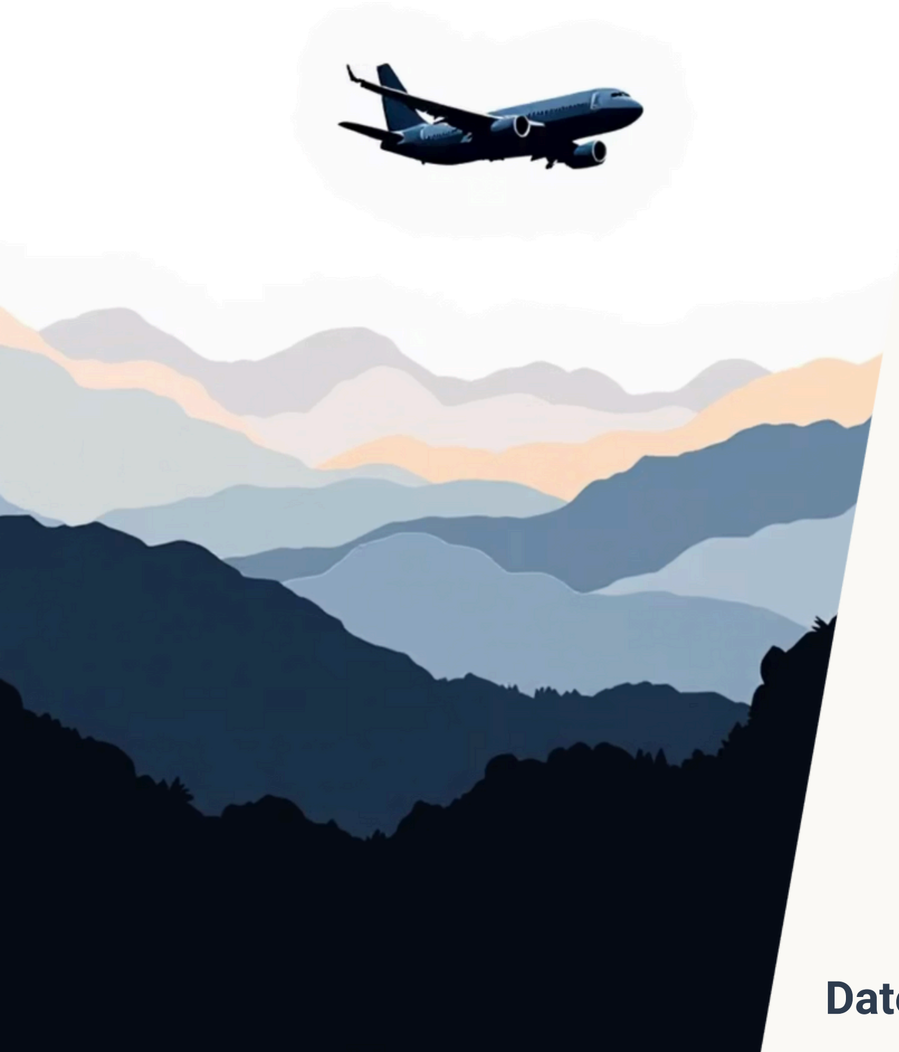


Airline Route Planner Using Graph Data Structure

Group Project – Data Structures & Algorithms

Lecturer: Mr. Korat Natt
Room-209
Gen11-SEG4

Group Members:

- Seng Dina
- Chay SengHap
- Ly SrongChhay
- Yann SokMeng
- Roth Sorayuth
- Yun EyChhean

Date: 16 Dec 2025

Contents

The Challenge of Efficient Air Travel

Connecting Cities Efficiently

Airlines face the complex task of connecting passengers between various cities. The challenge arises because direct flights are not always available for every airport pair, making efficient route planning a critical operational need.

