题目地址(3. 无重复字符的最长子串)

https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/

入选理由

1. 这是最最经典的滑动窗口题目。滑动窗口有时候是需要计数的，那用什么计数呢？哈哈 下一篇咱就讲双指针，滑动窗口就是双指针中的一个部分。大家可以通过这道题预习一下。

题目描述

给定一个字符串，请你找出其中不含有重复字符的 最长子串 的长度。

示例 1:

输入: "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

示例 2:

输入: "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。
示例 3:

输入: "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

标签

- 双指针
- 滑动窗口
- 哈希表

难度

- 中等

前置知识

- 哈希表
- 双指针

方法1:暴力

我们可以枚举所有的子串。并注意判断是否满足**无重复字符的**条件。

代码

代码支持： JavaScript

Javascript Code:

```
/**
 * @param {string} s
 * @return {number}
 */
var lengthOfLongestSubstring = function (s) {
  let res = 0;
  for (let i = 0; i < s.length; i++) {
    let map = {};
    for (let j = i; j < s.length; j++) {
      if (map[s[j]] !== undefined) {
        break;
      }
      map[s[j]] = true;
      res = Math.max(res, j - i + 1);
    }
  }
  return res;
};
```

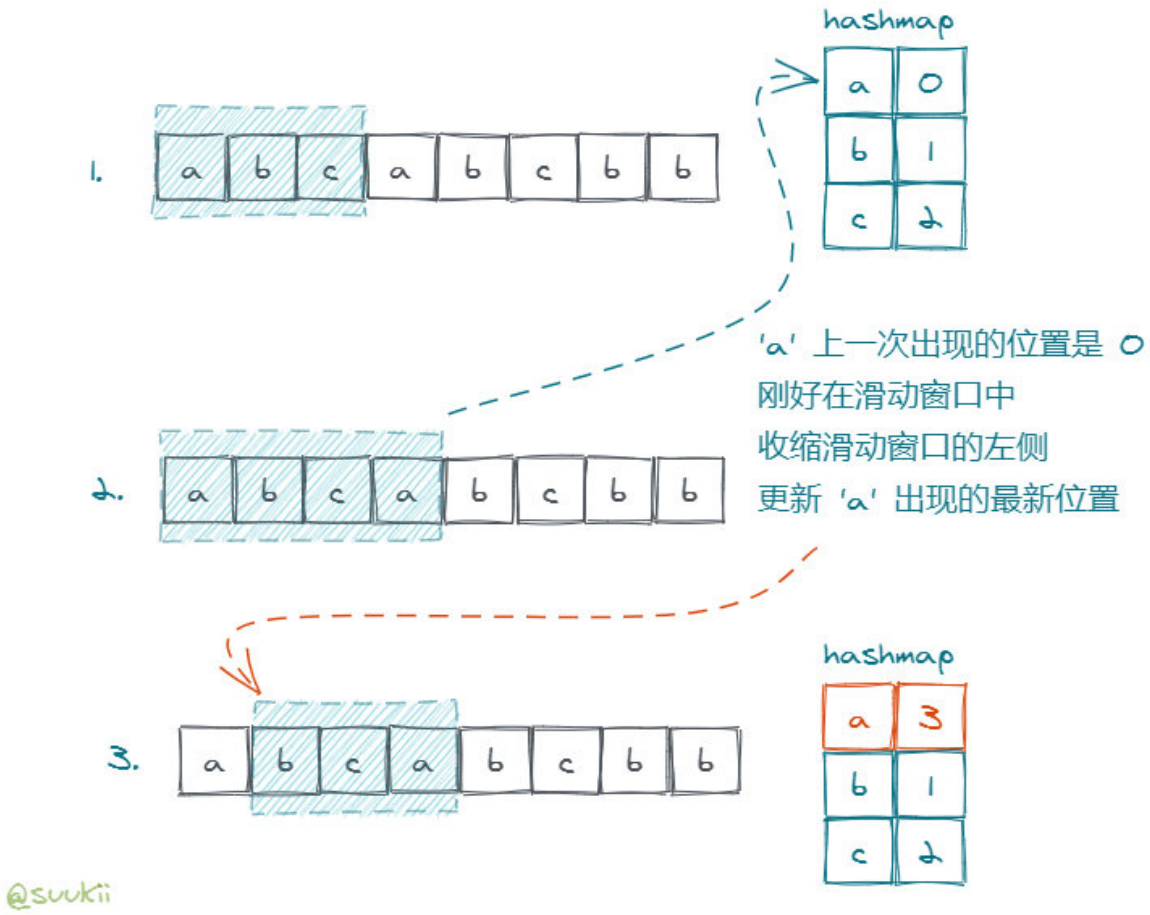
复杂度分析

- 时间复杂度: $O(n^2)$, n为字符串长度
- 空间复杂度: $O(s)$, s为字符集元素个数

方法2:哈希表 + 滑动窗口

思路A

- 维护一个滑动窗口，当窗口中的字符不重复时，继续向右扩大窗口。
- 当遇到重复字符 `d` 时，将窗口左侧收缩到 `d` 字符上次出现的位置 + 1。
- 为了快速找到字符上次出现的位置，我们可以用一个哈希表来记录每个字符最新出现的位置。
- 在滑动窗口遍历数组的过程中用一个变量记录窗口的最大长度。



代码(JavaScript/C++)

JavaScript Code

```
/**
 * @param {string} s
 * @return {number}
 */
var lengthOfLongestSubstring = function (s) {
    const map = {};
    let l = 0,
        r = 0,
```

```
        max = 0;

    while (r < s.length) {
        const pos = map[s[r]];
        // 如果 s[r] 曾在 [l, r] 滑动窗口中出现
        // 就收缩滑动窗口左侧，把 l 指针移动到 s[r] 上次出现的位置 + 1
        if (pos >= l && pos <= r) l = pos + 1;

        // 更新 s[r] 出现的位置
        map[s[r]] = r;
        // 计算滑动窗口大小
        max = Math.max(max, r - l + 1);
        // 滑动窗口继续右移扩张
        r++;
    }
    return max;
};
```

C++ Code

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        unordered_map<char,int> seen;
        int max_len = 0, l = 0, r = 0;
        while (r < s.size()) {
            if (seen.count(s[r]) > 0) {
                int last_pos = seen[s[r]];
                if (last_pos >= l && last_pos <= r) {
                    l = last_pos + 1;
                }
            }
            max_len = max(max_len, r - l + 1);
            seen[s[r]] = r;
            r++;
        }
        return max_len;
    }
};
```

思路B

利用**HashSet**来判断是否有重复字符，并且用一个变量记录起始位置，若set中出现了重复字符，则拿当前字串长度和maxLen取最大值，接着将起始位置**右移至重复字符的后一个位置**，并把之前的字符从set中**去除**。最后maxLen存的值即为最长的不重复的子串，注意最后返回条件还需要判断一次，因为可能题中所给的字串已经是不重复的了。

代码

代码支持： Java

Java Code:

```java
class Solution {

    public int lengthOfLongestSubstring(String s) {

        int left = 0;
        int maxLen = 0;
        Set<Character> set = new HashSet<>();

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

            if (!set.add(s.charAt(i))) {

                maxLen = Math.max(maxLen, set.size());
                while (s.charAt(left) != s.charAt(i)) {

                    set.remove(s.charAt(left));
                    left++;
                }

                left += 1;
            }
        }

        return Math.max(maxLen, set.size());
    }
}
```

复杂度分析

- 时间复杂度: $O(n)$，n为字符串长度
- 空间复杂度：$O(s)$，s为字符集元素个数

上一页 下一页