# **Design Document for Flight Reservation System**

## **Problem Statement**

The goal is to create a Flight Reservation System using linked lists to manage flight bookings. Each flight is represented as a node in the linked list, with operations to add, remove, search, book, unbook flights, and display all flight statuses. The solution uses Python 3.7+ and adheres to constraints such as avoiding external libraries.

## **Data Structure**

### **Choice of Data Structure: Singly Linked List**

A singly linked list is chosen for its simplicity and efficiency in dynamically managing a sequence of flights. Each node in the list represents a flight, containing:

* flight\_id: A unique identifier for the flight.
* description: A string describing the flight.
* status: The current status of the flight (Available or Booked).
* next: A pointer to the next node in the list.

#### **Justification:**

1. **Dynamic Size**: Linked lists allow for efficient insertion and deletion without requiring resizing or memory reallocation.
2. **Sequential Operations**: The problem heavily relies on sequential traversal (e.g., searching, displaying statuses), which suits the linear nature of linked lists.
3. **Efficiency**: Adding or removing flights at the ends has a constant time complexity, ideal for frequent operations.

## **Operations and Time Complexity**

### **1. Add a Flight**

* **Functionality**: Adds a new flight node at the end of the linked list and assigns a unique flight ID.
* **Implementation**: Appends the node to the tail.
* **Time Complexity**: O(1)

### **2. Remove a Flight**

* **Functionality**: Removes a flight node identified by description or flight ID.
* **Implementation**: Traverses the list to find and unlink the node.
* **Time Complexity**: O(n) (traversal required)

### **3. Search a Flight**

* **Functionality**: Searches for flights matching a keyword in the description or flight ID.
* **Implementation**: Iterates through the list and collects matching nodes.
* **Time Complexity**: O(n)

### **4. Book a Flight**

* **Functionality**: Marks a flight as Booked by updating its status.
* **Implementation**: Traverses the list to find the target node and modifies its status.
* **Time Complexity**: O(n)

### **5. Unbook a Flight**

* **Functionality**: Marks a flight as Available by updating its status.
* **Implementation**: Similar to booking, but resets the status.
* **Time Complexity**: O(n)

### **6. Show All Flight Status**

* **Functionality**: Displays the flight\_id, description, and status of all flights.
* **Implementation**: Iterates through the list and appends details to the output.
* **Time Complexity**: O(n)

## **Alternative Design**

### **Alternative Data Structure: Doubly Linked List**

A doubly linked list could be used for additional flexibility, allowing bidirectional traversal.

#### **Advantages:**

1. Faster deletions when the node's position is known, as pointers to both previous and next nodes are available.
2. Simplifies certain operations, such as reversing the list or moving backward in search.

#### **Disadvantages:**

1. Increased memory overhead due to storing an additional pointer (prev) per node.
2. Slightly higher complexity in maintaining the list during insertions and deletions.

#### **Cost Implications:**

* Higher memory usage and implementation complexity make this alternative less suitable for the given problem's simplicity.

## **Error Handling**

1. **Invalid Input File**: If the input file does not exist, an appropriate error message is displayed.
2. **Flight Not Found**: Operations like booking, unbooking, or removal return a specific message if the target flight is not found.
3. **Empty List**: Operations gracefully handle cases where the list is empty.

## **Run-Time Analysis**

| **Operation** | **Time Complexity** |
| --- | --- |
| Add a Flight | O(1) |
| Remove a Flight | O(n) |
| Search a Flight | O(n) |
| Book/Unbook a Flight | O(n) |
| Show All Flight Status | O(n) |

## **Conclusion**

The chosen implementation using a singly linked list efficiently meets the requirements of the Flight Reservation System. It balances simplicity with performance for the given operations. While alternatives like a doubly linked list offer benefits, the added complexity is unnecessary for this problem's scope.