1109. Corporate Flight Bookings

Medium Array Prefix Sum Leetcode Link

Problem Description

In this problem, we are managing a system that tracks flight bookings. We are given `n` flights, which are identified by numbers ranging from `1` to `n`. Additionally, we are provided with an array named `bookings`. Each entry in this array is another array that contains three elements, `[first_i, last_i, seats_i]`. This entry indicates a booking for multiple flights - from flight number `first_i` to flight number `last_i` (inclusive) - and each flight in this range has `seats_i` seats reserved.

The task is to compute and return an array, `answer`, which has the same length as the number of flights (`n`). Each element in `answer` should represent the total number of seats booked for the corresponding flight. In other words, `answer[i]` should tell us how many seats have been reserved for the ith flight.

Intuition

The key to solving this problem is recognizing that the range update operation (booking seats for a range of flights) can be efficiently tracked using the concept of difference arrays. A difference array can allow us to apply the effects of a range update in constant time and later reconstruct the original array with the complete effects applied.

The intuition behind using a difference array is as follows:

- When seats are booked from flight `first_i` to `last_i`, we do not necessarily need to add `seats_i` to each entry between `first_i` and `last_i` in the answer array. Instead, we could record the `seats_i` addition at the start index `first_i - 1` (since arrays are zero-indexed) and a negation of `seats_i` at the index `last_i`, indicating the end of the booking range.
- What this achieves is a sort of "bookend" model, where we place a marker for where seats start being added and where they stop. This is somewhat akin to noting down the start and end of a chalk mark without having to color in the whole segment.

Once we have made these updates to our answer array, which now serves as a difference array, we need to accumulate the effects to find out the total number of seats booked for each flight. Accumulating from left to right will add `seats_i` from the start index up until the end index, effectively applying the range update.

- For example, if we add 10 seats at index 0 (flight 1) and subtract 10 seats at index 5 (after flight 5), accumulating the values will result in each of the first 5 elements showing an increment of 10, and the increment will no longer apply from position 6 onward.
- The key operation to accomplish the accumulation efficiently in our Python code is to utilize the `accumulate` function from the `itertools` module.

This approach takes us from a brute-force method that might involve multiple iterations per booking (which would be inefficient and potentially time-consuming for a large number of bookings or flights) to a much more efficient algorithm with a time complexity that is linear with respect to the number of flights and bookings.

Solution Approach

Let's break down the implementation provided in the reference solution and see how it applies the intuition that we discussed above.

1. Initialize an array `ans` of zeros with the same length as the number of flights (`n`). This array will serve as our difference array where we'll apply the updates from the bookings and later accumulate these updates to get the final answer.

```
ans = [0] * n
```

2. Iterate over each booking in the bookings array. For each booking:

   - Increase the element at the index (`first - 1`) in the `ans` by `seats`. This corresponds to the start of the booking range where the seats begin to be reserved.
   - If the last flight in the booking is not the very last flight available, then decrease the element at the index `last` in the `ans` by `seats`. This will indicate the end of the booking range where the reserved seats are no longer available.

```
for first, last, seats in bookings:
    ans[first - 1] += seats
    if last < n:
        ans[last] -= seats
```

3. Use the `accumulate` function to accumulate the elements of `ans`. This Python function takes a list and returns a list where each element is the cumulative total from the first element to the current one. This effectively reconstructs the resulting array where each element represents the total number of seats reserved for the corresponding flight after applying all bookings. Since the `ans` array at this stage is a difference array, the accumulation will give us the true number of seats reserved for each flight.

```
return list(accumulate(ans))
```

In terms of data structures, the problem makes essential use of an array data structure. The algorithmic pattern used here is the "difference array" pattern, which enables us to efficiently apply range updates (in O(1) time) and then reconstruct the original array with a single pass to accumulate these changes (in O(n) time).

The overall algorithm can be summarized in the following steps:

- Initialize a difference array with zeros
- Apply the range update operations to the difference array
- Accumulate the difference array to reconstruct the final answer

This approach is much more efficient than directly applying increments to a range of elements within the array for each booking, especially when dealing with a large number of flights and bookings.

Example Walkthrough

Let's use a small example to illustrate the solution approach.

Suppose we have `n = 5` flights, and we receive the following bookings: `[[1, 2, 10], [2, 3, 20], [3, 5, 25]]`. This means:

- 10 seats are booked for flights 1 and 2.
- 20 seats are booked for flights 2 and 3.
- 25 seats are booked for flights 3 to 5 (inclusive).

We start with an array `ans` initialized with zeros, with the same length as the number of flights (`n`), thus `ans = [0, 0, 0, 0, 0]`.

Now, we iterate over each booking to update the `ans` array:

1. For the first booking `[1, 2, 10]`, we add 10 seats at index 0 and subtract 10 seats at index 2.

   - Before: `ans = [0, 0, 0, 0, 0]`
   - After: `ans = [10, 0, -10, 0, 0]`

2. For the second booking `[2, 3, 20]`, we add 20 seats at index 1 and subtract 20 seats at index 3.

   - Before: `ans = [10, 0, -10, 0, 0]`
   - After: `ans = [10, 20, -10, -20, 0]`

3. For the third booking `[3, 5, 25]`, we add 25 seats at index 2 and there is no need to subtract at the end since the booking range includes the last flight.

   - Before: `ans = [10, 20, -10, -20, 0]`
   - After: `ans = [10, 20, 15, -20, 0]`

Finally, we use the `accumulate` function to build the answer from the `ans` difference array:

- Accumulate: `[10, 30, 45, 25, 25]`
- This means the total number of seats booked for each flight is 10, 30, 45, 25, and 25, respectively.

