# Dry Run:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Step** | **right** | **char** | **freq[char]** | **Duplicate?** | **Action** | **left** | **Window** | **Length** | **maxLen** |
| 1 | 0 | a | 1 | ❌ | Expand right, update maxLen | 0 | "a" | 1 | 1 |
| 2 | 1 | b | 1 | ❌ | Expand right, update maxLen | 0 | "ab" | 2 | 2 |
| 3 | 2 | c | 1 | ❌ | Expand right, update maxLen | 0 | "abc" | 3 | 3 |
| 4 | 3 | a | 2 | ✅ | Shrink left until a's freq = 1 (left++) | 1 | "bca" | 3 | 3 |
| 5 | 4 | b | 2 | ✅ | Shrink: left++ until b's freq = 1 | 2 | "cab" | 3 | 3 |
| 6 | 5 | c | 2 | ✅ | Shrink: left++ until c's freq = 1 | 3 | "abc" | 3 | 3 |
| 7 | 6 | b | 2 | ✅ | Shrink: left++ until b's freq = 1 | 4 | "cb" | 2 | 3 |
| 8 | 7 | b | 2 | ✅ | Shrink: left++ until b's freq = 1 | 6 | "b" | 1 | 3 |

# Solution:

public class Solution {

    public int lengthOfLongestSubstring(String s) {

        int[] freq = new int[128];  // ASCII characters

        int left = 0, right = 0;

        int maxLen = 0;

        while (right < s.length()) {

            char c = s.charAt(right);

            freq[c]++;

            // If duplicate found, shrink from the left

            while (freq[c] > 1) {

                freq[s.charAt(left)]--;

                left++;

            }

            maxLen = Math.max(maxLen, right - left + 1);

            right++;

        }

        return maxLen;

    }

}