Solution: First Unique Character in a String

Let's solve the First Unique Character in a String problem using the Knowing What to Track pattern.

We'll cover the following

- Statement
- Solution
  - Time complexity
  - Space complexity

Statement

For a given string of characters, `s`, your task is to find the first non-repeating character and return its index. Return -1 if there's no unique character in the given string.

Constraints:

- Only lowercase english letters are accepted.
- There are no spaces in the string.

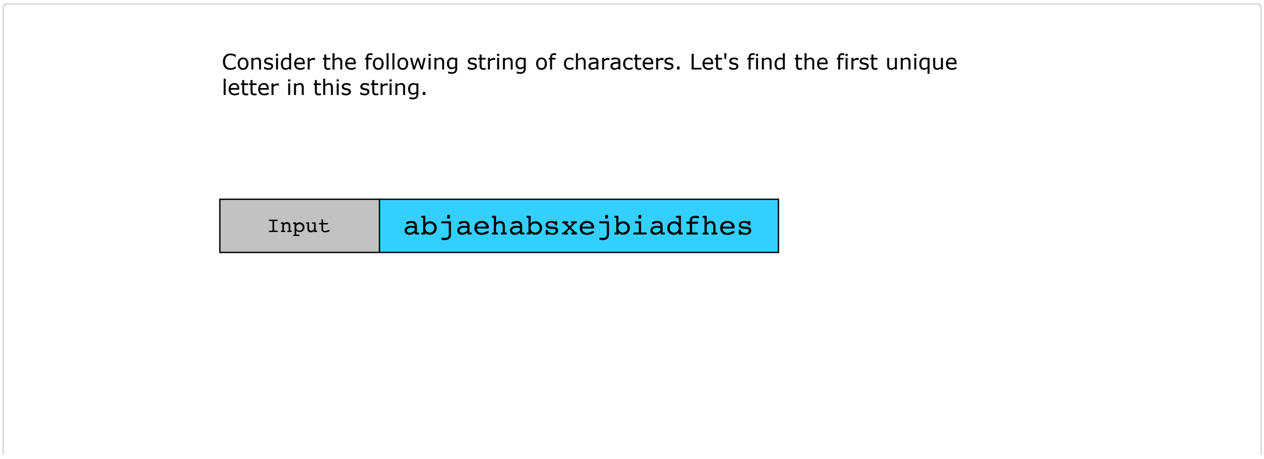
Solution

We need to keep track of the number of occurrences of each character in the string. To achieve this, we can use a hash map to store the character as a key and its number of occurrences in the string as its corresponding value.

The algorithm proceeds through the following steps:

- Create a hash map and start a loop to traverse over the given input string.
- At each iteration, we check if the current character is present in the hash map as a key.
  - If the key exists, we increment the value corresponding to this key character by 1.
  - Otherwise, add this new key-value pair in the hash map and set its value to 1.
- Traverse over the input string to find the character in the hash map whose value equals 1.
  - If it exists, return the index of this character in the string. Otherwise, return -1.

The slide deck below illustrates the key steps of the solution.



Consider the following string of characters. Let's find the first unique letter in this string.

Input	abjaehabsxejbiadfhes

Let’s look at the code for this solution below:

Java

```java
class UniqueCharacter {

    static int firstUniqueChar(String s) {
        HashMap <Character, Integer> wordCount = new HashMap <Character, Integer> ();

        // loop to iterate over the length of input string
        for (int i = 0; i < s.length(); i++) {
            // check if the character exists in the hash map
            char ch = s.charAt(i);
            if (wordCount.containsKey(ch)) {
                // if the character already exists, increase the counter by adding +1
                wordCount.put(ch, wordCount.get(ch) + 1);
            } else {
                // if the character doesn't exists, set the count of letter to 1
                wordCount.put(ch, 1);
            }
        }
```

```java
            // the first character to have a count of 1 should be returned
            char ch = s.charAt(i);
            if (wordCount.get(ch) == 1) {
                return i;
            }
        }
        // return -1 if all occurrences of letters have a count greater than 1
        return -1;
```

First Unique Character in a String

Time complexity

The cost of traversing the length of the input string twice is $O(2n)$, which can be simplified to $O(n)$.

Space complexity

The space complexity of the algorithm above is $O(1)$ because, at any time, a total of 26 keys will be stored in the hash map. This makes it a constant space used to store the frequency of the characters’ occurrence.

← Back
First Unique Character...

Next →
Find All Anagrams in ...

Mark as Completed