1052. Grumpy Bookstore Owner

Medium Array Sliding Window

Problem Description

In this problem, we have a bookstore owner who has their store open for `n` minutes. During each of those minutes, customers enter the store. The number of customers entering the store at the beginning of the `i`th minute is given by `customers[i]`. After each minute, all customers for that particular minute leave.

The bookstore owner's mood fluctuates, and this affects customer satisfaction. The mood is captured by an array `grumpy`. If `grumpy[i]` is `1`, the owner is grumpy in the `i`th minute; if `0`, they are not grumpy. When the owner is grumpy, customers leaving in that minute are not satisfied.

However, the owner knows a secret technique that allows them to be not grumpy for `minutes` consecutive minutes, but they can use this secret technique only once in the day.

The task is to determine the maximum number of customers that can be satisfied throughout the day by smartly applying the secret technique to a certain time window during the day.

Intuition

The solution aims to identify the best time segment where using the secret technique will result in the maximum number of satisfied customers. We can always satisfy customers when the owner is not grumpy, so these customers are not our focus. Instead, we want to maximize the number of satisfied customers during the owner's grumpy periods.

To achieve this, we observe that the key is to apply the secret technique during the consecutively grumpy moments when the number of customers is highest. Here's how the solution approaches the problem:

1. We track the number of customers that could be satisfied if the owner were not grumpy at all by adding up the number of customers during grumpy minutes. This gives us the total potential satisfaction if the owner were not grumpy.
2. Then, we create a sliding window with the width equal to `minutes` – the duration for which the secret technique can be applied. As we slide this window across the grumpy minutes, we calculate the number of dissatisfied customers in this window.
3. We keep track of the maximum number of these potentially dissatisfied customers that can be turned into satisfied customers by applying the secret technique in the current sliding window.
4. We use a variable `t` to calculate the temporary sum of customers that are currently not satisfied due to the owner being grumpy within the window. Whenever we move the window forward by a minute, we deduct the customers at the start of the window (if the owner was grumpy at that minute) and add the customers at the new end of the window (if the owner is grumpy at the new end).
5. We keep adjusting the best maximum number of satisfied customers found so far (`ans`) each time we slide the window one minute forward by subtracting the number of unsatisfied customers in the window from the total potential satisfaction.

In the end, we return the maximum number of customers that can be satisfied which will include the number of customers that were satisfied when the owner was not grumpy and the additional customers that were satisfied when using the secret technique during the grumpy periods.

Solution Approach

The solution approach utilizes a sliding window to find the optimal `minutes` to apply the secret technique so as to maximize the satisfaction of customers. Here's how the implementation details map to this strategy:

1. We first calculate `s`, the sum of customers that are not satisfied initially due to the owner being grumpy. This is done by taking the dot product of the `customers` and `grumpy` arrays. Only the minutes when the owner is grumpy (`grumpy[i] == 1`) contribute to `s`.
2. `cs` registers the total number of customers that visit the store throughout the day, which is the sum of the entire `customers` array without taking the owner's mood into account.
3. Two variables `t` and `ans` are initiated to zero. `t` will capture the total number of dissatisfied customers within the current window, and `ans` will store the maximum number of customers that can be satisfied during the best window of `minutes`.
4. We then iterate through each minute `i`, simultaneously calculating the sliding window sum `t` for grumpy minutes. We do this by incrementing `t` by `customers[i]` only if `grumpy[i] == 1`.
5. In each iteration, we check if the sliding window has extended beyond the prescribed `minutes`. This is done using a condition `(j := i - minutes) >= 0`. If true, it means we have a full window and we are moving it one step forward.
6. We calculate the potential maximum number of satisfied customers `ans` by subtracting from `cs` the total dissatisfaction without the current window (`s - t`). If it exceeds our previous best (`ans`), we update `ans`.
7. After updating `ans`, we adjust `t` by subtracting the customers at the starting point of the previous window frame, but only if that minute was a grumpy one (`customers[j] * grumpy[j]`).
8. Once the loop is done, all possible windows of size `minutes` have been checked. The `ans` variable holds the maximum number of customers that can be satisfied by applying the secret technique during the best possible window. This value is returned.

Through the use of a simple loop and the sliding window technique, the implementation finds the optimal moment to use the secret technique without the need for complex data structures or algorithms. This is an efficient and effective solution for the given problem that runs in linear time, based on the number of minutes the store is open (`O(n)`).

Example Walkthrough

Let us consider the bookstore is open for `n = 7` minutes, and the number of customers entering at each minute is given by `customers = [1, 10, 10, 1, 1, 10, 10]`. The owner's mood over the 7 minutes is represented by `grumpy = [0, 1, 1, 1, 0, 1, 0]`, and the owner can be not grumpy for `minutes = 3` consecutive minutes using the secret technique.

