PROJECT REPORT

Airline Route Management System

1. Abstract

This project implements an Airline Route Management System that allows users to create, analyze, and compute optimal flight routes between airports. The system models airports as nodes and routes as weighted edges in a graph. It supports efficient route queries using Dijkstra's algorithm and provides additional tools to visualize the connectivity structure. The program enables adding airports, adding routes, viewing all airports and routes, and computing the shortest path between any two airports based on distance. The primary aim is to demonstrate practical application of data structures—especially graphs—along with file handling and interactive menu-driven program design. The results show that real-world-like routing problems can be efficiently modeled and solved using the implemented graph algorithms.

2. Introduction

Air transportation networks can be effectively represented using graph data structures. Airports serve as vertices, and direct flight paths act as weighted edges. Efficient management of such networks requires dynamic addition of nodes, storage of route information, and computation of shortest paths based on cost or distance.

This project simulates a simplified version of this system using C++, emphasizing learning outcomes related to graphs, adjacency lists, file handling, and shortest path algorithms. The project is designed to provide hands-on experience with implementing theoretical concepts from data structures.

3. Problem Statement & Objectives

Problem Statement:

To design and implement a program that manages airline routes and computes optimal paths between airports using efficient data structures and algorithms.

Objectives:

- Implement an adjacency-list-based graph to store airports and routes.
- Enable addition of airports and routes dynamically through a user interface.
- Display airports and available direct routes.
- Implement Dijkstra's algorithm to compute the shortest path between two airports.
- Provide clean, modular, and maintainable code using Object-Oriented Programming principles.

---

4. Methodology / System Design

The system is built around a **Graph** class that stores all nodes (airports) and edges (routes).

An **adjacency list** is used: each airport maintains a list of connected airports along with the distance of the route.

The program follows a **menu-driven design**, allowing the user to select operations such as adding airports, adding routes, and computing shortest paths.

Design Components:

- **Graph Representation:**

  Implemented as unordered_map<string, vector<pair<string,int>>>, mapping each airport to its outgoing routes.

- **Dijkstra's Algorithm:**
  A priority queue (min-heap) is used to continuously select the smallest-weight node. The algorithm computes the shortest path from a source airport to a destination based on route distance.

Workflow Summary:

1. Initialize graph.
2. Load previous airport and route data.
3. User selects an operation.
4. System performs requested function (add, display,remove, shortest path).

5. On exit, updated data is saved.

---

5. Implementation Details

The project contains several core modules:

Graph Class

Handles storage and management of the network:

- addAirport(string): Adds a new airport.
- addRoute(a, b, dist,cost): Adds a weighted edge.
- removeAirport(string): Remove airport
- removeRoute(src,dest):Remove route.
- getAirports(): Returns all the airports
- printGraph(): Returns the entire graph network
- fromData,toJSON,fromJSON: used for saving and loading the data in JSON format

The adjacency list ensures memory efficiency and fast traversal.

Shortest Path Computation

Done using one of two algorithms:

- Dijkstra:

  Implemented using a priority queue (min-heap) to always expand the nearest unvisited airport. Maintains distance and parent maps for path reconstruction. Efficiently computes shortest paths

- Bellman–Ford Implementation

  Iteratively relaxes all edges for V−1 rounds, allowing detection of shortest paths even with negative weights.. Useful for scenarios where route costs may include penalties. Provides a reliable alternative to Dijkstra when edge weights are not guaranteed positive although that is not the case here ,here we have just used it as an alternative for dijkstra algorithm

Menu-Driven Interface

Users interact through numerical options:

1. Add Airport
2. Add Route
3. Display Airports
4. Display Routes
5. Find Shortest Path
6. Exit

Each option triggers a corresponding function inside the graph class.

---

6. Results and Analysis

- The system correctly identifies the shortest path between any two connected airports.
- Adding airports/routes dynamically updates the graph and persists the data.
- The Dijkstra implementation handles large graph sizes efficiently due to the priority queue optimization.
- Route visualization displays all direct connections, improving system clarity.
- Testing with multiple airports and routes confirmed correctness of path reconstruction and distance calculation.

This demonstrates that graph algorithms can effectively handle transportation-network-type problems. The implementation is robust and scalable for larger datasets.

---

7. Conclusion

The Airline Route Management System successfully models an airport network using graph data structures and applies Dijkstra's algorithm for optimal route computation. The project reinforced key concepts such as adjacency lists, file handling, modular code structure, and algorithmic thinking. The outputs validate the efficiency and reliability of the system. Overall, the project meets its objectives and provides a strong foundation for further extensions such as cost-based routing, visualization, or real-time data integration.