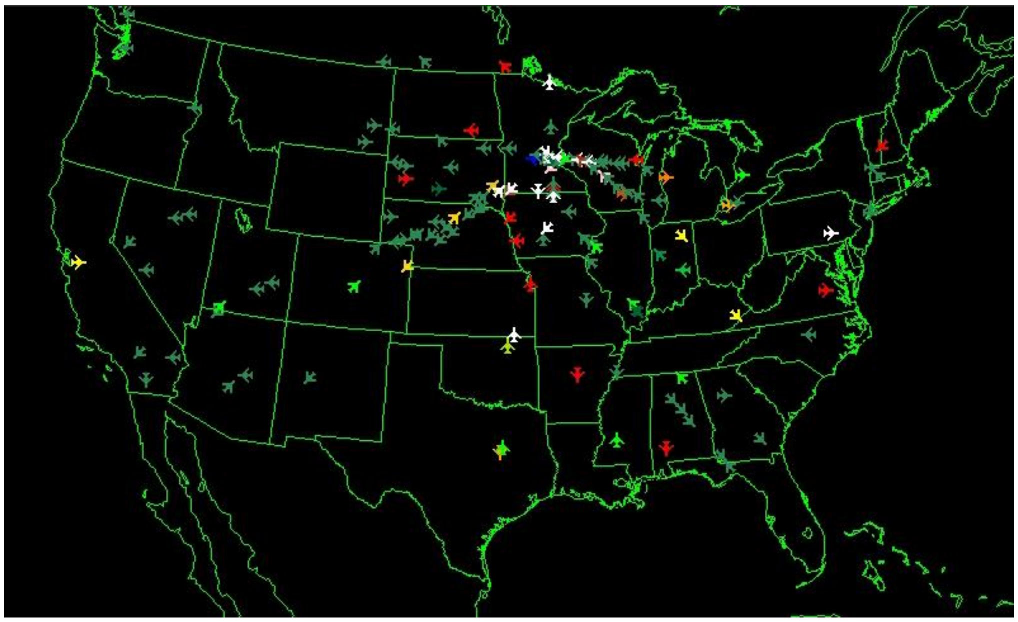
Flight Path Visualizer - SkyNav AI

General Instructions:

- **Group Setup:**
    - Work in groups of 3 (individual submissions allowed in rare cases).
- **Technologies:**
    - **Graphics:** Use SFML or GLUT for the graphical implementation.
    - **Data Structures:**
        - Do not use built-in data structures or the STL library (i.e., no std::vector, std::map, etc.).
        - Use custom-built data structures (e.g., linked lists, queues, stacks, trees and graphs) to handle flight data, layovers, routes, and user preferences.
        - You can use as many data structures you think are needed. All the suggested ones are just one of the many potential options.
- **Marking Criteria:**
    - **Understanding:** Demonstrate a clear understanding of the problem and the applied algorithms/data structures.
    - **Implementation:** Properly implement the custom data structures, algorithms, and graphics visualization.
    - **Innovation:** Include any innovative or extra features that enhance the functionality or user experience.
- **Deadline:** Submit the project by 8th December 2024 11:59 pm. (No Extensions)
- **Additional Guidelines:**
    - Use the provided txt file **Flights.txt** and **HotelCharges.**txt to read data.
    - Provide well-documented code and a pdf of all the GPT prompts you have used while making the project.
    - Good and effective prompting will lead to a few bonus marks.
    - Ensure the code is modular, with clear separation of logic for different components (e.g., booking, pathfinding, layover management).

Project Description

'SkyNav AI' is an advanced flight navigation tracker that visualizes and optimizes flight routes between multiple destinations. It processes complex flight data, including origin and destination cities, dates, flight durations, airlines, and layover times. The system helps users make informed decisions by analyzing direct and connecting flights, optimizing routes based on user preferences, and managing layovers efficiently.



Core Features and Their Data Structures & Graphical Implementations:

1. **Flight Data Representation**:
   - **Feature Description**: Flight data from `Flights.txt` is parsed into a graph where vertices represent cities and edges represent available flights between them. Each edge stores flight details, including airline name, flight duration, and travel dates.
   - **Example**: If a user searches for flights from New York (JFK) to Los Angeles (LAX) on a specific date, the graph helps display both direct and connecting flights, showing data like flight duration and airline details.
   - **Graphics Implementation**: Using SFML, the graph is visualized with nodes representing cities and edges representing flight routes. Edge colors and widths

denote flight durations, with tooltips displaying detailed flight information. Above picture is a potential visualization of the generated graph.

