative learning experience. It helped me practically apply the concepts of **data structures and graph algorithms, especially Dijkstra's algorithm, in solving a real-world problem.**

I also thank the faculty of Lovely Professional University for maintaining an inspiring academic atmosphere and providing the necessary tools and resources to complete this project successfully.

My heartfelt thanks to my friends and family for their patience, support, and encouragement throughout this journey.

Table of Contents

Chapter No.	Chapter Title	Page No.
Chapter 1	Introduction	4 - 5
Chapter 2	Training Overview	5 - 10
Chapter 3	Project Details	11 - 16
Chapter 4	Implementation	17 - 25
Chapter 5	Results and Discussion	26 - 29
Chapter 6	Conclusion	30

Chapter 1: Introduction

1.1 Company Profile / Institution Overview

Lovely Professional University (LPU), located in Punjab, India, is one of the **largest and most prestigious private universities** in the country. Known for its focus on **innovation, research, and practical learning**, LPU offers a wide range of undergraduate and postgraduate programs in **engineering, science, management, and more.** The university is committed to providing a holistic learning environment that nurtures both academic excellence and industry readiness.

As a part of its structured academic curriculum, LPU organizes **Summer Training Programs** to help students **bridge the gap between theoretical knowledge and real-world applications**. These training programs are an essential component of the B.Tech CSE curriculum and are designed to enhance technical skills, industry exposure, and practical problem-solving abilities.

1.2 Training Program Overview

During the **Summer Training Term (June–July 2025)**, I undertook a **6-week training program** comprising **50+ hours of hands-on learning and project development**. The training focused on applying core concepts of **data structures, algorithms, and real-world programming** in a guided project environment.

The objective of the training was to encourage students to independently develop a working project using modern tools and methodologies while adhering to software development best practices.

1.3 Objective of the Project

The primary objective of this summer project was to build a **functional and intelligent Airport Route Planner** that can compute the best possible flight route between two airports based on user preference — either **minimum time** or **minimum cost**. This was accomplished using **graph algorithms** (specifically, Dijkstra's algorithm), data abstraction techniques, and a structured software development lifecycle.

The project not only helped in enhancing my understanding of algorithms and graph theory but also gave me a chance to experience a self-driven software development cycle — from requirement gathering to implementation and testing.

Chapter 2: Training Overview

2.1 Tools & Technologies Used

The training was primarily focused on mastering **Data Structures and Algorithms (DSA)**, with hands-on implementation in **C++**. The following tools and technologies were used:

- **Programming Language:** C++
- **Development Environment:** Visual Studio Code / Code::Blocks
- **Concepts Practiced With:**
  - Time and Space Complexity Analysis
  - Arrays (1D and 2D), Strings
  - Linked Lists, Stacks, Queues
  - Recursion and Trees
  - Binary Search Trees, Heaps, Hashing
  - Graphs (BFS, DFS, Dijkstra's Algorithm)

- **Training Mode:** Instructor-led sessions via CodeTantra

- **Mentor:** *Prabhjeet Kaur (ID: 32420)*

2.2 Areas Covered During Training

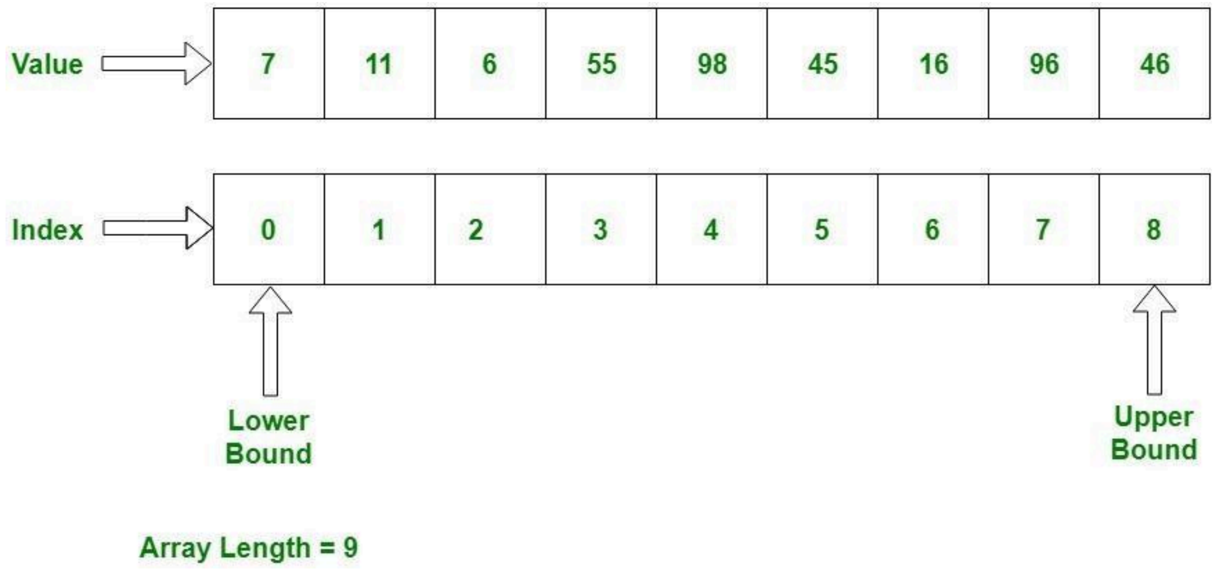
The **DSA Summer Bootcamp** was a **6-week intensive program** that aimed to take learners "From Basics to Brilliance." The training was meticulously structured to build strong problem-solving foundations using DSA. The program covered the following core areas:

Week 1: Getting Started with DSA

- Introduction to DSA

- Time and Space Complexity

- Arrays: Basics, Insertion, Deletion, Traversal

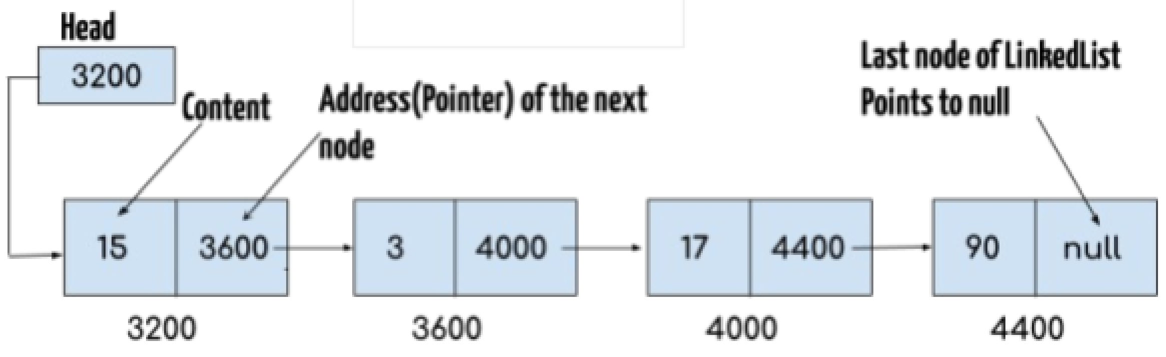


Week 2: Intermediate Data Structures

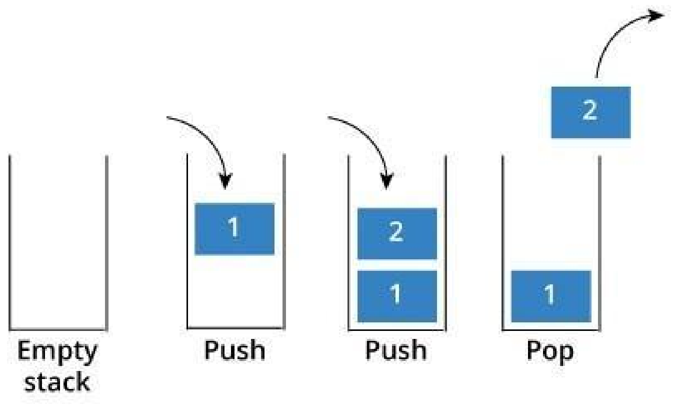
- Searching (Linear & Binary)
- Sorting Algorithms (Bubble, Selection, Insertion)
- 2D Arrays and Matrix Operations
- Strings: Declaration, Manipulation, and STL Functions

