

3. Sliding Window

121. Best Time to Buy and Sell Stock

🕒 Created	@June 23, 2021 9:07 PM
📌 Difficulty	Easy
📄 LC Url	https://leetcode.com/problems/best-time-to-buy-and-sell-stock/
📌 Importance	
🏷️ Tag	Sliding Window
📺 Video	

You are given an array `prices` where `prices[i]` is the price of a given stock on the `i` th day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

Constraints:

- `1 <= prices.length <= 10 5`
- `0 <= prices[i] <= 10 4`

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        max_profit, min_price = 0, float('inf')

        for price in prices:
            min_price = min(min_price, price)
            max_profit = max(max_profit, price - min_price)

        return max_profit
```

3. Longest Substring Without Repeating Character

🕒 Created	@July 14, 2020 5:22 AM
📌 Difficulty	Medium
🔗 LC Url	https://leetcode.com/problems/longest-substring-without-repeating-characters/
📌 Importance	
🏷️ Tag	NEET Sliding Window Two pointers
📺 Video	https://www.youtube.com/embed/wiGpQwVHdE0

Given a string `s`, find the length of the **longest substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

Constraints:

- `0 <= s.length <= 5 * 104`
- `s` consists of English letters, digits, symbols and spaces.

Solution

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # https://leetcode.cn/problems/longest-substring-without-repeating-characters/solution/wu-zhong-fu-zi-fu-de-zui-chang-zi-chuan-by-l

        cur_chars = set()
        n = len(s)


        right, length = -1, 0
        for left in range(n):
            if left != 0:
                cur_chars.remove(s[left - 1])

            while right + 1 < n and s[right + 1] not in cur_chars:
                cur_chars.add(s[right + 1])
                right += 1

            length = max(length, right - left + 1)
```

```
return length
```

力扣

 <https://leetcode.cn/problems/longest-substring-without-repeating-characters/solution/wu-zhong-fu-zi-fu-de-zui-chang-zi-chuan-by-leetc-2/>

复杂度分析


- 时间复杂度： $O(N)$ ，其中 N 是字符串的长度。左指针和右指针分别会遍历整个字符串一次。
- 空间复杂度： $O(|\Sigma|)$ ，其中 Σ 表示字符集（即字符串中可以出现的字符）， $|\Sigma|$ 表示字符集的大小。在本题中没有明确说明字符集，因此可以默认为所有 ASCII 码在 $[0, 128)$ 内的字符，即 $|\Sigma| = 128$ 。我们需要用到哈希集合来存储出现过的字符，而字符最多有 $|\Sigma|$ 个，因此空间复杂度为 $O(|\Sigma|)$ 。

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # 作者: seventeenth
        # 链接: https://leetcode.cn/problems/longest-substring-without-repeating-characters/solution/zen-yao-yong-hua-dong-chuang-kou-wei-he
        left, right, length = 0, 0, 0
        cur_chars = set()

        for i in range(len(s)):
            while s[i] in cur_chars:
                cur_chars.remove(s[left])
                left += 1
            cur_chars.add(s[i])
            length = max(length, right - left + 1)
            right += 1