1. We first calculate the total potential satisfaction without using the secret technique, which is the sum of customers that visit when the owner is not grumpy. From our example, this is `1 (1st minute) + 1 (5th minute) + 10 (7th minute) = 12`.
2. Next, using a sliding window of size `minutes = 3`, we want to find a period where applying the secret technique can maximize customer satisfaction. We start with the window at the beginning:

- For the first window (`minutes 1-3`), the owner is in a good mood in the 1st minute, but grumpy in the 2nd and 3rd minute, where we have 10 + 10 customers, but we skip the 1st minute since the owner is not grumpy.

3. We slide the window to the right by one minute (`minutes 2-4`), the owner is grumpy in all these minutes, resulting in 10 + 10 + 1 customers.
4. Continuing to slide our window to the right (`minutes 3-5`, `minutes 4-6`), we find the maximum number of grumpy customers that would be satisfied by the secret technique. The window `minutes 4-6` is where the owner is constantly grumpy with customers `[10, 1, 10]`.
5. As we slide, we update the temporary sum `t` as we go, adding new customers who entered during a grumpy minute and subtracting those at the starting minute if it was a grumpy period.
6. After sliding through all possible windows, the window `minutes 4-6` has the highest number of grumpy customers (`1 + 10 + 10 = 21`) that could be turned into satisfied customers.
7. The result is the sum of the initial satisfaction (12) and the additional satisfaction during the owner's grumpy period (21) when using the secret technique, which gives us the maximum customer satisfaction with the technique applied being 33.
8. Therefore, by applying the secret technique from the 4th to the 6th minute, the owner can maximize the number of satisfied customers for that day, resulting in a total satisfaction of 33 customers.

Solution Implementation

Python

```python
class Solution:
    def maxSatisfied(self, customers: List[int], grumpy: List[int], minutes: int) -> int:
        # Calculate the initial unsatisfied customer count (when the owner is grumpy)
        unsatisfied = sum(customer * grumpiness for customer, grumpiness in zip(customers, grumpy))

        # Calculate the total number of customers
        total_customers = sum(customers)

        # Tracking variable for the current window of grumpy period that can be improved
        temp_improved = max_improved = 0

        # Enumerate through each minute
        for minute_index, (customer, grumpiness) in enumerate(zip(customers, grumpy), 1):
            # Increase the count of potentially satisfied customers when the owner is grumpy
            temp_improved += customer * grumpiness

            # Slide the window of minutes for the "X" minute period
            if (start_index := minute_index - minutes) >= 0:
                # Update the max potential customers satisfied for any "X" minute window
                max_improved = max(max_improved, total_customers - (unsatisfied - temp_improved))

                # Remove the front of the window (the oldest minute moving out of the window)
                temp_improved -= customers[start_index] * grumpy[start_index]

        # The answer is the maximum potential customers satisfied in any "X" minute window
        return max_improved
```

Java

```java
class Solution {
    public int maxSatisfied(int[] customers, int[] grumpy, int minutes) {
        int totalGrumpyCustomers = 0; // This will hold the total number of customers affected by grumpiness
        int totalCustomersSatisfied = 0; // Sum of all customers that are satisfied
        int numCustomers = customers.length; // Total number of customers

        // Calculate the initial total number of customers that are satisfied and the total affected by grumpiness
        for (int i = 0; i < numCustomers; ++i) {
            if (grumpy[i] == 1) {
                // If the owner is grumpy during customer i's visit, add to totalGrumpyCustomers
                totalGrumpyCustomers += customers[i];
            }
            // Add to the total number of satisfied customers regardless of grumpiness
            totalCustomersSatisfied += customers[i];
        }

        int temporaryGrumpyCustomers = 0; // Temporary variable to keep track of grumpy customers during the 'minutes' window
        int maxGrumpyCustomersConverted = 0; // Maximum number of grumpy customers that can be converted to satisfied customers
        for (int i = 0; i < numCustomers; ++i) {
            // If the owner is grumpy, temporarily add this customer to the count
            if (grumpy[i] == 1) {
                temporaryGrumpyCustomers += customers[i];
            }
            // If we are beyond the 'minutes' window, subtract the customers at the start of the window
            if (i >= minutes) {
                if (grumpy[i - minutes] == 1) {
                    temporaryGrumpyCustomers -= customers[i - minutes];
                }
            }

            // Store the higher value between the current and previous maximum number of convertible grumpy customers
            maxGrumpyCustomersConverted = Math.max(maxGrumpyCustomersConverted, temporaryGrumpyCustomers);
        }

        // Calculate final answer: Subtract the initial grumpy customers from the total customers satisfied and add the convertil
        return totalCustomersSatisfied - totalGrumpyCustomers + maxGrumpyCustomersConverted;
    }
}
```