Optimizing Travel Routes

Determining the most efficient travel route, whether by shortest distance or fastest time, requires sophisticated tools. Our airline route planner is designed to address this by:

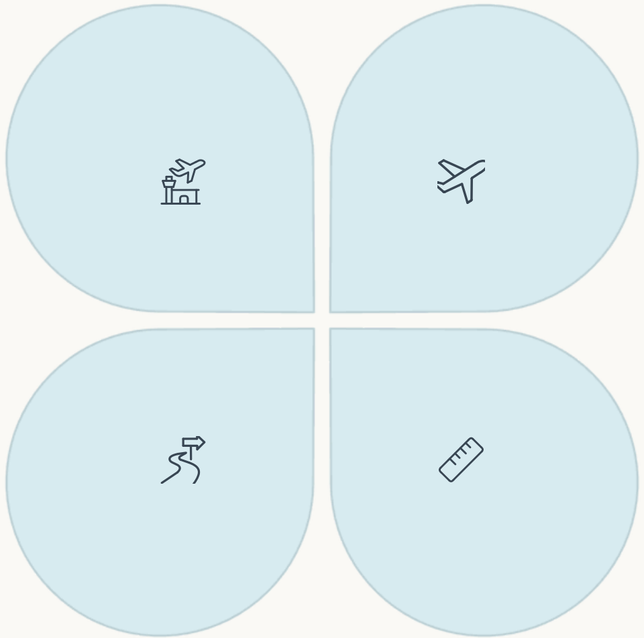
- Finding the fastest or shortest route between any two points.
- Selecting optimal connecting flights for complex route.
- Identifying all reachable airports from a given origin.

Why Graphs Are Essential for Route Planning

Graph theory provides an intuitive and powerful framework for modeling airline networks. This problem fits perfectly with graph modeling due to several key correspondences:

Airports = Nodes

Each airport in the network is represented as a node or vertex in the graph.



Flight Routes = Edges

Direct flight connections between airports are represented as edges or links connecting the respective nodes.

Two-Way Flights = Undirected

Since flights often operate in both direction, edges are undirected, reflecting the flow from origin to destination.

Distance/Time = Weight

The distance or travel time associated with each flight route acts as the "weight" of the edge, indicating cost or duration.

By leveraging graph properties, we can easily visualize and compute critical aspects of travel, including shortest paths, network connectivity, the number of stops required, and all possible routes between destinations.

Our Graph Design Overview

Our proposed graph model for the airline route planner is specifically designed to handle the complexities of flight networks. Here's a breakdown of its components:

Nodes: Airports

Each airport (e.g., KTI,SAI,SGN) is represented as a distinct node.

Graph Type: Undirected & Weighted

The graph is undirected, as flights operate in both directions between connected airports.

Edges: Undirect Flights

Edges represent direct flight connections between airports. Since flights exist in both directions, edges are undirected.

Edge Weight: Distance

The weight of each edge represents the distance in kilometers between connected airports.