Week 3: Linear Data Structures

- Linked Lists (Singly and Doubly)

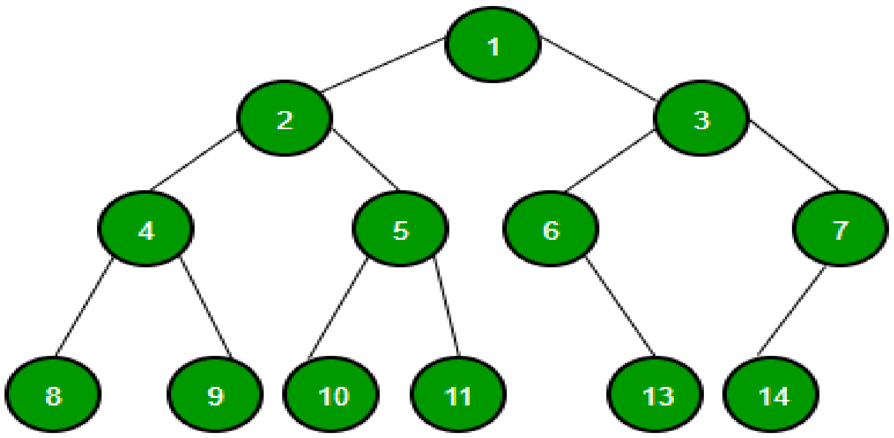

Head
3200
Content
Address(Pointer) of the next node
Last node of LinkedList Points to null
15
3600
3
4000
17
4400
90
null
3200
3600
4000
4400

- Stacks and Queues: Implementation and Applications



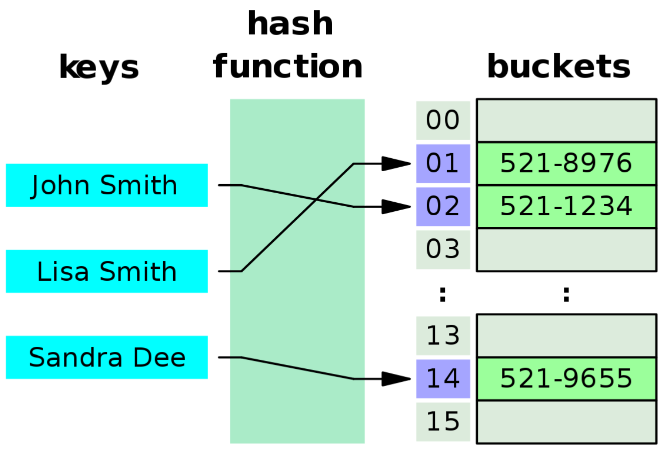
Week 4: Recursive Structures and Trees

- Recursion: Understanding Stack Frames and Backtracking
- Binary Trees and Tree Traversals
- Binary Search Trees: Operations and Use Cases



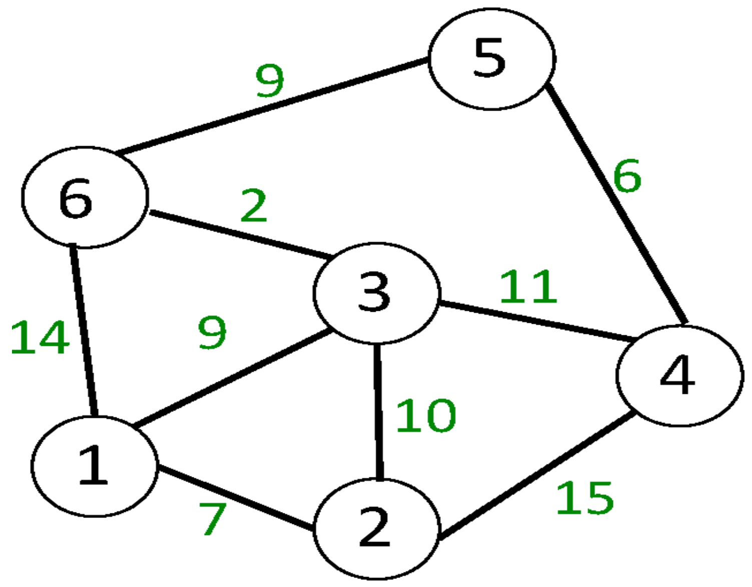
Week 5: Advanced Structures

- Heaps and Priority Queues
- Hashing Techniques: Linear Probing, Chaining



Week 6: Graphs and Algorithms

- Graph Basics: Representation (Adjacency List/Matrix)
- BFS and DFS Traversal
- Dijkstra’s Algorithm for Shortest Path



This progression helped in developing logical thinking, efficiency in coding, and solving real-world problems — directly contributing to the final project: **Airport Route Planner**.

2.3 Daily / Weekly Work Summary

Each week consisted of daily 2-hour live sessions, followed by self-practice. A blend of theoretical learning and coding assignments ensured holistic understanding. Here's a summarized timeline:

Week	Focus	Key Activities
Week 1	**Introduction, Arrays, Time & Space Complexity**	**Learned array operations, space optimization, and big-O notation.**
Week 2	**Searching, Sorting, Strings**	**Practiced pattern problems using sorting and string manipulation.**
Week 3	**Linked Lists, Stacks, Queues**	**Implemented each structure from scratch, solved medium-level problems.**
Week 4	**Recursion and Trees**	**Tackled tree traversal problems and recursive problem-solving approaches.**
Week 5	**Heaps and Hashing**	**Applied priority-based problems and efficient lookups using hashing.**
Week 6	**Graphs & Dijkstra's Algorithm**	**Built pathfinding solutions; implemented Dijkstra's from scratch for use in final project.**

Table 1: Timeline of the course

Chapter 3: Project Details

3.1 Title of the Project

Airport Route Planner – An **intelligent system** to determine the most **optimal route** between two airports based on user preference for either minimum travel time or lowest cost, using **Dijkstra's algorithm**.

3.2 Problem Overview: This is the shortest path algorithm problem as we need to calculate the **earliest arrival time**, with a given start time. So, we can take the problem as a **graph where all the airports are nodes and the flights are the edges with weight of time interval** i.e. the time difference between arrival time of destination airport and departure time of source airport. The challenge lies in developing a solution that can efficiently compute the optimal path considering two variables: **cost** and **time**. This requires the use of appropriate **data structures and algorithms** to process and compare multiple routes in a simulated flight network. Here we need to take care that while going from origin airport to destination airport, and while taking flights from one airport to another, we can take only those flights which can be caught. So, time plays an important role in this problem as we can take only the flights whose departure time from the airport is later than our arrival time at the airport. So, to solve this we will use Dijkstra's Algorithm with a little modification.