C++

```cpp
class Solution {
public:
    int maxSatisfied(vector<int>& customers, vector<int>& grumpy, int minutes) {
        int totalUnsatisfied = 0; // To keep track of the total unsatisfied customers if the owner is grumpy
        int totalCustomers = 0; // To keep track of the total number of customers served satisfactorily whether the owner is grum
        int windowUnsatisfied = 0; // To keep track of unsatisfied customers in the current time window of size 'minutes'
        int maxIncrease = 0; // Variable to store the maximum increase in satisfied customers achievable by using the technique i

        int n = customers.size(); // Total number of customers, assuming customers and grumpy are of same size
        // Calculate the initial state, considering the owner is grumpy all the time
        for (int i = 0; i < n; ++i) {
            totalUnsatisfied += customers[i] * grumpy[i]; // Only add customers that are unsatisfied because the owner is grumpy
            totalCustomers += customers[i]; // Add all customers
        }

        // Use a sliding window to determine when to utilize the owner's technique
        for (int i = 0; i < n; ++i) {
            windowUnsatisfied += customers[i] * grumpy[i]; // Add to window unsatisfied customers
            int endTime = i - minutes; // Calculate end time of window (i - minutes + 1) - 1 for 0-based indexing

            // If we've filled the first window, start calculating and updating maxIncrease
            if (endTime >= 0) {
                // Calculate the maximum satisfied customers by using the technique
                maxIncrease = max(maxIncrease, totalCustomers - (totalUnsatisfied - windowUnsatisfied));
                // Shrink the window from the left by removing the first element's impact if it was unsatisfied
                windowUnsatisfied -= customers[endTime] * grumpy[endTime];
            }
        }
        // After the loop, we must do one last check as the sliding window does not check the last window position
        maxIncrease = max(maxIncrease, totalCustomers - (totalUnsatisfied - windowUnsatisfied));

        // The answer is the maximum customers possible by using the technique in the best time window
        return maxIncrease;
    }
};
```

TypeScript

```typescript
let totalUnsatisfied: number = 0; // Track total unsatisfied customers if the owner is grumpy
let totalCustomers: number = 0; // Track total number of customers served satisfactorily
let windowUnsatisfied: number = 0; // Track unsatisfied customers in the current 'minutes' window
let maxIncrease: number = 0; // Maximum increase in satisfied customers possible by not being grumpy for 'minutes' duration

function maxSatisfied(customers: number[], grumpy: number[], minutes: number): number {
    totalUnsatisfied = 0;
    totalCustomers = 0;
    windowUnsatisfied = 0;
    maxIncrease = 0;

    const n: number = customers.length;
    // Initial calculations without utilizing the special technique
    for (let i = 0; i < n; i++) {
        totalUnsatisfied += customers[i] * grumpy[i];
        totalCustomers += customers[i];
    }

    // Using a sliding window to find the best time to apply the technique
    for (let i = 0; i < n; i++) {
        windowUnsatisfied += customers[i] * grumpy[i];
        let endTime: number = i - minutes;

        if (endTime >= 0) {
            maxIncrease = Math.max(maxIncrease, totalCustomers - (totalUnsatisfied - windowUnsatisfied));
            windowUnsatisfied -= customers[endTime] * grumpy[endTime];
        }
    }
    // Handle the last possible window to make sure we don't miss the chance of optimizing at the end
    maxIncrease = Math.max(maxIncrease, totalCustomers - (totalUnsatisfied - windowUnsatisfied));

    // The result is the best scenario where the technique is applied in the optimal window
    return maxIncrease;
}
```


```python
class Solution:
    def maxSatisfied(self, customers: List[int], grumpy: List[int], minutes: int) -> int:
        # Calculate the initial unsatisfied customer count (when the owner is grumpy)
        unsatisfied = sum(customer * grumpiness for customer, grumpiness in zip(customers, grumpy))

        # Calculate the total number of customers
        total_customers = sum(customers)

        # Tracking variable for the current window of grumpy period that can be improved
        temp_improved = max_improved = 0

        # Enumerate through each minute
        for minute_index, (customer, grumpiness) in enumerate(zip(customers, grumpy), 1):
            # Increase the count of potentially satisfied customers when the owner is grumpy
            temp_improved += customer * grumpiness

            # Slide the window of minutes for the "X" minute period
            if (start_index := minute_index - minutes) >= 0:
                # Update the max potential customers satisfied for any "X" minute window
                max_improved = max(max_improved, total_customers - (unsatisfied - temp_improved))

                # Remove the front of the window (the oldest minute moving out of the window)
                temp_improved -= customers[start_index] * grumpy[start_index]

        # The answer is the maximum potential customers satisfied in any "X" minute window
        return max_improved
```


Time and Space Complexity

The time complexity of the provided code is `O(N)`, where `N` is the length of the `customers` or `grumpy` lists. This is because the code consists of a single loop that iterates over the entire length of the input lists exactly once. Within the loop, there are constant-time operations being performed, such as arithmetic operations, comparisons, and an assignment inside an if-statement that triggers based on the loop index.

The space complexity of the code is `O(1)`, meaning it requires constant additional space. The extra space used in the algorithm is for variables `s`, `cs`, `t`, `ans`, and `j` (the latter found in the conditional if-statement). These variables do not depend on the size of the input, hence the constant space complexity. No additional data structures or variable-sized containers are used that would require space proportional to the input size.