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。

来源：力扣（LeetCode）第三题

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

这是一个典型的滑动窗口问题。

解题思路：什么是滑动窗口?

其实就是一个队列,比如例题中的 abcabcbb，进入这个队列（窗口）为 abc 满足题目要求，当再进入 a，队列变成了 abca，这时候不满足要求。所以，我们要移动这个队列！

如何移动？

我们只要把队列的左边的元素移出就行了，直到满足题目要求！

一直维持这样的队列，找出队列出现最长的长度时候，求出解！

时间复杂度：O(n)

这样做的时间复杂度会大幅度降低！！！！

代码如下：（Python）

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        Max_len = 0
        string = set()
        if len(s) == 1:  # 字符串只有一个，直接返回，不要在算下去了
```

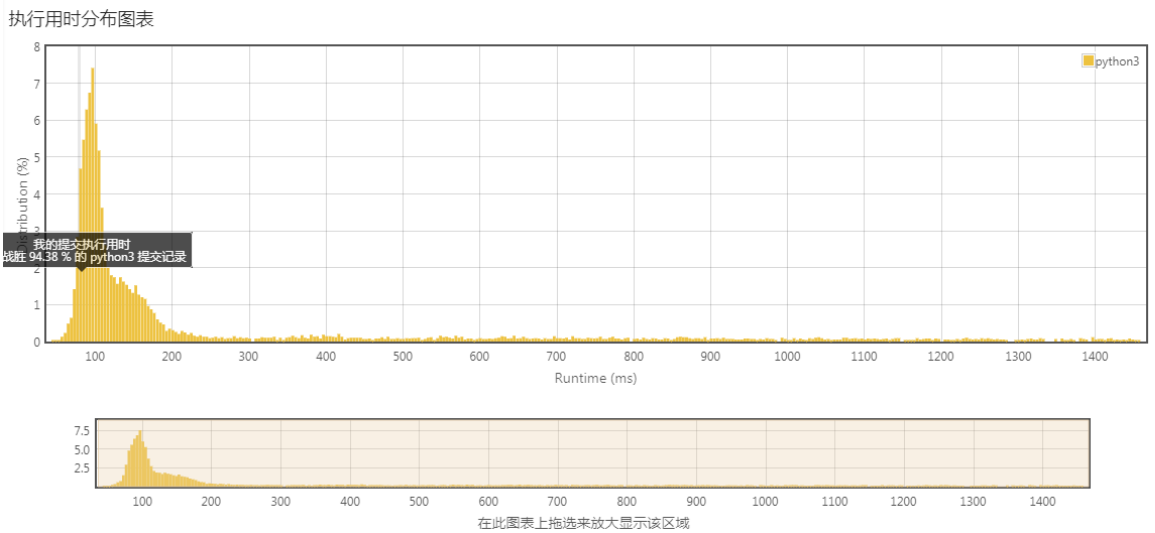
```
            return 1
        i = 0
        for char in s:
            while char in string:  # 如果遍历到一个已经有了的字符，说明这一段字符串结束，将其全部出队，从下一个不重复的字符开始重新运算。
                if Max_len < len(string):  # 该段字符串的长度如果比之前的大，那么就代替要返回的值，否则全部扔掉。
                    Max_len = len(string)
                string.remove(s[i])
                i += 1
            string.add(char)
            if s[-1] == char and Max_len < len(string):  # 如果传入的字符串没有一个是重复的话
                Max_len = len(string)
        return Max_len
```

如果不使用队列解决方法的话，时间复杂度是O（n^3）

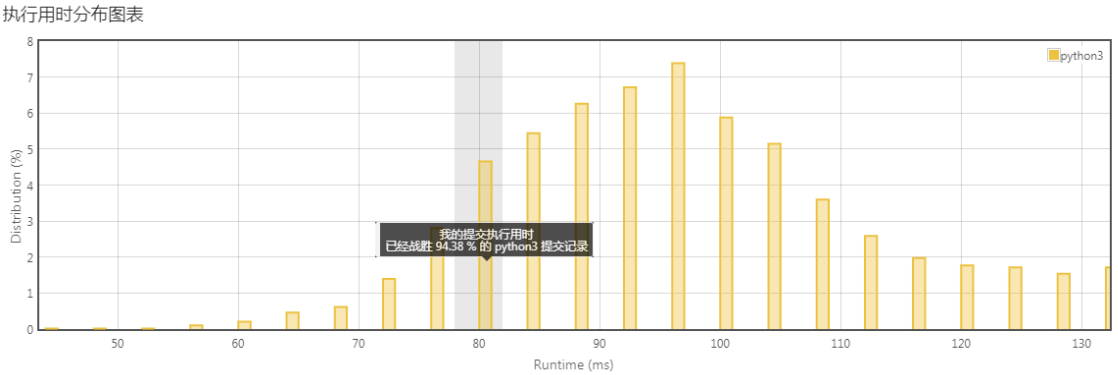
代码如下：（Python）

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        Max_len = 0
        if len(s) == 1:
            return 1
        for i in range(len(s)):
            string = []
            for char in s[i:]:
                if char in string:
                    if Max_len < len(string):
                        Max_len = len(string)
                    break
                string.append(char)
                if s[-1] == char and Max_len < len(string):
                    Max_len = len(string)
        return Max_len
```

可能直接看代码不是非常的直观，不知道第二种方法到底有多慢，我以Python为例，截取了一张所有使用Python的同学提交的代码的时间分布表。



由于LeetCode后期强制写了一个测试：一段几千字的字符串，还设置了代码执行时间限制。导致如果用第二种方法的话会不通过，所以后期所有人用的都是滑动窗口的方法。



可以看到我写的代码只用了80ms，最短的用了44ms

执行用时为 44 ms 的范例

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        dic = {}
        length = 0
        begin = 0
        for index, string in enumerate(s):
            if (string in dic) and (dic[string] >= begin):
                lth =  index - begin
                length = lth if lth > length else length
                begin = dic[string] + 1
            dic[string] = index
        lth =  len(s) - begin
        length = lth if lth > length else length
        return length
```

而如果使用第二种方法的话，

执行用时为 **1456 ms** 的范例

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        """
        :type s: str
        :rtype: int
        """
        longStr = set([])
        curStr = set([])
        max = 0
        for i in range(len(s)):
            longStr = set(s[i])
            curStr = set(s[i])
            for j in range(i + 1, len(s)):
                longStr.add(s[j])
                if len(longStr) > len(curStr):
                    curStr.add(s[j])
                else:
                    break
            if len(longStr) > max:
                max = len(longStr)
        return max
```

用了整整1456ms，这平均至少也相差了10倍左右的速度。

总结：

在思考问题时，我们需要灵活的使用所学过的知识，尤其是数据结构，就比如这个问题，使用队列就可以很轻松的解决，并且运行效率非常的高效。