

■ Longest Substring Without Repeating Characters

The sliding window classic — expand right, shrink left

● String

■ MEDIUM

■ What You Will Learn

- The sliding window technique with a HashSet
- Why a window expands greedily and shrinks on conflict
- How to track the maximum window size seen
- Optimized version using HashMap for O(n) guaranteed

■ Intuition First

Imagine a caterpillar moving along the string. Its head (right pointer) moves forward eating characters. If it tries to eat a character it already has inside it, its tail (left pointer) moves forward until it spits out the duplicate. The longest the caterpillar ever gets is your answer.

■ Sliding Window Intuition

A window [left, right] represents the current valid substring. Keep a set of chars in the window. Expand right freely. When duplicate found, shrink from left until no duplicates.

■ Visual — Step by Step

```
s = 'abcabcbb'

right=0: add 'a' → window=[a]          len=1  max=1
right=1: add 'b' → window=[a,b]        len=2  max=2
right=2: add 'c' → window=[a,b,c]      len=3  max=3
right=3: 'a' exists! → remove 'a', left=1
          add 'a' → window=[b,c,a]      len=3  max=3
right=4: 'b' exists! → remove 'b', left=2
          add 'b' → window=[c,a,b]      len=3  max=3
right=5: 'c' exists! → remove 'c', left=3
          add 'c' → window=[a,b,c]      len=3  max=3
right=6: 'b' exists! → remove 'a','b', left=5
          add 'b' → window=[c,b]        len=2  max=3
right=7: 'b' exists! → remove 'c', left=6
          'b' still exists, remove 'b', left=7
          add 'b' → window=[b]          len=1  max=3

Answer: 3
```

■ Java Code

```
1  class Solution {
2      public int lengthOfLongestSubstring(String s) {
3          Set<Character> set = new HashSet<>();
4          int left = 0, maxlen = 0;
5
6          for (int right = 0; right < s.length(); right++) {
7              char c = s.charAt(right);
8
9              // Shrink window from left until no duplicate
10             while (set.contains(c)) {
11                 set.remove(s.charAt(left));
12                 left++;
13             }
14             set.add(c);
15             maxlen = Math.max(maxlen, right - left + 1);
16         }
17         return maxlen;
18     }
19 }
```

■■ Complexity

Type	Value	Meaning
Time	$O(n)$	Each char added and removed at most once. n =string length
Space	$O(\min(n,m))$	Set stores at most $\min(n,26)$ chars — m =charset size

■■ Common Mistakes

■■ Mistake

Resetting left to 0 when a duplicate is found. You should only move left PAST the previous occurrence of the duplicate, not all the way back to 0.

■ Optimized Version

Use `HashMap<Character,Integer>` to store last index of each char. When duplicate found: `left = max(left, lastIndex[c]+1)`. This jumps left directly instead of shrinking one by one.

■ Quick Revision

✓	Sliding window: left and right pointers on same string
✓	HashSet tracks characters in current window

✓	Duplicate found: shrink from left until clean
✓	$\text{maxLen} = \max(\text{maxLen}, \text{right-left}+1)$ each step
✓	Time $O(n)$, Space $O(\min(n,m))$