1358. Number of Substrings Containing All Three Characters

Medium | Hash Table | String | Sliding Window | Leetcode Link

Problem Description

The given problem requires us to find the number of substrings in a string `s` that contains at least one occurrence of the characters 'a', 'b', and 'c'. The string `s` consists only of these three characters. A substring is any sequence of consecutive characters from the string. The goal is to count all such possible substrings where each of the three characters appears at least once.

Intuition

To solve this problem, we use a sliding window approach to track the latest positions of 'a', 'b', and 'c' while iterating through the string. The key intuition here is to understand that once we have found a substring containing all three characters, extending this substring to the right (by adding more characters in sequence) will also form valid substrings containing all three characters.

Here's the step-by-step intuition:

1. Initialize a dictionary to store the latest index of 'a', 'b', and 'c'. By default, they are set to -1, indicating that they haven't been found yet.
2. Iterate through the string character by character, updating the dictionary with the new index of each character encountered.
3. At each step, determine the smallest of the three indices because the smallest index indicates the rightmost position up to which we have seen all three characters together.
4. For the current index `i`, the count of valid substrings ending at `i` will be the minimum index among the latest indices of 'a', 'b', and 'c' plus one. This is because any substring starting from index 0 to the minimum index will have all three characters up to the current position `i`.
5. Sum up all these counts to get the total number of substrings containing all three characters.

This simple yet elegant solution effectively counts all the necessary substrings by considering each valid end position and how many substrings it can generate based on the earlier occurrences of 'a', 'b', and 'c'.

Solution Approach

The solution approach uses the concept of pointers and a hash map (in Python, a dictionary) to efficiently keep track of the latest occurrence of each character 'a', 'b', and 'c'.

```
d = {"a": -1, "b": -1, "c": -1}  # Dictionary to store the latest index for 'a', 'b', and 'c'
ans = 0  # This will hold the total count of substrings
```

The dictionary `d` serves as our hash map, holding the most recent indices of each character. Initially, all characters are set to `-1`, indicating they have not been encountered yet.

Then, we start iterating over each character in the string:

```
for i, c in enumerate(s):
    d[c] = i  # Update the latest index for the character `c`
    ans += min(d["a"], d["b"], d["c"]) + 1  # Count substrings ending at the current index `i`
```

As we iterate, for each character (`c`) at index `i`, we update its latest index in the dictionary. The `min(d["a"], d["b"], d["c"])` finds the smallest index of the three, which, as previously mentioned, is the furthest right we can go while still having all three characters. By adding `1` to this minimum value, we get the number of substrings ending at index `i` that has all three characters.

Finally, the sum of all such substrings is returned as the final answer:

```
return ans
```

Algorithm

1. Initialize a dictionary with keys 'a', 'b', 'c', and values `-1`.
2. Start iterating over each character in the string using a `for` loop.
3. Each time we find a character, update its latest index in the dictionary.
4. Calculate the number of valid substrings that end at the current index (minimum index of 'a', 'b', 'c' + 1).
5. Accumulate this count to a running total.
6. After the loop finishes, return the total count as the answer.

This algorithm is efficient because it processes each character exactly once and uses constant space for the dictionary, resulting in a time complexity of O(n) and a space complexity of O(1), where n is the length of the string.

Example Walkthrough

Let's use a string `s = "abcabc"` to illustrate the solution approach.

1. We start by initializing the dictionary `d` with `{"a": -1, "b": -1, "c": -1}`.
2. Set our running total of valid substrings `ans` to `0`.

Now we iterate through the string while keeping track of the latest index of each character in `d`.

- For `i = 0` (character = 'a'):
  - Update `d` to `{"a": 0, "b": -1, "c": -1}`.
  - The minimum index among 'a', 'b', and 'c' is `-1`, so `ans += -1 + 1`, resulting in `ans = 0`.
- For `i = 1` (character = 'b'):
  - Update `d` to `{"a": 0, "b": 1, "c": -1}`.
  - The minimum index among 'a', 'b', and 'c' is still `-1`, so `ans += -1 + 1`, resulting in `ans = 0`.
- For `i = 2` (character = 'c'):
  - Update `d` to `{"a": 0, "b": 1, "c": 2}`.
  - Now we have all three characters, and the minimum index is `0`, so `ans += 0 + 1`, resulting in `ans = 1`.
- For `i = 3` (character = 'a'):
  - Update `d` to `{"a": 3, "b": 1, "c": 2}`.
  - The minimum index among 'a', 'b', and 'c' is now `1`, so `ans += 1 + 1`, resulting in `ans = 3`.
- For `i = 4` (character = 'b'):
  - Update `d` to `{"a": 3, "b": 4, "c": 2}`.
  - The minimum index among 'a', 'b', and 'c' is `2`, so `ans += 2 + 1`, resulting in `ans = 6`.
- For `i = 5` (character = 'c'):
  - Update `d` to `{"a": 3, "b": 4, "c": 5}`.
  - The minimum index among 'a', 'b', and 'c' is `3`, so `ans += 3 + 1`, resulting in `ans = 10`.

