3. 无重复字符的最长子串

中等 相关标签 相关企业 A文

给定一个字符串 `s` ，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:** s = "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:** s = "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:** s = "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。

提示:

10.3K 9.1K

用滑动窗口解决，借助 unordered_set

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int len = s.length();
        if( s.empty()|| len <= 0) return 0;

        unordered_set<char> charset;
        int maxLen = 0;
        int left = 0;

        for(int right = 0;right<len;right++){
            while(charset.find(s[right])!=charset.end()){
                charset.erase(s[left]);
                left++;
            }
            charset.insert(s[right]);
            maxLen = max(maxLen,right - left +1);
        }
         return maxLen;
    }

};
```

用 unordered_map 解决

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        if(s.empty()||s.length()<=0) return 0;

        unordered_map<char,int> charmap;
        int left = 0;
        int maxlen = 0;

        for(int right = 0;right<s.length();right++){
            if(charmap.find(s[right]) != charmap.end()&&charmap[s[right]] >=
left){
                left = charmap[s[right]]+1;
            }
            charmap[s[right]] = right;
            maxlen = max(maxlen,right-left+1);

        }
        return maxlen;
        }
};
```

为什么使用 `unordered_map` 时需要检查字符是否在左边界之后？

- `unordered_map` 记录每个字符的索引位置。
- 当遇到一个重复的字符时（例如'a'在位置 3 再次出现），需要确认这个重复字符的位置是否在当前窗口的范围内（即是否在 `left` 指针之后）。
- 如果是这样，我们通过移动 `left` 指针跳过这些重复字符，确保窗口内的子串是无重复的。
- 如果不检查左边界，可能会错误地将 `left` 指针移动到已经无效的位置，导致结果不正确。

为什么 `unordered_set` 不需要考虑左边界之后？

- `unordered_set` 只记录窗口内的字符，而不关心这些字符的位置。
- 当发现重复字符时，通过逐步移动 `left` 指针并移除字符，直到窗口内不再有重复字符。
- `unordered_set` 只需要确保窗口内的字符唯一性，不需要处理具体的位置问题，因此不需要检查是否在左边界之后。