3.2.1 Dijkstra's Algorithm: Dijkstra's Algorithm is a **shortest path algorithm** used to calculate the shortest path between nodes in a graph. This algorithm was created and published by computer scientist Dr. Edsger W. Dijkstra. The algorithm exists in many variants. Dijkstra's original algorithm was to find the shortest path between two given nodes, but a more common variant fixes a **single node as the "source" node** and finds shortest paths from the source to **all other nodes in the graph**, producing a shortest-path tree. In Dijkstra's algorithm, we start from a source node and initialize its distance by zero. Next, we push the source node to a **priority queue** with a cost equal to zero. After that, we perform multiple steps. In each step, we extract the node with the lowest cost, update its neighbors' distances, and push them to the priority queue if needed. Each of the neighboring nodes is inserted with its respective new cost, which is equal to the cost of the extracted node plus the edge we just passed through. We continue to visit all nodes until there are no more nodes to extract from the priority queue. Then, we return the calculated distances.

3.3 Scope and Objectives

3.3.1 Scope of the Project

This project aims to develop a smart **Airport Route Planner** that assists users in finding the **fastest and/or cheapest** routes between airports using real or realistic flight data. By integrating **graph algorithms**, **machine learning for delay**

prediction, and optionally a **visual and web-based interface**, the system provides users with intelligent and informed travel options. The scope covers:

- Modeling real-world flight routes as a **graph of airports and flights**.
- Implementing **shortest path algorithms (Dijkstra)** to determine optimal travel paths.
- Storing and retrieving user trips via a **local SQLite database**.
- Enabling **data-driven delay predictions** to make route recommendations smarter.
- Laying groundwork for future integration with **real-time APIs** (e.g., OpenSky, FAA).
- Potential future extension to **cost-based optimization** and **web deployment**.

3.3.2 Objectives

1. **Dataset Integration**
    - Load and preprocess Indian Airlines flight data from Kaggle.
    - Simulate fare and delay data for enhanced prediction.
2. **Graph Construction**
    - Represent airports and flights as a directed, weighted graph.
    - Assign edge weights based on time (and optionally cost).

3. **Route Planning**
    - Implement Dijkstra's Algorithm to find the **fastest route**.
    - Extend logic to support **least-cost** or **multi-objective optimization**.
4. **Delay Prediction (ML Module)**
    - Use dummy or real features to train a simple regression model to predict delays.
    - Display estimated arrival times with predicted delays.
5. **User Management**
    - Create login/register functionality using SQLite.
    - Save and retrieve **user trips and itineraries** from the database.
6. **Graph Visualization**
    - Generate visual maps of routes using **Folium** or similar libraries.
7. **UI Layer**
    - Design a basic interface using **CLI** or a web interface via **Flask/Streamlit**.

3.4 System Requirements

3.4.1 Hardware Requirements

- **Minimum 4 GB RAM**
- **Intel i3 Processor or higher**
- **At least 200 MB of free disk space**

3.4.2 Software Requirements

- **Operating System: Windows 10 or Linux**
- **Programming Language: Python (version 3.x)**
- **IDE: Visual Studio Code**
- **Environment Manager (Optional): Anaconda**
- **Required Libraries:**
  - `pandas` **(for data handling)**
  - `networkx` **(for graph and Dijkstra algorithm)**
  - `folium` **(for map visualizations)**
  - `matplotlib` **(for plotting routes if needed)**
  - `sqlite3` **(for database operations)**
  - `colorama` **(for colored command-line output)**
- **Other Tools:**
  - **Kaggle CLI (for dataset download and access)**

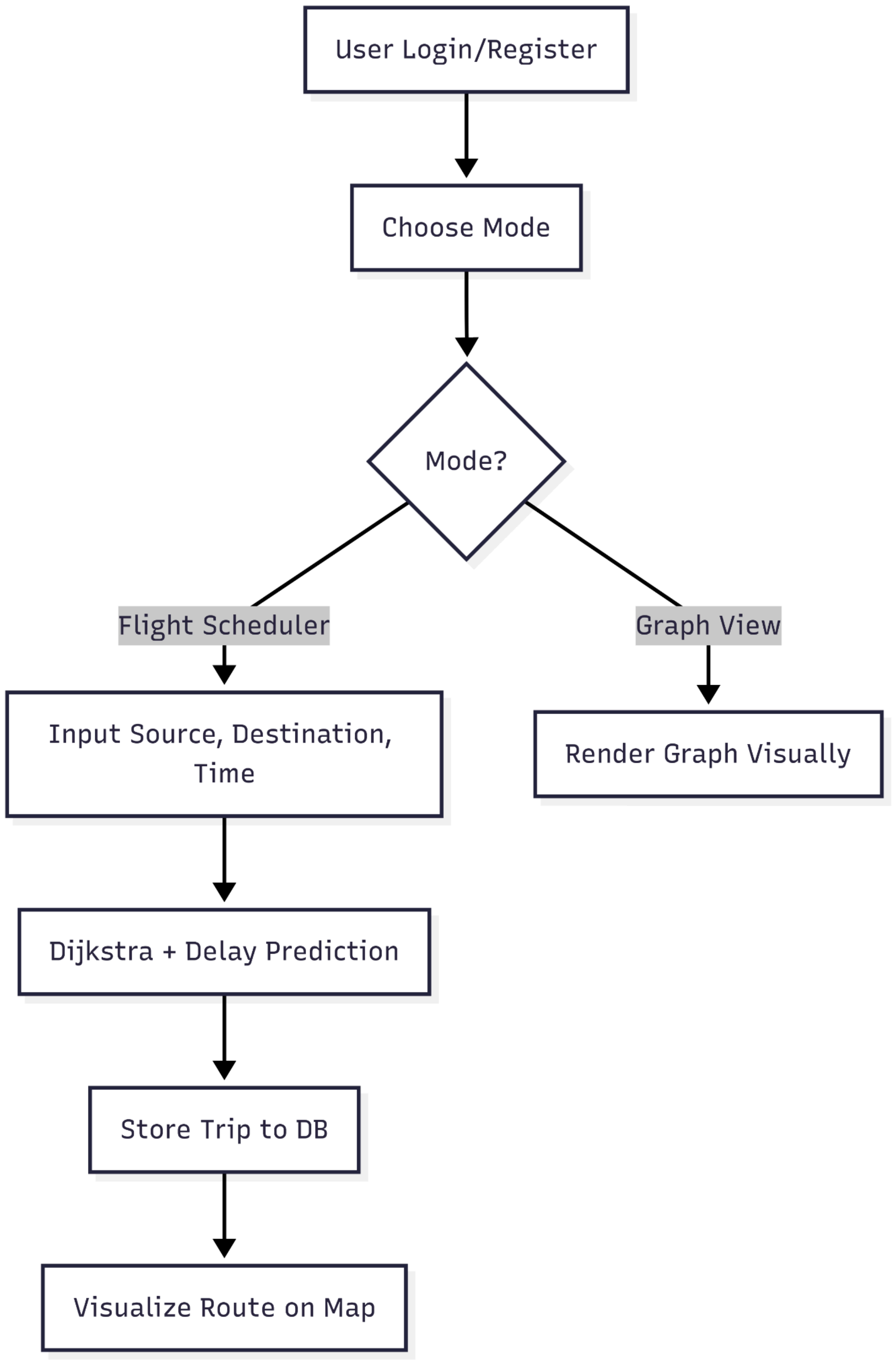


Figure 3.5: This flowchart outlines the complete functionality of *Flight Agenda – Intelligent Route Planner*.

