3. Longest Substring Without Repeating Characters

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 = "bbbbb"
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
```

Example 4:

```
Input: s = ""
Output: 0
```

Constraints:

- `0 <= s.length <= 5 * 10^4`
- `s` consists of English letters, digits, symbols and spaces.

最长无重复子字符串

String, Two pointers, variant size sliding window

1. boolean[128] to store the appearance of character, two pointers move same direction
2. if not, update the character to true and move right forward

3. if yes, update the MAX and move left until it meets with the same character **(move the left until all preincluded elements are out of the sliding window)** finally move both left and right one step
4. update the MAX, and return it

---

76. Minimum Window Substring

Given two strings `s` and `t`, return *the minimum window in `s` which will contain all the characters in `t`*. If there is no such window in `s` that covers all characters in `t`, return *the empty string `""`*.

Note that If there is such a window, it is guaranteed that there will always be only one unique minimum window in `s`.

Example 1:

```
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
```

Example 2:

```
Input: s = "a", t = "a"
Output: "a"
```

Constraints:

- `1 <= s.length, t.length <= 10^5`
- `s` and `t` consist of English letters.

Follow up: Could you find an algorithm that runs in `O(n)` time?

最小窗口的字串

variant size sliding window, String, two pointers

1. store the frequency of target string
2. find the left edge of the first feasible index (char in source string also in target string), move right to left
3. **while (right < source string length)**, update the frequency of source string, increase the count when a valid char is found, **move right** to the next feasible index
4. **while (count equals target string lenth)**, update the result, decrease the count and the frequency of source string, and **move left** to the next feasible index

5. increase acts are opposite to the decrease acts

---

209. Minimum Size Subarray Sum

Given an array of **n** positive integers and a positive integer **s**, find the minimal length of a **contiguous** subarray of which the sum ≥ **s**. If there isn't one, return 0 instead.

Example:

```
Input: s = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: the subarray [4,3] has the minimal length under the problem constraint.
```

Follow up:
If you have figured out the *O*(*n*) solution, try coding another solution of which the time complexity is *O*(*n* log *n*).

最小的子字符串之和

Array, Two pointers, variant size sliding window

1. two pointers to iterate the array, accumulate the values
2. if the accumulated sum is greater than the criteria, then update the result, move the left forward to remove the element out of the sliding window, and move the right forward to add new value into the sum

---