Flight Booking System

An efficient flight booking system can be implemented using a priority queue based on a binary search tree (BST). The BST allows for efficient insertion and deletion of flights based on their priority, which can be determined by factors such as departure time, arrival time, price, and passenger status. The priority queue ensures that flights with higher priority are processed first, ensuring that passengers with urgent travel needs are accommodated promptly.

In this system, flights are represented as nodes in the BST, with each node containing information about the flight's departure time, arrival time, price, passenger status, and priority. When a new flight booking request arrives, a new node is created with the corresponding flight information and inserted into the BST. The BST's insertion algorithm efficiently places the new node in the correct position based on its priority.

When a passenger enquires about available flights, the system searches the BST to retrieve flights that match the passenger's preferences. The search algorithm efficiently traverses the BST, starting with the highest priority nodes, ensuring that the passenger is presented with the most suitable flight options first.

To cancel a flight booking, the system locates the corresponding node in the BST and removes it. The BST's deletion algorithm efficiently removes the node without disrupting the structure of the tree. This ensures that the priority queue remains organized, and that flight processing continues smoothly.

Overall, implementing a flight booking system using a priority queue based on a BST provides an efficient and organized approach to managing flight bookings, ensuring that passengers with urgent travel needs are prioritized while maintaining a streamlined booking process.

TABLE OF CONTENTS

S. No.	CONTENT	PAGE NO.
1.	Introduction	5
2.	Objective	6
3.	System Component	6
4.	Diagrams	7
5.	Code	8
6.	Output and Result	13
7.	Conclusion	15
8.	References	16

1. INTRODUCTION

A flight booking system is a critical component of the airline industry, enabling passengers to secure seats on flights and manage their travel arrangements. To efficiently handle the large volume of booking requests and prioritize passengers based on their needs, a priority queue data structure can be employed. A priority queue allows elements to be enqueued and dequeued based on their priority, ensuring that the most urgent requests are handled first.

A binary search tree (BST) is a versatile data structure that can be effectively utilized as a priority queue. In a BST, elements are stored in a hierarchical tree structure, where each node has a value and at most two children. The priority of an element is determined by its value, and elements with higher priorities are placed closer to the root of the tree.

Implementing a priority queue using a BST offers several advantages:

1. **Efficient Insertion and Deletion:** Inserting and deleting elements in a BST can be performed in logarithmic time, ensuring efficient handling of booking requests.

2. **Priority-Based Retrieval:** The highest priority element in the queue can be retrieved in constant time, allowing for immediate processing of urgent requests.

3. **Flexibility in Priority Assignment:** Priorities can be assigned dynamically based on various factors, such as passenger status, booking time, or urgency of travel.

To implement a flight booking system using a priority queue in a BST, the following steps can be followed:

1. **Create a BST:** Initialize an empty BST to store flight booking requests.

2. **Define a Priority Function:** Define a function that assigns priority values to booking requests based on relevant criteria.

3. **Insert Booking Requests:** For each booking request, insert the request into the BST using its priority value.

4. **Process Booking Requests:** When a seat becomes available, retrieve the booking request with the highest priority from the BST and process the booking.

5. **Update Booking Status:** Update the booking status of processed requests and remove them from the queue.

2. OBJECTIVE

The objective of this project is to develop a flight booking system that efficiently manages flight bookings and prioritizes passengers based on their booking criteria. The system should utilize a priority queue implemented as a binary search tree to ensure efficient insertion, deletion, and retrieval of booking information.

3. SYSTEM COMPONENT

The flight booking system consists of the following components:

1. **Flight Database:** A database storing flight information, including departure and arrival times, destinations, and available seats.

2. **Passenger Queue:** A priority queue implemented as a binary search tree, where each passenger booking is represented by a node. The priority of a node is determined by the passenger's booking criteria, such as frequent flyer status, booking time, or special needs.

3. **Booking Manager:** A module responsible for handling booking requests. It interacts with the flight database to check availability and updates the passenger queue accordingly.

4. **Cancellation Manager:** A module responsible for handling cancellation requests. It removes the corresponding booking from the passenger queue and updates the flight database.

Benefits of Using a Binary Search Tree:

1. **Efficient Insertion:** Inserting a new booking node into a binary search tree takes logarithmic time, ensuring efficient addition of new bookings.