Chapter 4: Implementation

Table 2: Tools Used

Tool/Technology	Purpose
Python 3.10+	Core development language
SQLite	Lightweight relational database for storing users and trip data
Pandas	Reading and manipulating CSV flight datasets
NetworkX	Graph creation and Dijkstra's shortest path algorithm
Matplotlib / Folium	Visualizing flight paths on maps
Scikit-learn	Delay prediction using machine learning models
Kaggle Dataset (Indian Airlines)	Real-world flight data used for graph creation
Command Line Interface (CLI)	User interface to interact with the application
Anaconda	Python environment and package manager
VS Code / Jupyter	IDE used for development and prototyping

4.2 Methodology

Our implementation follows a modular, CLI-driven, graph-based design using real airline data to simulate and predict the most efficient flight path. Here's the step-by-step methodology:

1. **User Authentication:**
    - Users can register/login.

    - Credentials are securely stored in a SQLite database.

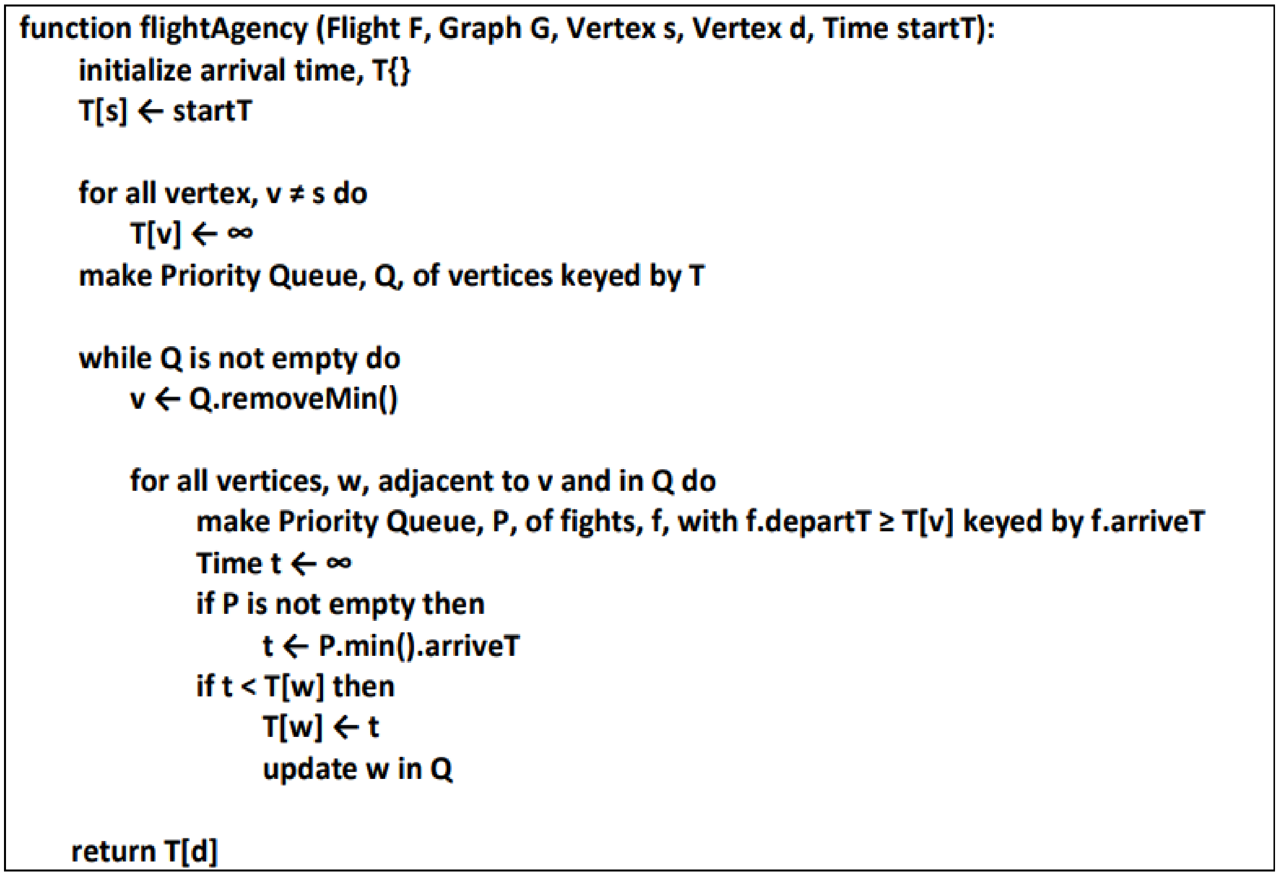
2. **Flight Graph Creation:**
    - Data is loaded from a Kaggle Indian Airlines dataset (CSV).
    - Nodes = Airports, Edges = Routes.
    - A weighted directed graph is created using NetworkX.
    - Weights represent travel time or cost (user's choice).

3. **Shortest Path Calculation:**
    - Dijkstra's algorithm is used to compute the shortest path based on either cost or time.
    - Predicted delays are optionally added using a dummy ML model for realism.

4.2.1 Algorithm: The algorithm for this problem is the slight modification of Dijkstra's Algorithm. Here we have the database about airports and the flights. There is information about flight number, origin airport and destination, the flights have departure and arrival time. So, given origin and destination airports and start time we need to make our data structure properly to keep the data and find the shortest arrival time. Here the di graph is used where the airports are the vertices and flights are di-edges with weights: departure time and arrival time. The distance should be the function of earliest arrival times at airports. Then based upon those earliest arrival time, we use a minimum priority queue for airports. For the initial origin airport let the earliest arrival time be start time and for others it will be infinity in the beginning. Now, until the queue is not empty, adjacent loop is to be formed where we can only select the flights which can be caught, and we also want the flight with minimum arrival time. So, for that we create another priority queue of flights where its departure time should be later than arrival time of another flight at the airport. And we also use another variable time, which is used to update the flight priority queue. And finally, we perform relaxation, where if the above time is less than the earliest arrival time at adjacent airport, then we change the earliest arrival time of the airport to be the time and update in the airport queue accordingly.

Pseudocode:

From: http://www.csl.mtu.edu/cs2321/www/newLectures/30_More_Dijkstra.htm



```
function flightAgency (Flight F, Graph G, Vertex s, Vertex d, Time startT):
    initialize arrival time, T{}
    T[s] ← startT

    for all vertex, v ≠ s do
        T[v] ← ∞
    make Priority Queue, Q, of vertices keyed by T

    while Q is not empty do
        v ← Q.removeMin()

        for all vertices, w, adjacent to v and in Q do
            make Priority Queue, P, of fights, f, with f.departT ≥ T[v] keyed by f.arriveT
            Time t ← ∞
            if P is not empty then
                t ← P.min().arriveT
            if t < T[w] then
                T[w] ← t
                update w in Q

    return T[d]
```

4. **Trip Storage:**

    ○ The final path, delays, and estimated arrival times are stored in the database under the user’s ID.