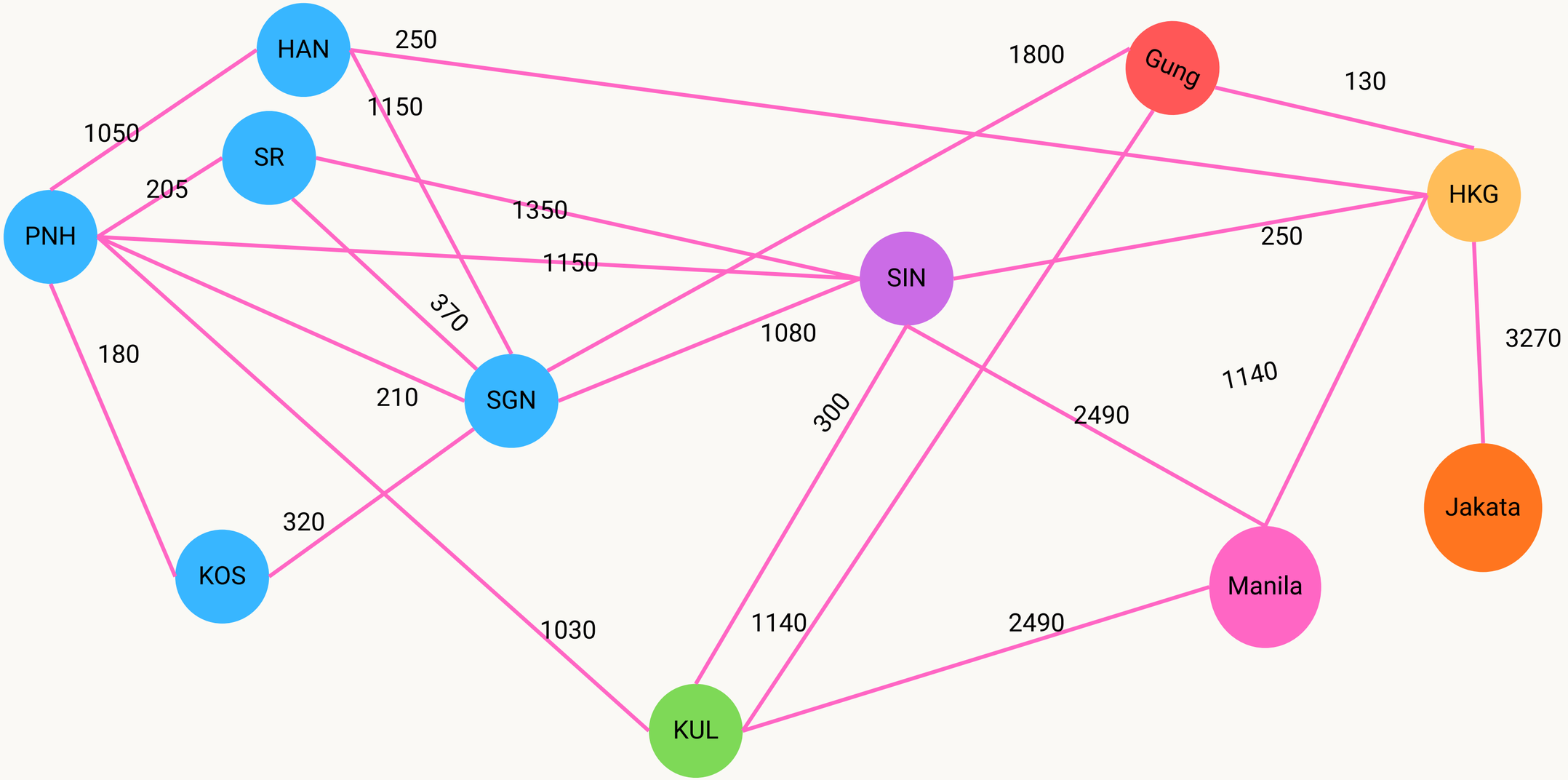
Representation Methods

To efficiently store and manipulate this data, we utilize a dual representation:

- **Adjacency List:** Ideal for sparse graphs, providing quick access to a node's neighbors.
- **Adjacency Matrix:** Useful for dense graphs and quick checks for direct connections.

Illustrative Example Graph Structure

To better understand our graph model, consider the following example illustrating airports and their direct flight distances:



Adjacency List and Matrix Representations

Our graph is represented using both an adjacency list and an adjacency matrix to leverage the strengths of each method for different operations.

Adjacency List

```
PNH: REP(230), KOS(180), HAN(1050), SGN(210), KUL(1030),
SIN(1150)
REP: PNH(230), SGN(420), SIN(1350)
KOS: PNH(180), SGN(320)
SGN: PNH(210), REP(420), KOS(320), HAN(1150), SIN(1080),
CAN(1800)
HAN: PNH(1050), SGN(1150)
SIN: PNH(1150), REP(1350), SGN(1080), KUL(300), HKG(2550)
KUL: PNH(1030), SIN(300), MNL(2490)
MNL: KUL(2490), HKG(1140)
HKG: SIN(2550), MNL(1140), CAN(130), CGK(3270)
CAN: SGN(1800), HKG(130)
CGK: HKG(3270)
```

The adjacency list provides an efficient way to represent sparse graphs by listing only the connections that exist for each node, making it ideal for iterating through a node's neighbors.

