LeetCode 1876: Substrings of Size Three with Distinct Characters

■ Problem Statement:

You are given a string s, and you need to return the number of substrings of size three with all distinct characters.

■ Example:

Input: s = 'xyzzaz'
Output: 1
Explanation: The substrings of size 3 are: 'xyz', 'yzz', 'zza', 'zaz'. Only 'xyz' has all distinct characters.

■ Approach 1: Brute Force (O(n^3))

Check every substring of size 3 and for each substring, check if all characters are unique. This results in O(n^3) time complexity (O(n^2) for generating substrings and O(3) for uniqueness check).

■ *Python (Brute Force):*

```
def countGoodSubstrings(s: str) -> int:
    count = 0
    for i in range(len(s)):
        for j in range(i + 1, len(s) + 1):
            sub = s[i:j]
            if len(sub) == 3 and len(set(sub)) == 3:
                count += 1
    return count

# Example
print(countGoodSubstrings("xyzzaz"))  # Output: 1
```

■ *C++ (Brute Force):*

```
#include <iostream>
#include <unordered_set>
using namespace std;

int countGoodSubstrings(string s) {
    int count = 0;
    for (int i = 0; i < s.size(); i++) {
        for (int j = i + 1; j <= s.size(); j++) {
            string sub = s.substr(i, j - i);
            if (sub.size() == 3) {
                unordered_set<char> st(sub.begin(), sub.end());
                if (st.size() == 3) count++;
            }
        }
    }
    return count;
}

int main() {
    string s = "xyzzaz";
    cout << countGoodSubstrings(s);
}
```

■ *Java (Brute Force):*

```
public class Main {
    public static int countGoodSubstrings(String s) {
        int count = 0;
        for (int i = 0; i < s.length(); i++) {
            for (int j = i + 1; j <= s.length(); j++) {
                String sub = s.substring(i, j);
                if (sub.length() == 3) {
                    if (sub.charAt(0) != sub.charAt(1) &&
                        sub.charAt(0) != sub.charAt(2) &&
                        sub.charAt(1) != sub.charAt(2)) {
                        count++;
                    }
                }
            }
        }
        return count;
    }

    public static void main(String[] args) {
        System.out.println(countGoodSubstrings("xyzzaz")); // Output: 1
    }
}
```

■ Approach 2: Sliding Window (O(n))

We use a sliding window of size 3 to check distinct characters efficiently. Keep track of character frequencies and adjust as the window moves.

■ *Python (Sliding Window):*

```
def countGoodSubstrings(s: str) -> int:
    count = 0
    for i in range(len(s) - 2):
        window = s[i:i+3]
        if len(set(window)) == 3:
            count += 1
    return count

print(countGoodSubstrings("aababcabc"))  # Output: 4
```

■ *C++ (Sliding Window):*

```
#include <iostream>
#include <unordered_map>
using namespace std;

int countGoodSubstrings(string s) {
    int n = s.size(), count = 0;
    unordered_map<char, int> freq;

    for (int i = 0; i < n; i++) {
        freq[s[i]]++;
        if (i >= 3) {
            freq[s[i - 3]]--;
            if (freq[s[i - 3]] == 0) freq.erase(s[i - 3]);
        }
        if (i >= 2 && freq.size() == 3) count++;
    }
    return count;
}

int main() {
    string s = "aababcabc";
    cout << countGoodSubstrings(s); // Output: 4
}
```

■ *Java (Sliding Window):*

```
import java.util.*;

public class Main {
    public static int countGoodSubstrings(String s) {
        int count = 0;
        Map<Character, Integer> freq = new HashMap<>();

        for (int i = 0; i < s.length(); i++) {
            freq.put(s.charAt(i), freq.getOrDefault(s.charAt(i), 0) + 1);

            if (i >= 3) {
                char left = s.charAt(i - 3);
                freq.put(left, freq.get(left) - 1);
                if (freq.get(left) == 0) freq.remove(left);
            }

            if (i >= 2 && freq.size() == 3) count++;
        }
        return count;
    }

    public static void main(String[] args) {
        System.out.println(countGoodSubstrings("aababcabc")); // Output: 4
    }
}
```

■■ Time & Space Complexity:

Brute Force: O(n^3) time, O(1) space
Sliding Window: O(n) time, O(1) space