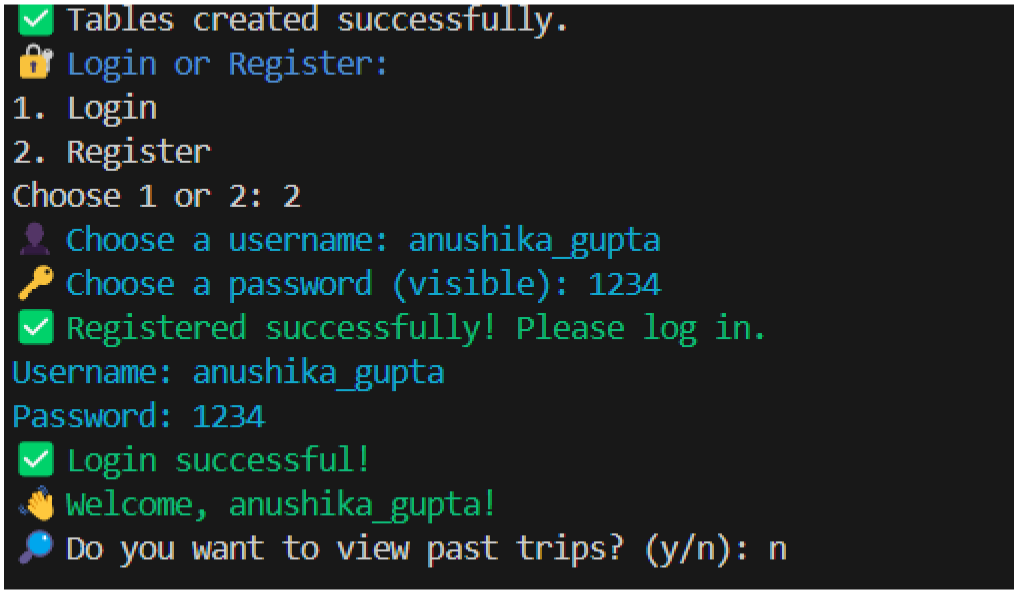
5. **Past Trips Retrieval:**

○ Users can view their previous trips after login.

6. **Map Visualization:**

   ○ The route is displayed using Folium as an interactive HTML map.

4.3 Modules / Screenshots



```
✅ Tables created successfully.
🔐 Login or Register:
1. Login
2. Register
Choose 1 or 2: 2
👤 Choose a username: anushika_gupta
🔑 Choose a password (visible): 1234
✅ Registered successfully! Please log in.
Username: anushika_gupta
Password: 1234
✅ Login successful!
👋 Welcome, anushika_gupta!
🔎 Do you want to view past trips? (y/n): n
```

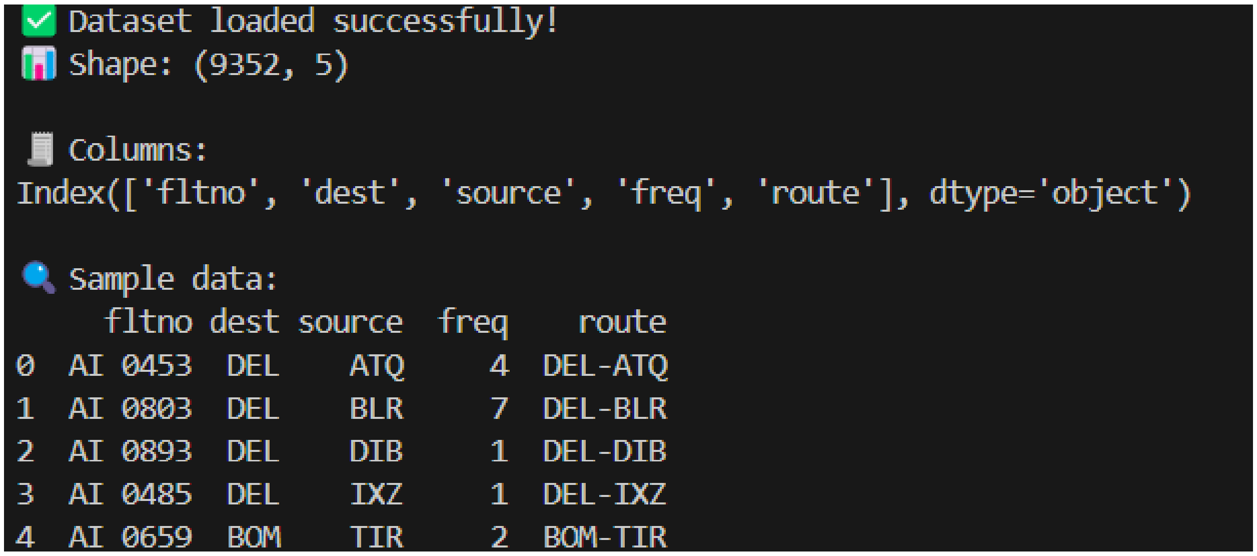
Figure 4.1: User login CLI

4.3.1 User Authentication

- **Handles user registration and login.**
- **Database: flight_users.db**

4.3.2 Dataset Loading and Graph Construction

- **Loads Indian-Airlines-Dataset.csv**
- **Creates graph using NetworkX.**



```
✅ Dataset loaded successfully!
📊 Shape: (9352, 5)

📋 Columns:
Index(['fltno', 'dest', 'source', 'freq', 'route'], dtype='object')

🔍 Sample data:
     fltno dest source  freq    route
0  AI 0453  DEL    ATQ     4  DEL-ATQ
1  AI 0803  DEL    BLR     7  DEL-BLR
2  AI 0893  DEL    DIB     1  DEL-DIB
3  AI 0485  DEL    IXZ     1  DEL-IXZ
4  AI 0659  BOM    TIR     2  BOM-TIR
```

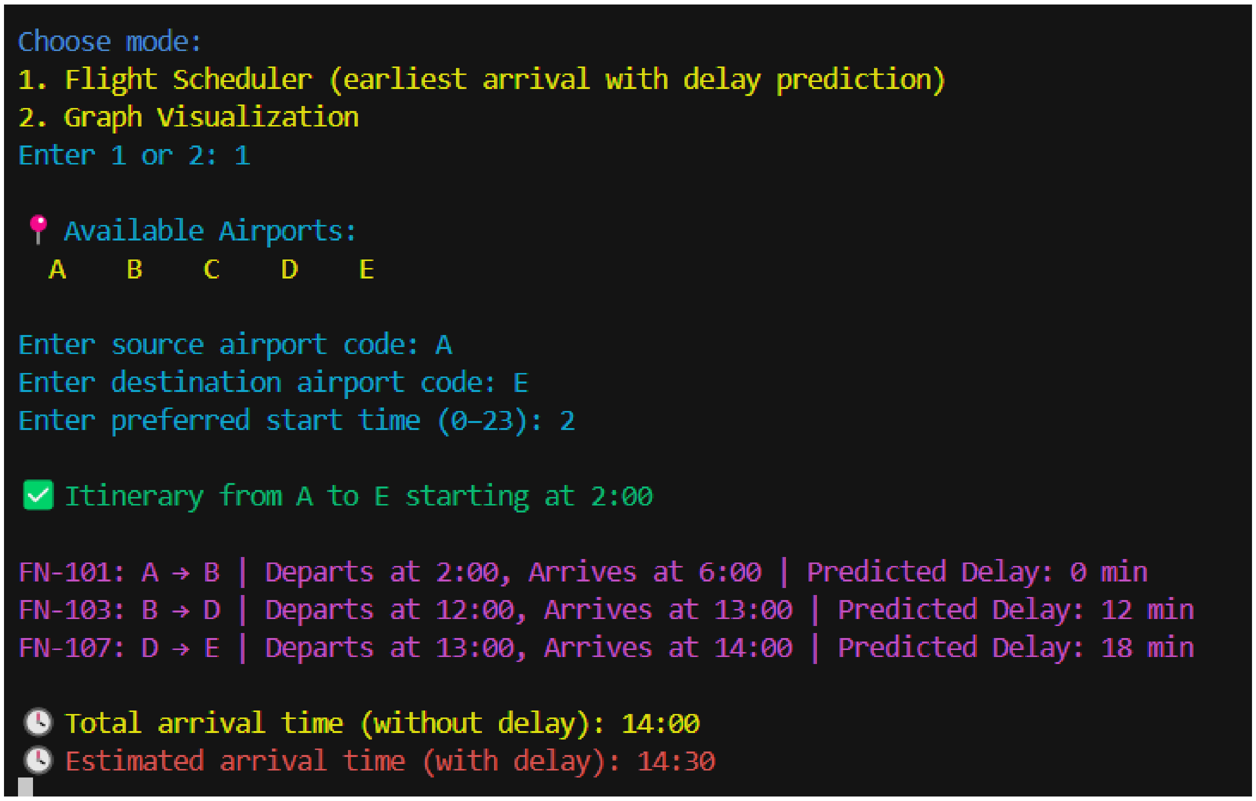
Figure 4.2: Dataset loaded (via inspect_dataset.py)

