<https://leetcode.com/problems/first-unique-character-in-a-string/description/>

Given a string s, find the **first** non-repeating character in it and return its index. If it **does not** exist, return -1.

**Example 1:**

**Input:** s = "leetcode"

**Output:** 0

**Explanation:**

The character 'l' at index 0 is the first character that does not occur at any other index.

**Example 2:**

**Input:** s = "loveleetcode"

**Output:** 2

**Example 3:**

**Input:** s = "aabb"

**Output:** -1

**Constraints:**

1 <= s.length <= 10^5

s consists of only lowercase English letters.

**Attempt 1: 2025-03-29**

**Solution 1: (10 min)**

**Refer to Deepseek**

Here's a Java solution to find the first unique character in a string:

class Solution {

public int firstUniqChar(String s) {

// Create a frequency map to count occurrences of each character

int[] freq = new int[26];

// First pass: count frequency of each character

for (int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

freq[c - 'a']++;

}

// Second pass: find the first character with frequency 1

for (int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

if (freq[c - 'a'] == 1) {

return i;

}

}

// If no unique character found

return -1;

}

}

Time Complexity: O(n)

Space Complexity: O(1)

**Explanation:**

Frequency Array: We use an integer array of size 26 (for lowercase English letters) to count the occurrences of each character in the string.

First Pass: We iterate through the string and increment the count for each character we encounter.

Second Pass: We iterate through the string again and check the frequency of each character. The first character with a frequency of 1 is our answer.

Return -1: If we don't find any character with frequency 1, we return -1.

**Time and Space Complexity:**

Time Complexity: O(n), where n is the length of the string. We make two passes through the string.

Space Complexity: O(1), since we're using a fixed-size array (26 elements) regardless of the input size.

**Alternative Solution (Using HashMap):**

If the input can contain Unicode characters (not just lowercase letters), we can use a HashMap:

class Solution {

public int firstUniqChar(String s) {

HashMap<Character, Integer> map = new HashMap<>();

// Build frequency map

for (int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

map.put(c, map.getOrDefault(c, 0) + 1);

}

// Find the first unique character

for (int i = 0; i < s.length(); i++) {

if (map.get(s.charAt(i)) == 1) {

return i;

}

}

return -1;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

This solution has:

Time Complexity: O(n)

Space Complexity: O(m), where m is the number of unique characters in the string

**Refer to**

[L451.P14.5.Sort Characters By Frequency (Ref.L347)](note://512A6121D907491D8CF8FDD9119E5CC3)

[L2351.First Letter to Appear Twice](note://WEB332128777a8912192f9cabb85a19e1cf)