Documentation

First Unique Character in a String

The problem of finding the first unique character in a string is a common algorithmic challenge that tests one's ability to manipulate and analyze string data efficiently. The goal is to identify the first character in the string that does not repeat and return its index. If no such character exists, the function should return -1. This problem is often encountered in coding interviews and competitive programming due to its straightforward nature and the need for an optimized solution.

Intuition

The intuition behind solving this problem lies in understanding the frequency of characters in the string. To determine the first unique character, we need to know how many times each character appears. By counting the frequency of each character, we can easily identify which characters are unique. Once we have this information, we can iterate through the string and check the frequency of each character to find the first one that appears only once.

Approach

The approach involves two main steps. First, we create a frequency map or dictionary to store the count of each character in the string. This is done by iterating through the string once and updating the count for each character in the dictionary. Second, we iterate through the string again and check the frequency of each character using the dictionary. The first character with a frequency of 1 is the answer, and we return its index. If no such character is found after the iteration, we return -1.

Frequency Counting

Frequency counting is a crucial part of this solution. By using a dictionary, we can efficiently store and retrieve the count of each character. This step ensures that we have a clear picture of how many times each character appears in the string. The frequency map acts as a lookup table, allowing us to quickly determine whether a character is unique or not during the second iteration.

Finding the First Unique Character

Once the frequency map is built, the next step is to find the first unique character. This involves iterating through the string again and checking the frequency of each character in the map. The first character with a frequency of 1 is the answer, and its index is returned immediately. This step ensures that we find the first unique character in the order in which they appear in the string.

Edge Case Handling

Handling edge cases is an essential part of any algorithmic solution. In this problem, the edge case occurs when no unique characters are in the string. For example, in a string like "aabb", all characters repeat, and there is no unique character. In such cases, the function should return -1 to indicate that no unique character exists. This ensures that the solution is robust and handles all possible inputs correctly.

Time Complexity

This solution's time complexity is O(n), where n is the length of the string. We traverse the string twice: once to build the frequency map and once to find the first unique character. Both traversals are linear concerning the length of the string, making the overall time complexity efficient and suitable for large inputs.

Space Complexity

This solution's space complexity is O(1) because the size of the frequency map is bounded by the number of unique characters. Since the problem specifies that the string consists of only lowercase English letters, the maximum size of the frequency map is 26. This constant space usage makes the solution memory-efficient and scalable for large inputs.