4.3.3 Route Input + Path Computation

- **Takes user's source, destination, and time preference.**

- **Calculates best route using Dijkstra's algorithm.**

Main Program: So, for our main program we made following setup: Airport A, B, C, D, E as the vertices of the Graph G. The list of 7 flights: FN-101: From airport A to B with departure time 02:00 and arrival time 06:00 FN-102: From airport A to C with departure time 02:00 and arrival time 08:00 FN-103: From airport B to D with departure time 12:00 and arrival time 13:00 FN-104: From airport B to E with departure time 11:00 and arrival time 17:00 FN-105: From airport C to B with departure time 09:00 and arrival time 10:00 FN-106: From airport C to D with departure time 06:00 and arrival time 10:00 FN-107: From airport D to E with departure time 13:00 and arrival time 14:00 The flights are the edges. Airport A as the origin airport. Airport E as the destination airport. 02:00 as the start time.

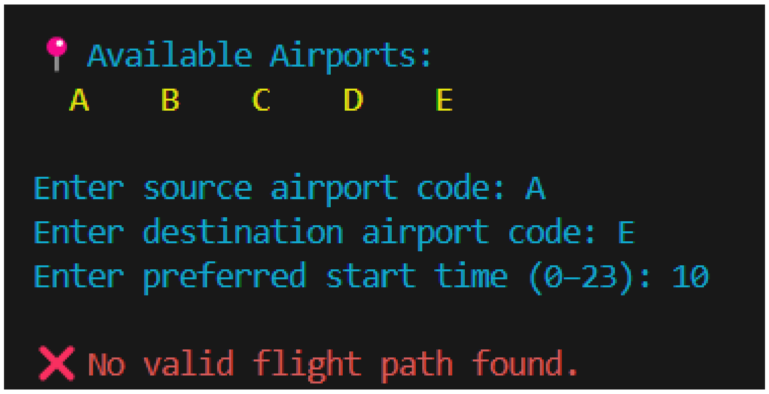


```
Choose mode:
1. Flight Scheduler (earliest arrival with delay prediction)
2. Graph Visualization
Enter 1 or 2: 1

📍 Available Airports:
  A    B    C    D    E

Enter source airport code: A
Enter destination airport code: E
Enter preferred start time (0-23): 2

✅ Itinerary from A to E starting at 2:00

FN-101: A → B | Departs at 2:00, Arrives at 6:00 | Predicted Delay: 0 min
FN-103: B → D | Departs at 12:00, Arrives at 13:00 | Predicted Delay: 12 min
FN-107: D → E | Departs at 13:00, Arrives at 14:00 | Predicted Delay: 18 min

🕓 Total arrival time (without delay): 14:00
🕓 Estimated arrival time (with delay): 14:30
```

Figure 4.3: CLI taking airport inputs and display of itinerary with delays

We got the output that, the earliest arrival time for the airport E from airport A is 14:00. So, starting at 02:00 from airport A the shortest path to reach airport E, the earliest arrival time is 14:00.

Similarly, if we had tried the same setup with start time 10:00 then we get the result as:



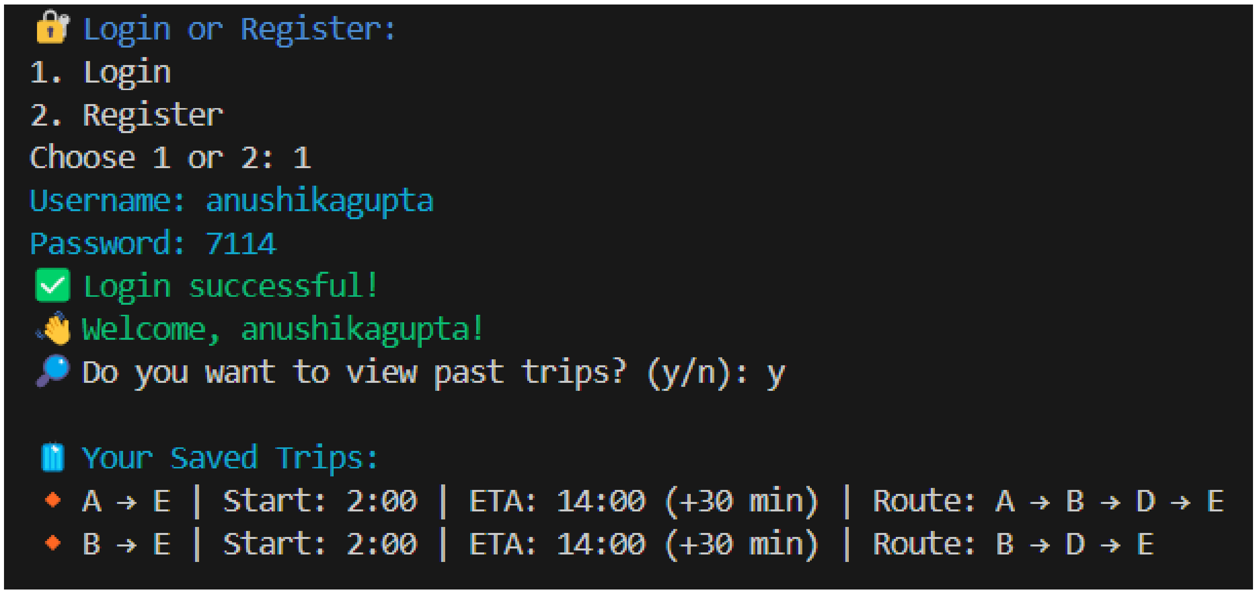
```
📍 Available Airports:
  A    B    C    D    E

Enter source airport code: A
Enter destination airport code: E
Enter preferred start time (0-23): 10

❌ No valid flight path found.
```

We got the output as, No flight from airport A to airport E after 10:00. Also, we would get a similar result, if we try to go from origin to destination airport where there is no flight like from B to A.