Adjacency Matrix

FROM \ TO	PNH	REP	KOS	SGN	HAN	SIN	KUL	MNL	HKG	CAN	CGK
PNH	0	230	180	210	1050	1150	1030	0	0	0	0
REP	230	0	0	420	0	1350	0	0	0	0	0
KOS	180	0	0	320	0	0	0	0	0	0	0
SGN	210	420	320	0	1150	1080	0	0	0	1800	0
HAN	1050	0	0	1150	0	0	0	0	0	0	0
SIN	1150	1350	0	1080	0	0	300	0	2550	0	0
KUL	1030	0	0	0	0	300	0	2490	0	0	0
MNL	0	0	0	0	0	0	2490	0	1140	0	0
HKG	0	0	0	0	0	2550	0	1140	0	130	3270
CAN	0	0	0	1800	0	0	0	0	130	0	0
CGK	0	0	0	0	0	0	0	0	3270	0	0

The adjacency matrix, a 11x11grid in this case, stores the direct distances, with '0' indicating no direct flight. It's efficient for checking the existence and weight of an edge between any two nodes quickly.

Key Algorithms for Route Optimization

To effectively navigate and optimize routes within our airline graph, we employ several fundamental graph algorithms, each serving a specific purpose in route planning.

Dijkstra’s Algorithm

This algorithm is crucial for finding the **shortest path** from a single source node to all other nodes in a weighted graph. It's ideal for determining the most fuel-efficient or fastest routes.

Breadth-First Search (BFS)

BFS is used to explore a graph level by level. In our context, it helps us:

- Check for reachability between airports.
- Determine the minimum number of flight connections (stops) required.

Depth-First Search (DFS)

DFS explores as far as possible along each branch before backtracking. It's invaluable for:

- Exploring all possible paths from an airport.
- Identifying isolated airports or subnetworks within the larger airline system.

Together, these powerful algorithms are the reason our route planner gives customers the fastest, most reliable, and best-connected flight options.

Dijkstra's Algorithm: Workflow & Pseudocode

Dijkstra's Algorithm is the cornerstone of our shortest path calculation. Its systematic approach guarantees finding the optimal route in weighted graphs. Below is the workflow:

Start from Source

Choose a source airport (e.g., Airport A) from which all shortest paths will be calculated.

Initialize Distances

Set the distance to the source as 0 and to all other airports as infinity (∞).

Use Priority Queue

Maintain a priority queue to efficiently extract the node with the smallest known distance.

Relax Edges

For each neighbor of the selected node, update its distance if a shorter path is found through the current node.