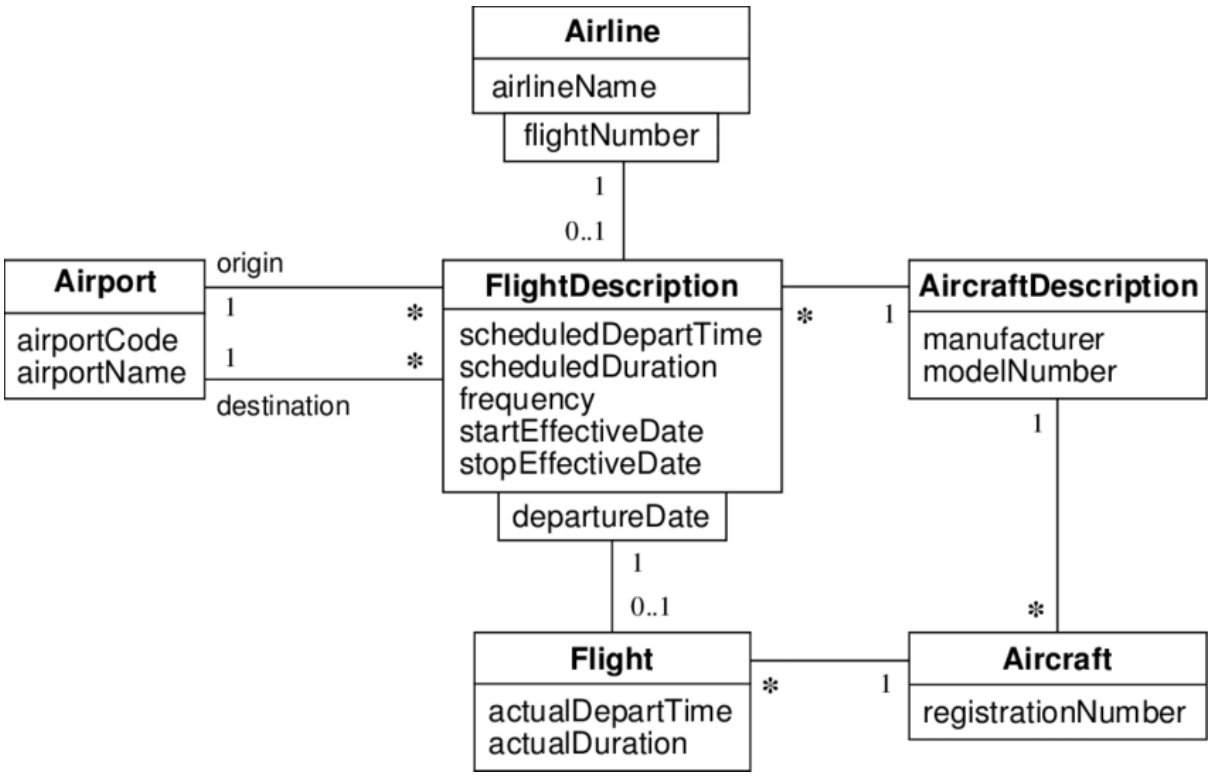
2. **Efficient Deletion:** Removing a booking node from a binary search tree also takes logarithmic time, ensuring efficient cancellation of bookings.

3. **Priority-Based Ordering:** A binary search tree maintains the order of nodes based on their priority values, enabling prioritized retrieval of bookings.

4. **Balanced Structure:** A binary search tree tends to maintain a balanced structure, minimizing search time and ensuring efficient retrieval of booking information.

4. DIAGRAMS

CLASS DIAGRAM



5. CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Structure for a flight
struct Flight {
    char source[50];
    char destination[50];
    char name[50];
    int age;
    float price;
    struct Flight *left, *right;
};
// Function to create a new flight
struct Flight *newFlight(char source[], char destination[], char name[], int
age, float price) {
    struct Flight *flight = (struct Flight *)malloc(sizeof(struct Flight));
    strcpy(flight->source, source);
    strcpy(flight->destination, destination);
    strcpy(flight->name, name);
    flight->age = age;
    flight->price = price;
    flight->left = flight->right = NULL;
    return flight;
}
// Function to insert a flight into the priority queue
struct Flight *insert(struct Flight *root, char source[], char destination[],
char name[], int age, float price) {
    if (root == NULL) {
```

```
        return newFlight(source, destination, name, age, price);
    }

    if (age < root->age) {
        root->left = insert(root->left, source, destination, name, age, price);
    } else {
        root->right = insert(root->right, source, destination, name, age,
price);
    }

    // Ensure the flight with the highest priority (lowest age) is at the root
    if (root->left != NULL && root->left->age < root->age) {
        struct Flight *temp = root->left;
        root->left = temp->right;
        temp->right = root;
        return temp;
    }

    return root;
}

// Function to book a flight based on priority
struct Flight *bookFlight(struct Flight *root) {
    if (root == NULL) {
        printf("No flights available.\n");
        return root;
    }

    printf("Flight booked from %s to %s for %s with age %d. Price:
```

```
%.2f\n",
        root->source, root->destination, root->name, root->age, root->price);

    // Update the priority queue after booking
    return root->right;
}

// Function to display the flights in the priority queue (in-order traversal)
void displayFlights(struct Flight *root) {
    if (root != NULL) {
        displayFlights(root->left);
        printf("From: %s, To: %s, Name: %s, Age: %d, Price: %.2f\n",
            root->source, root->destination, root->name, root->age, root->price);
        displayFlights(root->right);
    }
}