4.3.4 Trip Saving + History

- **Saves the current trip to flight_data.db**

- **Allows users to view previous trips.**



```
🔐 Login or Register:
1. Login
2. Register
Choose 1 or 2: 1
Username: anushikagupta
Password: 7114
✅ Login successful!
👋 Welcome, anushikagupta!
🔎 Do you want to view past trips? (y/n): y

📋 Your Saved Trips:
🔸 A → E | Start: 2:00 | ETA: 14:00 (+30 min) | Route: A → B → D → E
🔸 B → E | Start: 2:00 | ETA: 14:00 (+30 min) | Route: B → D → E
```

Figure 4.4: ***Table of past trips***

4.3.5 Route Visualization

- **Generates interactive HTML maps using Folium.**

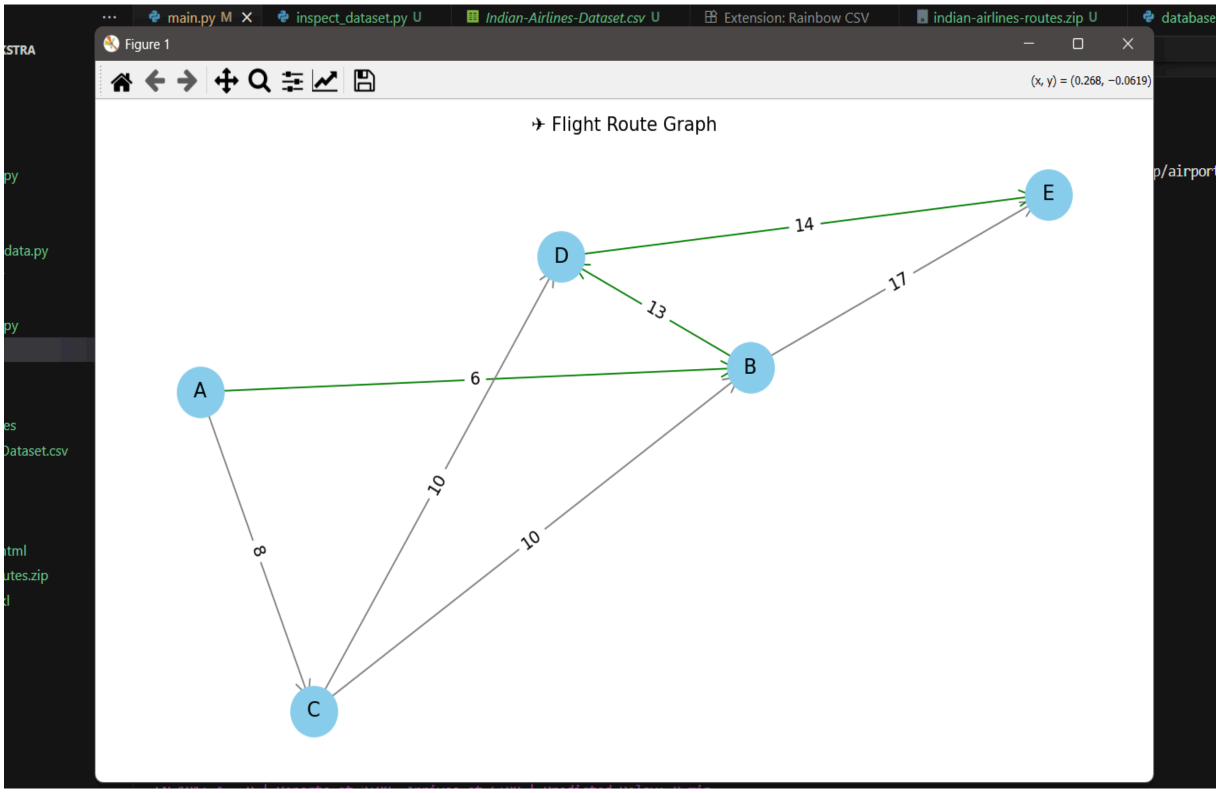


Figure 4.5 : Graph visualization

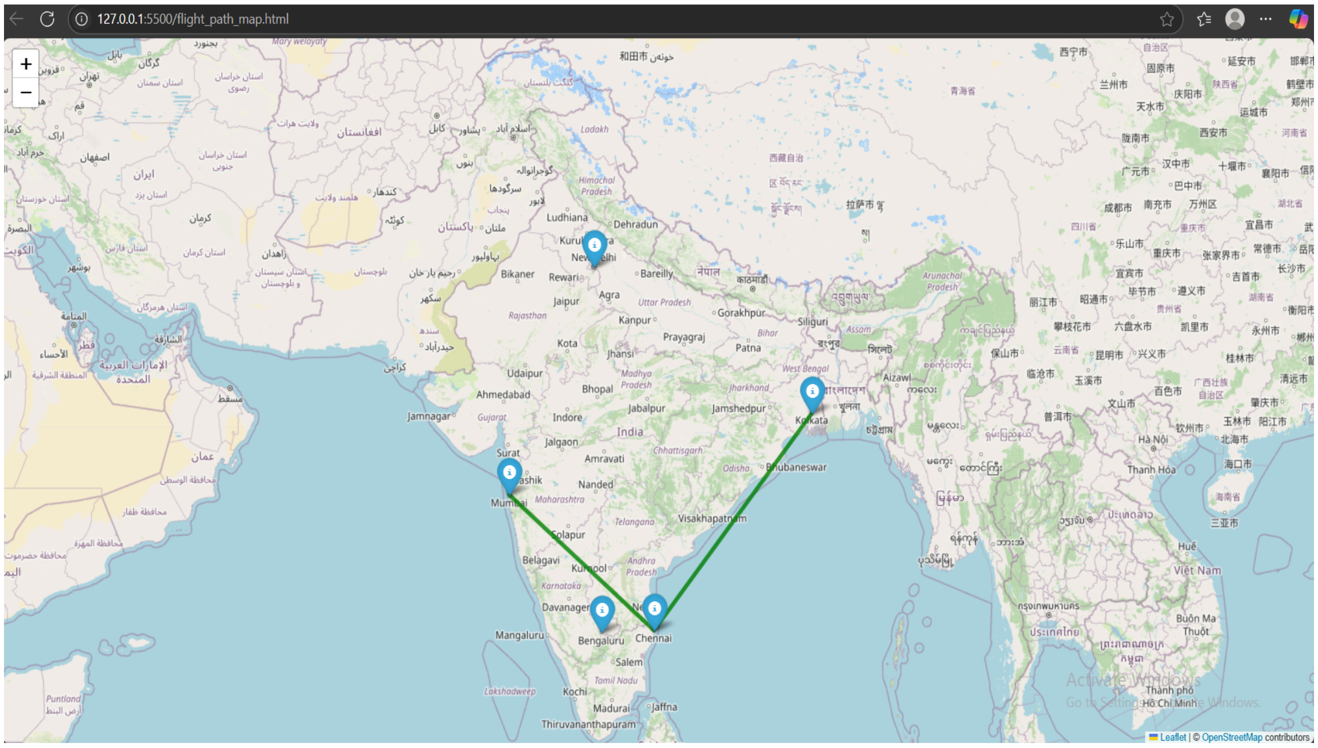

127.0.0.1:5500/flight_path_map.html
Mumbai
Bengaluru
Chennai
Kolkata
New Delhi
Leaflet | © OpenStreetMap contributors

Figure 4.6 ***Map in browser or image of flight_path_map.html***

Disclaimer:

The screenshots presented in this chapter are based on a simplified version of the system using a limited dataset. This has been intentionally done to ensure clear understanding of the core concepts such as user login, trip scheduling, shortest path calculation, and route visualization.