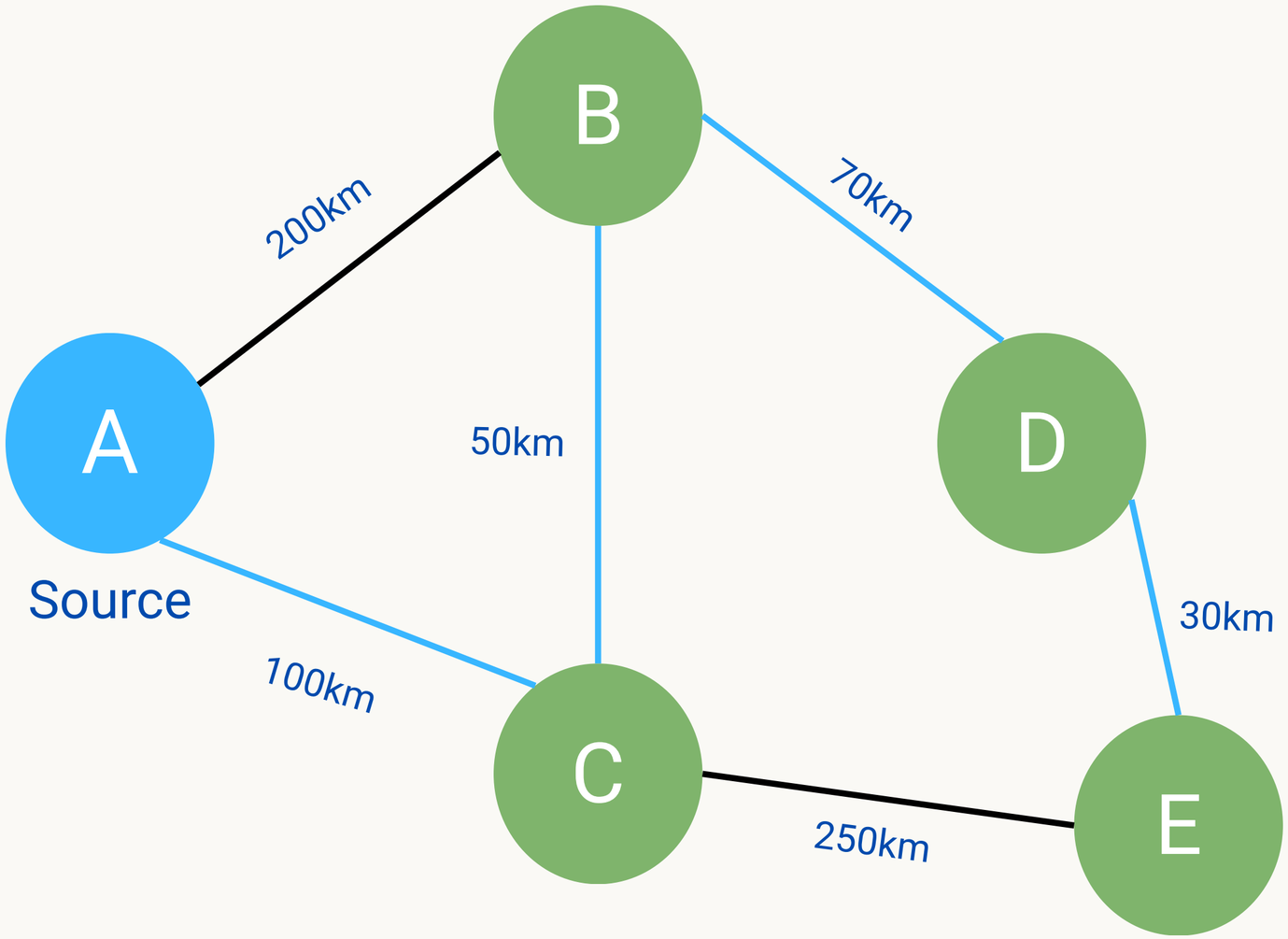
Iterate

Repeat the process until all nodes have been visited or the priority queue is empty.

Simplified Pseudocode

```
DIJKSTRA(Graph, source):

    for each vertex v in Graph:
        distance[v] = ∞
        visited[v] = false

    distance[source] = 0

    create priority_queue PQ
    insert (0, source) into PQ

    while PQ is not empty:

        (dist, u) = pop(PQ)   // pop the node with minimum distance

        if visited[u] = true:
            continue

        visited[u] = true

        for each edge (u, v, w) in Graph:
            if visited[v] = false AND distance[u] + w < distance[v]:
                distance[v] = distance[u] + w
                insert (distance[v], v) into PQ

    return distance
```

Dijkstra's Algorithm: Workflow Graph Explaination



Shortest Distances from A

- A → A = 0 km
- A → C = 100 km
- A → B = 150 km
- A → D = 220 km
- A → E = 250 km

Shortest Path Tree

- C: A → C
- B: A → C → B
- D: A → C → B → D
- E: A → C → B → D → E

Key Result

Shortest path from A to E:

A → C → B → D → E (250 km)

Example Input & Expected Output

When the program start

```
=== Airline Route Planner ===

Menu Options:
1. Display all available airports
2. Display all direct routes
3. Find shortest path between two airports
4. Find path with minimum stops
5. Find all possible paths between two airports
6. Exit
Enter your choice (1-6):
```

- At the beginning of the program, the console shows a main menu containing all available features.
- This menu allows the user to view airports, view direct routes, find the shortest path, or exit the program.

