

**STRING**

# Problem – 3. Longest Substring Without Repeating Characters

Medium



LeetCode

[leetcode.com/problems/longest-substring-without-repeating-characters](https://leetcode.com/problems/longest-substring-without-repeating-characters)

## Problem Statement

- You are given a string and the goal is to find the longest substring without repeating characters

- **Example**

**Input:** "abcbd"

**Output:** 4 (abcd since "b" is repeated)

# Solution – 3. Longest Substring Without Repeating Characters

Medium



LeetCode

[leetcode.com/problems/longest-substring-without-repeating-characters](https://leetcode.com/problems/longest-substring-without-repeating-characters)

## Solution

- Use sliding window algorithm (left and right)
- Loop through the string
- Try to find if the current character is already added by using unordered set or bitmap
- If added, remove from the set alongside with others using left pointer
- If not, add to the unordered set or bitmap
- Maximum length will be  $\text{right} - \text{left} + 1$

# Example – 3. Longest Substring Without Repeating Characters

Medium



LeetCode

[leetcode.com/problems/longest-substring-without-repeating-characters](https://leetcode.com/problems/longest-substring-without-repeating-characters)

## Example

- String: abcbd. Our goal is to return 3 (**abc**bd)
- Initialize **maxLength = 0**
- Loop through the string

**Iteration 1:** left = 0, right = 0, string[left] = 'a',

bitmap = ['a'] ('a' is not in bitmap, add), **maxLength = max(maxLength, right - left + 1) = 1**

**Iteration 2:** left = 0, right = 1, string[right] = 'b'

bitmap = ['a','b'], **maxLength = 2**

**Iteration 3:** left = 0, right = 2, string[right] = 'c'

bitmap = ['a','b','c'], **maxLength = 3**

**Iteration 4:** left = 0, right = 3, string[right] = 'b'

bitmap = ['a','b','c','b']

'b' is already in the bitmap. start "clearing" the character using left:

**Iteration 4a:** left = 0, string[left] = 'a' is different from 'b', so remove 'a'

bitmap = ['b','c','b']

**Iteration 4b:** left = 1, string[left] = 'b' is the same as the repeated one, remove

bitmap = ['c','b']

**Iteration 5:** left = 1, right = 4, string[right] = 'd'

bitmap = ['c','b','d']

# Code – 3. Longest Substring Without Repeating Characters

Medium

## Code (unordered\_set)

- Use unordered\_set when question requires unicode chars

```
int lengthOfLongestSubstring(string s) {
    int maxLength = 0;
    int left = 0, right = 0;
    // track the seen characters
    unordered_set<char> seen;
    for (right = 0; right < s.size(); ++right) {
        char currentChar = s[right];
        // if currentChar is in the set, clean
        // the character and everything from left of it
        // basically, reset the longest substring
        while (seen.count(currentChar)) {
            char c = s[left];
            seen.erase(c);
            left++;
        }
        // insert the current read character
        seen.insert(currentChar);
        // set max length
        maxLength = max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

# Code – 3. Longest Substring Without Repeating Characters

Medium

## Code (bitmap)

- Using bitset: create a bitmask with 128 bits where each bit represent a character
- Optimal solution for ASCII since ASCII size is 127 characters
- Unicode / UTF-8 can represent over 1.1 million characters, so use **unordered\_set** approach instead

```
int lengthOfLongestSubstring(string s) {
    std::bitset<128> bitmask;
    uint32_t left = 0;
    uint32_t maxLength = 0;

    for (uint32_t right = 0; right < s.length(); ++right) {
        uint32_t bitIndex = s[right];
        // if char is already in the bitmask, move left until we reset the bits
        while (bitmask.test(bitIndex)) {
            bitmask.reset(s[left]);
            ++left;
        }