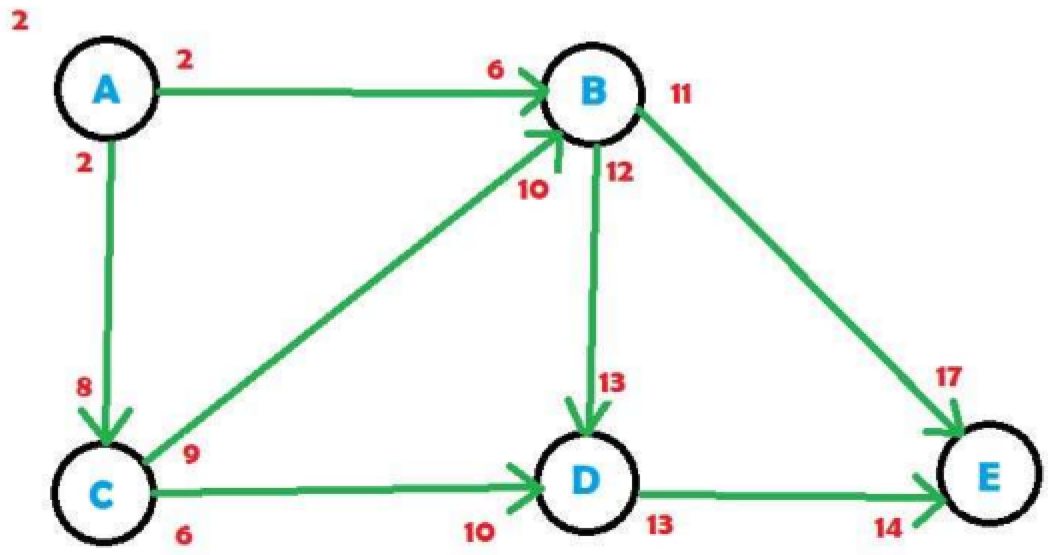
To avoid overwhelming complexity during explanation, the current version uses a small, easy-to-follow dataset instead of the full-scale Kaggle or real-time flight data. However, the system is fully designed and structured to scale with real-world datasets such as those from Kaggle or APIs like OpenSky.

These illustrative examples allow for easier interpretation of logic, flow, and functionality—particularly during evaluation or academic review.

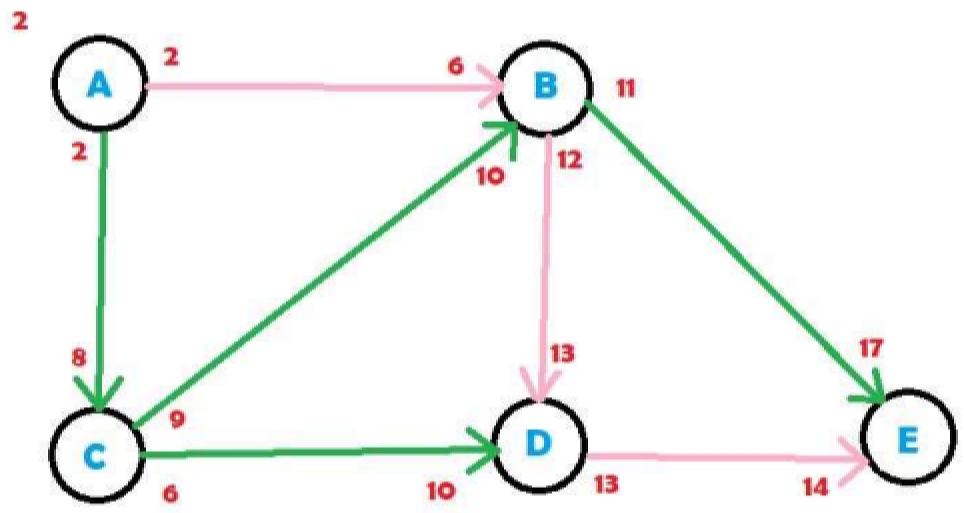
Chapter 5 : Results and Discussion

5.1 Output / Report

Analysis: So, based upon the above setup, we would have got a graph like below:



In above graph, there is a flight that leaves from airport A at 2 and reaches C at 8 and another flight leaves airport A at 2 and reach airport B at 6 and so on. From airport A, we can go to airport B and C. The best possible arrival time for airport B and C would be 6 and 8 respectively. The flight from C to D departs at 6. So, starting at time 2 and reaching airport C at 8, there is no way we can catch that flight. So, from C, now we can only go to B. We can reach airport B either from A or from C, but the earliest arrival time would be from airport A at 6. Similarly, from B we can go to D or E and so on. So, finally using the algorithm the best possible shortest pathway we can reach airport E from airport A would be via route A – B – D – E.



And, using those flights, we would reach airport E at 14, which is the answer given by our program.

After successful implementation of the key modules of our airport route planner, the application was able to:

- Allow users to register and login securely.
- Display available Indian airports dynamically from a real-world dataset (Kaggle).
- Plan a flight route using Dijkstra's algorithm based on earliest arrival time.
- Display the itinerary with estimated delays (using a dummy ML delay model).
- Visualize the planned route on a map using HTML (via Folium).
- Store user trips in a database, and fetch past trips at login.

5.2 Challenges Faced

During the development of the project, the following challenges were encountered:

Challenge	**Description**
Database Conflict	Multiple `.db` files were unintentionally created due to inconsistent pathing. This caused tables like `users` and `trips` to go missing during login.
Kaggle CLI Setup	Configuring the Kaggle API and ensuring that the token (`kaggle.json`) was correctly placed in the `.kaggle` directory took some time and debugging.
Dataset Limitations	Indian airline datasets are not as exhaustive as US datasets, which limits coverage and realism for ML training.
Map Rendering	Folium maps require HTML rendering. Some learners initially expected them to appear directly in the CLI.
Time & Cost Integration	Modifying the graph to support both "earliest arrival time" and "least cost" required a custom weighting strategy and conditional logic in the pathfinding algorithm.

5.3 Learnings

The project helped us gain practical knowledge in several key areas:

5.3.1 Technical Skills

- Python basics to advanced (modular code, OOP concepts)
- File handling, working with large CSVs using Pandas
- Database operations using SQLite3
- Pathfinding algorithms (specifically, Dijkstra's Algorithm)

- Data visualization with Folium
- CLI tools & utilities (Kaggle CLI, command line debugging)
- API configuration and project scalability thinking

5.3.2 Conceptual Understanding

- How graph theory applies to real-world problems like flight route planning.
- Delay prediction modeling — integrating machine learning logic into practical tools.
- Handling real-world datasets that are messy or incomplete.
- Importance of clean folder structure and consistent file paths in multi-file projects.

Chapter 6: Conclusion

6.1 Summary:

Title: *Airport Route Planner*
Duration: 6 Weeks (June–July 2025)
Training Platform: CodeQuest – DSA Bootcamp (50+ Hours)
Technologies Used: C++, SQLite, Python (for dataset inspection), Graph Algorithms (Dijkstra), Map Visualization
Dataset: Indian Airlines Dataset (Kaggle) – *9352 rows × 5 columns*
Mentor: Prabhjeet Kaur (LPU)

This project, developed during a 6-week intensive summer training program, aims to solve the real-world challenge of finding the most efficient route between two airports based on user preference for either **minimum time** or **minimum cost**.

The system simulates an intelligent travel planner that uses **Dijkstra's algorithm** to find the shortest path between airports in a network modeled as a **weighted graph**. It leverages a **real-world dataset** of Indian domestic airlines, and integrates several advanced features such as:

- **User login and registration**
- **Graph-based itinerary planner using Dijkstra's algorithm**
- **Trip storage and history using SQLite**
- **Map-based visualization of the final route**
- **Interactive terminal interface for planning trips**

The current version is terminal-based, fully functional, and focused on algorithmic accuracy and data integration. Features like **real-time flight tracking**, **delay prediction using ML**, **ticket price optimization**, and a **web-based UI using Flask or Streamlit** are outlined in the Future Scope.