PNH : Phnom penh
REP : Siem Reap
KOS : Sihanouk
SGN : Saigon
HAN : Hanoi
SIN : Singapore
KUL : Kuala Lumpur
MNL : Manila
HKG : Hong Kong
CAN : Guangzhou
CGK : Jakarta

Display all available airports

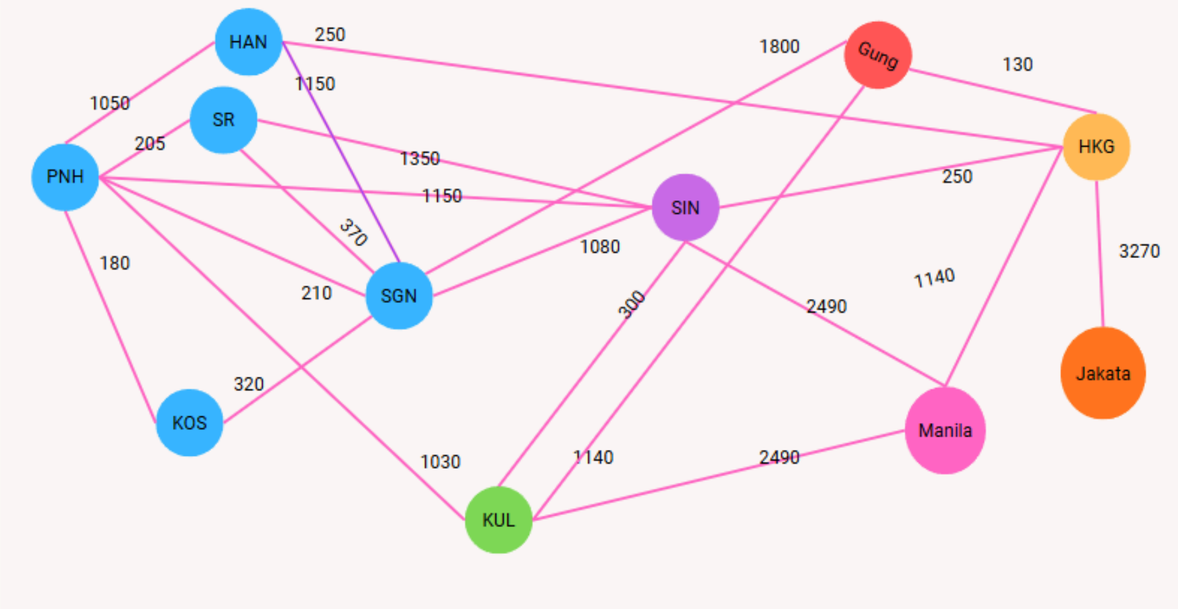
```
Menu Options:
1. Display all available airports
2. Display all direct routes
3. Find shortest path between two airports
4. Find path with minimum stops
5. Find all possible paths between two airports
6. Exit
Enter your choice (1-6): 1
Available Airports: PNH REP KOS SGN HAN SIN KUL MNL HKG CAN CGK
```

Example Input & Expected Output

Display Direct Routes

```
Direct Routes:
PNH <-> REP = 205
PNH <-> KOS = 180
PNH <-> SGN = 210
PNH <-> HAN = 1050
PNH <-> SIN = 1150
PNH <-> KUL = 1030
REP <-> SGN = 370
REP <-> SIN = 1350
KOS <-> SGN = 320
SGN <-> HAN = 1150
SGN <-> SIN = 1080
SGN <-> CAN = 1800
HAN <-> HKG = 250
SIN <-> KUL = 300
SIN <-> MNL = 2490
SIN <-> HKG = 250
KUL <-> MNL = 2490
KUL <-> CAN = 1140
MNL <-> HKG = 1140
HKG <-> CAN = 130
HKG <-> CGK = 3270
```

Display the shortest parrt




```
Enter your choice (1-6): 3
Enter source airport code (e.g., PNH): PNH
Enter destination airport code (e.g., SIN): HKG

Shortest distance from PNH to HKG: 1300
Full route: PNH -> HAN -> HKG
```

Example Input & Expected Output

Find Path With MInimum Stop

```
Enter your choice (1-6): 4
Enter source airport code (e.g., PNH): SGN
Enter destination airport code (e.g., CGK): HKG

Path with Minimum Stops from SGN to HKG:
Minimum stops: 2
Total distance: 1400
Route: SGN -> HAN -> HKG
```