        return length
```

力扣

 <https://leetcode.cn/problems/longest-substring-without-repeating-characters/solution/zen-yao-yong-hua-dong-chuang-kou-wei-he-35418/>

图解算法

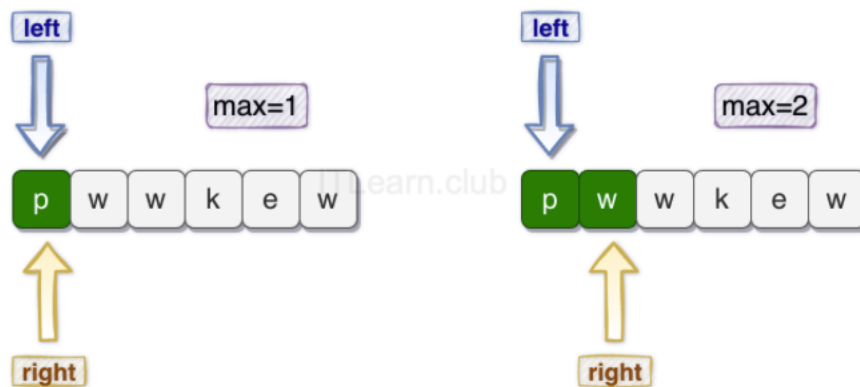
定义 `left` 指针、`right` 指针分别表示窗口的左端、右端；

[`left` , `right`] 区间内的字符串用 `HashSet` 实现判断重复操作，

随着 [`left` , `right`] 区间的变化对 `HashSet` 中的元素进行增减；

定义 `max` 变量用来存储 不重复子串最大长度 作为结果返回。

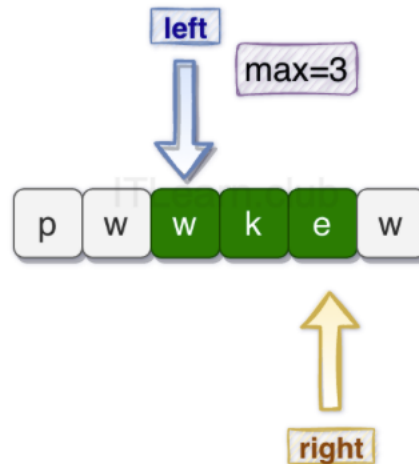
1. `left` 不变，`right` 向右移动，扩大 [`left` , `right`] 区间范围，同时更新 `max`



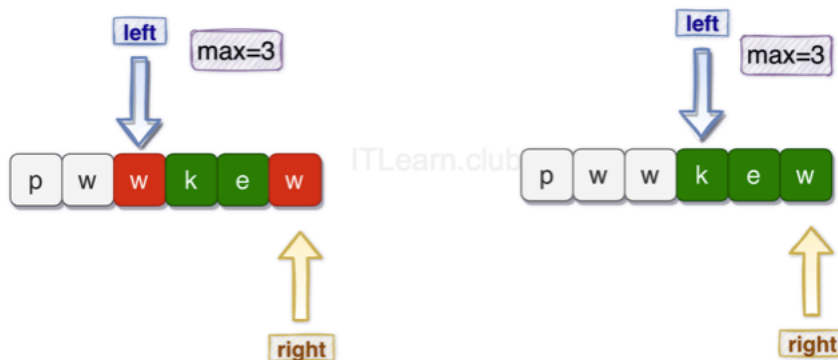
2. 继续移动 `right` 指针，发现区间内出现重复字符；移动 `left` 指针来消除重复



3. 发现更大的 不重复区间长度，更新 `max`



4. 继续移动 `right` 指针，发现区间内出现重复字符；移动 `left` 指针来消除重复



<https://www.youtube.com/watch?v=9VcYiqTqzUY>

```

class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s == null || s.length() == 0) return 0;
        int left = 0, right = 0;
        int n = s.length();
        boolean[] used = new boolean[128];

        int max = 0;
        while (right < n) {
            if (used[s.charAt(right)] == false) {
                used[s.charAt(right)] = true;
                right++;
            } else {
                max = Math.max(max, right-left);
                while (left < right && s.charAt(right) != s.charAt(left)) {
                    used[s.charAt(left)] = false;
                    left++;
                }
                left++;
                right++;
            }
        }
        max = Math.max(max, right-left);
        return max;
    }
}

```

424. Longest Repeating Character Replacement

🕒 Created	@February 3, 2022 11:13 AM
📌 Difficulty	Medium
📄 LC Url	https://leetcode.com/problems/longest-repeating-character-replacement/
📌 Importance	****
🏷️ Tag	NEET Sliding Window String
📺 Video	https://www.youtube.com/watch?v=gqXU1UyA8pk

You are given a string `s` and an integer `k`. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most `k` times.

Return the length of the longest substring containing the same letter you can get after performing the above operations.

Example 1:

```
Input: s = "ABAB", k = 2
Output: 4
Explanation: Replace the two 'A's with two 'B's or vice versa.
```

Example 2:

```
Input: s = "AABABBA", k = 1
Output: 4
Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.
```

Constraints:

- `1 <= s.length <= 10 5`
- `s` consists of only uppercase English letters.
- `0 <= k <= s.length`

Solution

$O(n)$