After the loop finishes, we return the total count, which is `10`.

Throughout this process, we efficiently tracked the latest positions of 'a', 'b', and 'c', calculated the substrings terminating at each character position, and accumulated this to find the total number of substrings containing all three characters.

Python Solution

```python
class Solution:
    def numberOfSubstrings(self, string: str) -> int:
        # Create a dictionary to keep track of the last seen index of 'a', 'b', and 'c'
        last_seen_index = {"a": -1, "b": -1, "c": -1}
        # Initialize answer to store the number of valid substrings
        answer = 0
        # Enumerate over the characters of the string
        for index, char in enumerate(string):
            # Update the last seen index for the current character
            last_seen_index[char] = index
            # Increment the answer by one more than the smallest last seen index among 'a', 'b', and 'c'
            # This is because a valid substring must include at least one of each
            answer += min(last_seen_index.values()) + 1

        # Return the total count of valid substrings that contain at least one of each 'a', 'b', and 'c'
        return answer

```

Java Solution

```java
class Solution {
    public int numberOfSubstrings(String s) {
        // Array to store the latest positions of characters 'a', 'b', and 'c'
        int[] latestPosition = new int[] {-1, -1, -1};

        // This will hold the count of valid substrings
        int answer = 0;

        // Iterate over each character in the string
        for (int i = 0; i < s.length(); ++i) {
            char currentChar = s.charAt(i);

            // Update the latest position of the current character
            latestPosition[currentChar - 'a'] = i;

            // Find the smallest index among the latest positions of 'a', 'b', and 'c'
            // and add 1 to get the count of valid substrings ending with the current character
            int minPosition = Math.min(latestPosition[0], Math.min(latestPosition[1], latestPosition[2]));
            answer += minPosition + 1;
        }

        return answer; // Return the total count of valid substrings
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Function to count the number of substrings containing all three characters 'a', 'b', and 'c'.
    int numberOfSubstrings(string s) {
        // Initialize an array to store the last seen positions of 'a', 'b', and 'c'.
        int lastSeenPositions[3] = {-1, -1, -1};

        // Initialize the answer to 0.
        int substringCount = 0;

        // Iterate over the string.
        for (int index = 0; index < s.size(); ++index) {
            // Update the last seen position for the current character.
            lastSeenPositions[s[index] - 'a'] = index;

            // Find the smallest index among the last seen positions of 'a', 'b', and 'c'.
            // Add 1 because indices are 0-based, and we're interested in the number of elements.
            int minLastSeenPosition = min(lastSeenPositions[0],
                                          min(lastSeenPositions[1], lastSeenPositions[2])) + 1;

            // Add the number of valid substrings ending with the current character.
            // This is possible because any substring ending at the current index
            // and starting before or at the smallest last seen index will contain all three characters.
            substringCount += minLastSeenPosition;
        }

        // Return the total count of valid substrings.
        return substringCount;
    }
};

```

Typescript Solution

```typescript
// Function to count the number of substrings containing all three characters 'a', 'b', and 'c'.
function numberOfSubstrings(s: string): number {
    // Initialize an array to store the last seen positions of 'a', 'b', and 'c'.
    const lastSeenPositions: number[] = [-1, -1, -1];

    // Initialize the answer to 0.
    let substringCount: number = 0;

    // Iterate over the string.
    for (let index = 0; index < s.length; ++index) {
        // Update the last seen position for the current character.
        lastSeenPositions[s.charCodeAt(index) - 'a'.charCodeAt(0)] = index;

        // Find the smallest index among the last seen positions of 'a', 'b', and 'c'.
        const minLastSeenPosition: number = Math.min(lastSeenPositions[0],
                                                     Math.min(lastSeenPositions[1], lastSeenPositions[2])) + 1;

        // Add the number of valid substrings ending with the current character.
        // This is calculated by considering any substrings that end at the current index
        // and start before or at the smallest last seen index, thus including all three characters.
        substringCount += minLastSeenPosition;
    }

    // Return the total count of valid substrings.
    return substringCount;
}

```

Time and Space Complexity

The time complexity of the given code is `O(n)`, where `n` is the length of the string `s`. This is because the code iterates over each character in the string exactly once. Within the loop, updating the dictionary and calculating the minimum value and the cumulative sum is done in constant time.

The space complexity of the code is `O(1)`. The space is constant because the dictionary `d` only stores three key-value pairs regardless of the size of the input string, corresponding to the characters 'a', 'b', and 'c'.