All Possible Path

```
All Possible Paths from PNH to CGK:
Total paths found: 100

Path 1: PNH -> REP -> SGN -> HAN -> HKG -> CGK | Distance: 5245 | Stops: 5
Path 2: PNH -> REP -> SGN -> SIN -> KUL -> MNL -> HKG -> CGK | Distance: 8855 | Stops: 7
Path 3: PNH -> REP -> SGN -> SIN -> KUL -> CAN -> HKG -> CGK | Distance: 6495 | Stops: 7
Path 4: PNH -> REP -> SGN -> SIN -> MNL -> KUL -> CAN -> HKG -> CGK | Distance: 11175 | Stops: 8
Path 5: PNH -> REP -> SGN -> SIN -> MNL -> HKG -> CGK | Distance: 8555 | Stops: 6
Path 6: PNH -> REP -> SGN -> SIN -> HKG -> CGK | Distance: 5175 | Stops: 5
Path 7: PNH -> REP -> SGN -> CAN -> KUL -> SIN -> MNL -> HKG -> CGK | Distance: 10715 | Stops: 8
Path 8: PNH -> REP -> SGN -> CAN -> KUL -> SIN -> HKG -> CGK | Distance: 7335 | Stops: 7
Path 9: PNH -> REP -> SGN -> CAN -> KUL -> MNL -> SIN -> HKG -> CGK | Distance: 12015 | Stops: 8
Path 10: PNH -> REP -> SGN -> CAN -> KUL -> MNL -> HKG -> CGK | Distance: 10415 | Stops: 7
Path 11: PNH -> REP -> SGN -> CAN -> HKG -> CGK | Distance: 5775 | Stops: 5
Path 12: PNH -> REP -> SIN -> SGN -> HAN -> HKG -> CGK | Distance: 7305 | Stops: 6
Path 13: PNH -> REP -> SIN -> SGN -> CAN -> KUL -> MNL -> HKG -> CGK | Distance: 12475 | Stops: 8
Path 14: PNH -> REP -> SIN -> SGN -> CAN -> HKG -> CGK | Distance: 7835 | Stops: 6
Path 15: PNH -> REP -> SIN -> KUL -> MNL -> HKG -> CGK | Distance: 8755 | Stops: 6
Path 16: PNH -> REP -> SIN -> KUL -> CAN -> SGN -> HAN -> HKG -> CGK | Distance: 9465 | Stops: 8
Path 17: PNH -> REP -> SIN -> KUL -> CAN -> HKG -> CGK | Distance: 6395 | Stops: 6
Path 18: PNH -> REP -> SIN -> MNL -> KUL -> CAN -> SGN -> HAN -> HKG -> CGK | Distance: 14145 | Stops: 9
Path 19: PNH -> REP -> SIN -> MNL -> KUL -> CAN -> HKG -> CGK | Distance: 11075 | Stops: 7
Path 20: PNH -> REP -> SIN -> MNL -> HKG -> CGK | Distance: 8455 | Stops: 5
Path 21: PNH -> REP -> SIN -> HKG -> CGK | Distance: 5075 | Stops: 4
Path 22: PNH -> KOS -> SGN -> REP -> SIN -> KUL -> MNL -> HKG -> CGK | Distance: 9420 | Stops: 8
Path 23: PNH -> KOS -> SGN -> REP -> SIN -> KUL -> CAN -> HKG -> CGK | Distance: 7060 | Stops: 8
```

Limitations

While our graph-based airline route planner provides robust solutions for pathfinding, it's important to acknowledge its current limitations of graph modeling.

Ignores Delays & Cancellations

The current model does not account for real-time operational issues such as flight delays, cancellations, or gate changes.

No Consideration for Pricing & Schedules

Ticket prices, varying flight schedules, and seat availability are not factored into the shortest path calculations.

Excludes Real-Time Weather Effects

Dynamic environmental factors, like adverse weather conditions affecting flight paths, are beyond the scope of this model.

Concluding Remarks

- Airline networks by undirected weighted graphs, making graph theory a natural fit for modeling in real life.
- Dijkstra’s algorithm is highly effective in solving shortest path problems, crucial for optimizing travel.
- Graph modeling proves essential for tackling real-life travel routing.


Thank
You