```
class Solution:
    def characterReplacement(self, s: str, k: int) -> int:
        count = {}
        res = 0

        left = 0
        max_freq = 0

        for right in range(len(s)):
            count[s[right]] = count.get(s[right], 0) + 1
            max_freq = max(max_freq, count[s[right]])

            while (right - left + 1) - max_freq > k:
                count[s[left]] -= 1
                left += 1

            res = max(res, right - left + 1)

        return res
```

567. Permutation in String

🕒 Created	@October 9, 2022 9:56 PM
📌 Difficulty	Medium
🔗 LC Url	https://leetcode.com/problems/permutation-in-string/
📌 Importance	
🏷️ Tag	NEET Sliding Window
📺 Video	

Given two strings `s1` and `s2`, return `true` if `s2` contains a permutation of `s1`, or `false` otherwise.

In other words, return `true` if one of `s1`'s permutations is the substring of `s2`.

Example 1:

```
Input: s1 = "ab", s2 = "eidbaooo"
Output: true
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

```
Input: s1 = "ab", s2 = "eidboao"
Output: false
```

Constraints:

- `1 <= s1.length, s2.length <= 104`
- `s1` and `s2` consist of lowercase English letters.

Solution

```

class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
        m1 = len(s1)
        m2 = len(s2)

        if m1 > m2:
            return False

        cnt1 = [0] * 26
        cnt2 = [0] * 26

        for i in range(m1):
            cnt1[ord(s1[i]) - ord('a')] += 1
            cnt2[ord(s2[i]) - ord('a')] += 1

        if cnt1 == cnt2:
            return True

        for i in range(m1, m2):
            cnt2[ord(s2[i - m1]) - ord('a')] -= 1
            cnt2[ord(s2[i]) - ord('a')] += 1
            if cnt1 == cnt2:
                return True

        return False

```

76. Minimum Window Substring

🕒 Created	@February 4, 2022 8:10 AM
⬇️ Difficulty	Hard
≡ LC Url	https://leetcode.com/problems/minimum-window-substring/
⬇️ Importance	****
⋮ Tag	Sliding Window String
≡ Video	https://www.youtube.com/watch?v=qvou9R_7m10&list=PL2rWx9cCzU84DLLKckx-9kLQZzieHo3r_&index=5

Given two strings `s` and `t` of lengths `m` and `n` respectively, return the **minimum window substring** of `s` such that every character in `t` (**including duplicates**) is included in the window. If there is no such substring, return the empty string `""`.

The testcases will be generated such that the answer is **unique**.

A **substring** is a contiguous sequence of characters within the string.

Example 1:

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.
```

Example 2:

```
Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.
```

Example 3:

Input: s = "a", t = "aa"

Output: ""

Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.

Constraints:

- `m == s.length`
- `n == t.length`
- `1 <= m, n <= 105`
- `s` and `t` consist of uppercase and lowercase English letters.

Follow up:

Could you find an algorithm that runs in

$O(m + n)$

time?

Solution

```
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if not t or not s:
            return ''

        dict_t = Counter(t)
        required = len(dict_t)

        filtered_s = []
        for i, char in enumerate(s):
            if char in dict_t:
                filtered_s.append((i, char))

        left, right = 0, 0
        formed = 0
        window_counts = {}
        ans = [float('inf'), None, None]

        while right < len(filtered_s):
            char, char_i = filtered_s[right]
            window_counts[char] = window_counts.get(char, 0) + 1

            if window_counts[char] == dict_t[char]:
                formed += 1

            while left < right and (not char in window_counts or window_counts[char] > dict_t[char]):
                char, left_i = filtered_s[left]
                window_counts[char] -= 1
                left = left_i + 1

            if formed == required:
                ans[0] = min(ans[0], right - left + 1)
                ans[1] = left
                ans[2] = right
                right += 1
```

```

character = filtered_s[right][1]
window_counts[character] = window_counts.get(character, 0) + 1

if window_counts[character] == dict_t[character]:
    formed += 1

while left <= right and formed == required:
    character = filtered_s[left][1]

    start = filtered_s[left][0]
    end = filtered_s[right][0]
    if end - start + 1 < ans[0]:
        ans = (end - start + 1, start, end)

    window_counts[character] -= 1
    if window_counts[character] < dict_t[character]:
        formed -= 1
    left += 1

    right += 1

return '' if ans[0] == float('inf') else s[ans[1]: ans[2] + 1]

```