The final result of our algorithm gives us the array `[10, 30, 45, 25, 25]`, which indicates the total number of seats reserved for each flight from 1 to 5 after all the bookings have been applied.

This example demonstrates the effectiveness of the difference array pattern in managing range updates efficiently. By applying incremental updates at the start and a decremental update at the end of each booking range, we avoid multiple iterations over the range for each booking. Once all the incremental and decremental updates are applied, accumulating the changes provides the total number of booked seats for each flight.

Python Solution

```
from typing import List
from itertools import accumulate

class Solution:
    def corpFlightBookings(self, bookings: List[List[int]], n: int) -> List[int]:
        # Initialize a result list to hold the number of seats booked for each flight
        flight_seats = [0] * n

        # Iterate through each booking to update flight_seats
        for start, end, seats in bookings:
            # Increase the seats count at the start index
            flight_seats[start - 1] += seats

            # Decrease the seats count at the index after the end to nullify the increase
            # which will happen due to prefix sum (accumulate) later
            if end < n:
                flight_seats[end] -= seats

        # Calculate the prefix sum which gives the total number of seats booked
        # for each flight from the beginning
        return list(accumulate(flight_seats))
```

Java Solution

```
class Solution {

    /*
    This method calculates the number of seats booked on each flight.

    Parameters:
    bookings - an array of bookings, where bookings[i] = [first_i, last_i, seats_i]
    n - the number of flights

    Returns:
    an array containing the total number of seats booked for each flight.
    */
    public int[] corpFlightBookings(int[][] bookings, int n) {
        // Initialize an array to hold the answer, with n representing the number of flights.
        int[] answer = new int[n];

        // Iterate over each booking.
        for (int[] booking : bookings) {
            // Extract the start and end flight numbers and the number of seats booked.
            int startFlight = booking[0];
            int endFlight = booking[1];
            int seats = booking[2];

            // Increment the seats for the start flight by the number of seats booked.
            answer[startFlight - 1] += seats;

            // If the end flight is less than the number of flights,
            // decrement the seats for the flight immediately after the end flight.
            if (endFlight < n) {
                answer[endFlight] -= seats;
            }
        }

        // Iterate over the answer array, starting from the second element,
        // and update each position with the cumulative sum of seats booked so far.
        for (int i = 1; i < n; ++i) {
            answer[i] += answer[i - 1];
        }

        // Return the populated answer array.
        return answer;
    }
}
```

C++ Solution

```
class Solution {
public:
    // Function to calculate how many seats are booked on each flight
    vector<int> corpFlightBookings(vector<vector<int>>& bookings, int n) {
        vector<int> seatsBooked(n); // Array to store the final number of seats booked for each flight

        // Loop over all the booking records
        for (auto& booking : bookings) {
            int startFlight = booking[0]; // Start flight number of the booking
            int endFlight = booking[1];   // End flight number of the booking
            int seats = booking[2];       // Number of seats to be booked in this range

            // Increment the seats booked for the start flight by the number of seats booked
            seatsBooked[startFlight - 1] += seats;

            // If the booking ends before the last flight, decrement the seats booked
            // for the first flight after the end flight. This will be used later for
            // calculating cumulative sum.
            if (endFlight < n) {
                seatsBooked[endFlight] -= seats;
            }
        }

        // Calculate the cumulative sum to get the actual number of seats booked for each flight
        for (int i = 1; i < n; ++i) {
            seatsBooked[i] += seatsBooked[i - 1];
        }

        // Return the final results
        return seatsBooked;
    }
};
```

Typescript Solution

```
/**
 * This function takes a list of bookings and the total number of flights (n),
 * and returns an array where each element represents the total number of seats
 * booked on that flight.
 *
 * @param {number[][]} bookings - A list where each element is a booking in the
 * form [first, last, seats] indicating a booking from flight 'first' to flight
 * 'last' (inclusive) with 'seats' being the number of seats booked.
 * @param {number} n - The total number of flights.
 * @return {number[]} - An array representing the total number of seats booked
 * on each flight.
 */
const corpFlightBookings = (bookings: number[][], n: number): number[] => {
    // Initialize answer array with n elements set to 0.
    const answer: number[] = new Array(n).fill(0);

    // Loop through each booking.
    for (const [first, last, seats] of bookings) {
        // Increment the seats for the first flight in the current booking range.
        answer[first - 1] += seats;
        // Decrement the seats for the flight just after the last flight in
        // the current booking range, if it is within bounds.
        if (last < n) {
            answer[last] -= seats;
        }
    }

    // Aggregate the booked seats information.
    // Each flight's bookings include its own and also the cumulative bookings
    // from the previous flight. This is because the seats booked in the flight ranges
    // overlap and the initial range difference accounted for it.
    for (let i = 1; i < n; ++i) {
        answer[i] += answer[i - 1];
    }

    // Return the final aggregated information about flight bookings.
    return answer;
};
```

Time and Space Complexity

The time complexity of the provided code is composed of two parts: the iteration through the `bookings` list and the accumulation of the values in the `ans` array.

The iteration through the `bookings` list happens once for each booking. For each booking, the code performs constant-time operations: an increment at the start position and a decrement at the end position. Therefore, if there are `k` bookings, this part has a time complexity of $O(k)$.

The `accumulate` function is used to compute the prefix sums of the `ans` array, which has `n` elements. This operation is linear in the number of flights, so it has a time complexity of $O(n)$.

Combining these two parts, the overall time complexity of the code is $O(k + n)$, since we have to consider both the number of bookings and the number of flights.

The space complexity of the code is primarily determined by the storage used for the `ans` array, which has `n` elements. The `accumulate` function returns an iterator in Python, so it does not add additional space complexity beyond what is used for the `ans` array. Therefore, the space complexity of the provided code is $O(n)$.