2. **Flight Booking Feature**:
    - **Feature Description**: Users can book flights by selecting their origin, destination, and date. If direct flights aren't available, the system shows connecting flights with layover feasibility and available time.
    - **Example**: A user searching for flights from Boston to Paris without a direct flight option would see possible routes like Boston to London to Paris, ensuring layover times are sufficient.
    - **Graphics Implementation**: All the possible direct and connecting routes from the specified source and destination will be highlighted in the graph.
3. **Shortest and Cheapest Route Finder**:
    - **Feature Description**: Dijkstra's and A* algorithms are used to find the shortest or least expensive routes between cities. Priority queues (implemented using heaps) enable efficient node processing, reducing computation time for optimal route discovery.
    - **Example**: A query from San Francisco to Tokyo finds the most economical path, with layovers in cities such as Seattle or Honolulu, using Dijkstra's algorithm and priority queues.
    - **Graphics Implementation**: The optimal route is highlighted in real-time with animations showing node evaluations. The completed path is highlighted with a glowing effect for clarity.
4. **Custom Flight Paths and Preferences**:
    - **Feature Description**: Users can set preferences like preferred airlines or transit cities, allowing the system to filter flights and display relevant options.
    - **Example**: A traveler preferring Emirates flights with a layover in Dubai would see routes filtered to show only relevant options, focusing on Emirates flights through Dubai.
    - **Graphics Implementation**: Preferred cities are marked with some different icons, and recalculated paths based on user preferences are highlighted with unique visual effects.

5. **Layover Management with Queue**:
    - **Feature Description**: The system uses a queue to manage layover times at each airport. When users select a route with layovers, the system processes flight and layover times in a queue, ensuring smooth transitions between flights.
    - **Example**: A flight from Miami to London with a 14-hour layover in New York is processed by the queue, factoring in both flight and layover times. The system calculates the next available flight and layover duration accordingly.
    - **Graphics Implementation**: Layovers are visualized with dashed lines, and a displays information on layover times and next available flights, updating dynamically as users navigate through their journey.
6. **Advanced Route Generation using Linked List**:
    - **Feature Description**: For complex flight paths with multiple stops, the system employs a linked list to represent and manage each segment of the journey. This allows for dynamic and flexible route generation, with each node in the linked list representing a segment of the trip.
    - **Example**: A user planning a multi-leg journey, such as from San Francisco to Tokyo with layovers in several cities, can visualize each leg of the journey using a linked list structure, which dynamically adapts to user changes in preferences.
    - **Graphics Implementation**: The linked list structure is visualized with arrows connecting each stop, and users can interactively adjust the path by adding or removing legs of the journey.

Graphical Query and Subgraph Generation:

- **Feature Description**: The system can generate subgraphs that focus on user queries, such as specific airlines or transit routes. Cities without relevant flights are excluded, optimizing the visualization and making it easier for users to focus on the most pertinent routes.
- **Example**: A user preferring Qatar Airways would see a subgraph showing only cities served by Qatar Airways, highlighting the most relevant routes.

- **Graphics Implementation**: The subgraph is displayed with reduced opacity for excluded cities, while active routes remain bold and clear, emphasizing the routes most relevant to the user’s preferences.

Advanced Algorithm Adaptations and Visuals:

- **Dijkstra’s and A***: These algorithms are adapted with visual feedback, enabling users to watch the step-by-step decision process. Each node is evaluated dynamically, and the path is built in real-time.
- **Pathfinding Enhancements**: Bidirectional search techniques are applied to improve long-distance route efficiency, with visual feedback showing progressively highlighted paths.

Conclusion:
SkyNav AI’s integration of C++ data structures like graphs, queues, and linked lists, along with dynamic and visually engaging graphics, provides an interactive and data-rich flight planning experience. Its comprehensive approach to visualizing complex flight data, managing layovers, booking flights, and adapting routes to user preferences ensures an optimal and personalized travel experience.

Additional Instruction about provided files:

The **Flights.txt** file contains information about the flights. In particular, each entry in the file contains the following information (in the same order, separated by space)

- Origin
- Destination
- Date of travel
- Flying time
- Landing time
- Ticket price
- Name of airline

It is important to note that the flying and landing times are added in the flight data however the passengers are not given the option to choose the preferred time of fly (rather passengers are can only choose the preferred date of travel). The flying and landing times are included in the flight data for the correct identification of connecting flights and transit times. For example, if a PIA flight from Islamabad to Karachi is from 9:00 AM to 11:00 AM. A passenger cannot take a connecting flight from Karachi with flying time earlier than 11:00 AM. Likewise, a passenger has a transit time of 6 hours in Karachi, if the connecting flight from Karachi is scheduled to fly at 5:00 PM.

Moreover, the **HotelCharges_perday.txt** file contains information about the hotel price if stay is more than 12 hours (or passenger preferred to have a longer transit stay).

Generate a main graph by reading the data provided in Flights.txt and HotelCharges_perday.txt files. For example, the vertices in the graph will be cities and edges represent availability of flights between the cities. Moreover, edges store information about the flights between the two adjacent vertices, e.g., name of airline, ticket price, date of travel, flying time, landing time etc. The hotel price information can also be stored in the vertices of the respective cities.