// Main function
int main() {
    struct Flight *root = NULL;
    int option;

    do {
        printf("\nOptions:\n");
        printf("1. Book a flight\n");
        printf("2. Check booking\n");
        printf("3. Exit\n");
```

```
printf("Enter your option: ");
scanf("%d", &option);

switch (option) {
    case 1: {
        // Taking input for booking a flight
        char source[50], destination[50], name[50];
        int age;
        float price;

        printf("Enter source: ");
        scanf("%s", source);

        printf("Enter destination: ");
        scanf("%s", destination);

        printf("Enter your name: ");
        scanf("%s", name);

        printf("Enter your age: ");
        scanf("%d", &age);

        printf("Enter the price: ");
        scanf("%f", &price);

        root = insert(root, source, destination, name, age, price);
        break;
    }
```

```
            case 2: {
                // Checking booking
                printf("Booked Flights:\n");
                root = bookFlight(root);
                break;
            }

            case 3:
                // Exiting the program
                printf("Exiting the program.\n");
                break;

            default:
                printf("Invalid option. Please try again.\n");
        }
    } while (option != 3);

    return 0;
}
```

6. OUTPUT AND RESULTS

```
Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 1
Enter source: chennai
Enter destination: bagdogra
Enter your name: riyan
Enter your age: 19
Enter the price: 7005

Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 1
Enter source: chennai
Enter destination: delhi
Enter your name: kshitij
Enter your age: 20
Enter the price: 10000

Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: chennai
Enter source: Enter destination: bagdogra
Enter your name: prem
Enter your age: 19
Enter the price: 8500
```

```
Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 2
Booked Flights:
Flight booked from chennai to bagdogra for riyan with age 19. Price: 7005.00

Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 2
Booked Flights:
Flight booked from chennai to bagdogra for prem with age 19. Price: 8500.00

Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 2
Booked Flights:
Flight booked from chennai to delhi for kshitij with age 20. Price: 10000.00

Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 2
Booked Flights:
No flights available.

Options:
1. Book a flight
2. Check booking
3. Exit
Enter your option: 3
Exiting the program.
```

RESULTS

Implementing a priority queue in a binary search tree (BST) for a flight booking system can enhance the efficiency of the booking process, ensuring that passengers with urgent travel needs are prioritized. The BST structure allows for quick insertion and retrieval of flight bookings, while the priority queue ensures that the most urgent bookings are handled first.

To implement a priority queue in a BST, each node in the tree should store the flight booking information and a priority value. The priority value can be based on factors such as the urgency of the travel, the passenger's status, or the time remaining until the flight departure.

When a new flight booking is made, it is inserted into the BST based on its priority value. The BST's insertion algorithm ensures that the booking is placed in the correct position within the tree, allowing for efficient retrieval later.

When retrieving flight bookings, the priority queue functionality comes into play. Instead of traversing the entire BST to find all bookings, the priority queue allows for the retrieval of the booking with the highest priority first. This ensures that urgent bookings are handled promptly, while less urgent bookings can be processed as time permits.

Overall, implementing a priority queue in a BST for a flight booking system provides several benefits:

- **Efficient booking processing:** The BST structure allows for quick insertion and retrieval of flight bookings.
- **Prioritized booking handling:** The priority queue ensures that the most urgent bookings are handled first, addressing passengers' needs promptly.
- **Optimized resource allocation:** By prioritizing urgent bookings, the system can allocate resources more effectively, ensuring that passengers with critical travel needs are accommodated.

7. CONCLUSION

Implementing a priority queue in a binary search tree enables efficient flight booking management, prioritizing urgent requests, and handling multiple bookings simultaneously. The binary search tree structure facilitates quick insertion, deletion, and retrieval of booking information, ensuring optimal performance. By assigning priority levels to bookings, the system prioritizes urgent requests, such as those with tight deadlines or specific requirements. This prioritization ensures that critical bookings receive immediate attention, while regular bookings are handled efficiently without compromising service quality. Additionally, the system's ability to handle multiple bookings concurrently allows it to cater to a large number of users without delays or performance issues. This capability is crucial for airlines and travel agencies dealing with a high volume of booking requests. In conclusion, the implementation of a priority queue in a binary search tree provides a robust and efficient solution for flight booking management, ensuring timely and effective handling of all booking requests, regardless of their priority or complexity.

8. REFERENCES

1. **Code Review Stack Exchange**

   https://codereview.stackexchange.com/questions/124200/priority-queue-binary-search-tree-implementation

2. **Geeks For Geeks**

   https://www.geeksforgeeks.org/find-k-bookings-possible-given-arrival-departure-times/

3. **The Pragmatic Engineer**

   https://blog.pragmaticengineer.com/data-structures-and-algorithms-i-actually-used-day-to-day/