        bitmask.set(bitIndex);
        maxLength = std::max(maxLength, right - left + 1);
    }
    return maxLength;
}
```

# Problem – 424. Longest Repeating Character Replacement

Medium

 [leetcode.com/problems/longest-repeating-character-replacement](https://leetcode.com/problems/longest-repeating-character-replacement)

## Problem

- You are given a **string s** and an **integer k**
- You can replace one character by any other uppercase English character **k** times
- Return the longest substring with the same character

- **Example:**

**Input:**

`s = "ABAB", k = 2`

**Output:** 4

Replace the two 'A's with two 'B's or vice versa.

# Problem – 424. Longest Repeating Character Replacement

Medium

 [leetcode.com/problems/longest-repeating-character-replacement](https://leetcode.com/problems/longest-repeating-character-replacement)

## Solution

- Start with two pointers: left and right
- Keep track of the frequencies of each letter in a **vector<int>** since we know there are 26 characters
- Initialize **maxFreq** to keep track of the letter with maximum frequency
- Initialize **maxLength** to keep track of the maximum substring
- Go over the string, and for each iteration:
  - calculate the windowSize
  - calculate the maximum frequency
  - check how many replacements is needed. That is, windowSize - maxFreq
  - if no replace can be done ( $k < \text{replaces}$ ) then move left pointer to the right



# Problem – 424. Longest Repeating Character Replacement

Medium

 [leetcode.com/problems/longest-repeating-character-replacement](https://leetcode.com/problems/longest-repeating-character-replacement)

**Code** Time:  $O(n)$  Space:  $O(1)$

```
int characterReplacement(string s, int k) {
    int left = 0;
    int maxLength = 0;
    int maxFreq = 0;
    vector<int> freq(26, 0);
    for (int right = 0; right < s.size(); ++right) {
        int index = s[right] - 'A';
        int windowSize = right - left + 1;
        // keep track of the frequencies
        freq[index]++;
        maxFreq = max(maxFreq, freq[index]);

        // check if the subwindow need to change
        int needReplace = windowSize - maxFreq;
        if (k < needReplace) {
            // need to move sub window
            int leftIndex = s[left] - 'A';
            freq[leftIndex]--;
            left++;
            windowSize = right - left + 1;
        }
        maxLength = max(maxLength, windowSize);
    }
    return maxLength;
}
```

# Problem – 76. Maximum Window Substring

Hard

 [leetcode.com/problems/minimum-window-substring](https://leetcode.com/problems/minimum-window-substring)

**Problem Statement / Solution / Code**   Time:  $O(-)$    Space:  $O(-)$

■ ...

# Problem – 242. Valid Anagram

Easy



LeetCode

[leetcode.com/problems/valid-anagram](https://leetcode.com/problems/valid-anagram)

## Problem

- You are given two strings **s** and **t**
- Return true if **t** is an anagram of **s**

- **Example:**

t = word

s = dwor

**Output:** true

both have the same number of same characters

# Problem – 242. Valid Anagram

Easy



LeetCode

[leetcode.com/problems/valid-anagram](https://leetcode.com/problems/valid-anagram)

## Solution

- Initialize a vector of integers to keep track of the count of each letter
- Loop over **s** and increase the count of each character found
- Then, loop over **t** and decrease the count of each character found
- Finally, loop over the vector and if there is one count greater than 0, return false

# Problem – 242. Valid Anagram

Easy



LeetCode

[leetcode.com/problems/valid-anagram](https://leetcode.com/problems/valid-anagram)

**Code** Time:  **$O(n)$**  Space:  **$O(1)$**

```
bool isAnagram(string s, string t) {  
    // count the number of characters in 's', store in a vector  
    // go over the vector and check if it's empty  
    vector<int> letters(26);  
    for (const auto& c : s) {  
        letters[c - 'a']++;  
    }  
    for (const auto& c : t) {  
        letters[c - 'a']--;  
    }  
    for (const auto& c : letters) {  
        if (c != 0) return false;  
    }  
    return true;  
}
```

# Problem – 49. Group Anagrams

Medium

 [leetcode.com/problems/group-anagrams](https://leetcode.com/problems/group-anagrams)

**Problem Statement / Solution / Code**   Time:  $O(-)$    Space:  $O(-)$

■ ...

# Problem – Valid Parentheses

Easy



LeetCode

[leetcode.com/problems/valid-parentheses](https://leetcode.com/problems/valid-parentheses)

## Problem Statement

- You are given a string containing only the characters '(', ')', '{', '}', '[' and ']'
- A valid input have closed brackets by its own type

- **Example**

`()[]{} → valid`

`[]{}( → invalid`

`{()} → valid`

# Solution – Valid Parentheses

Easy



LeetCode

[leetcode.com/problems/valid-parentheses](https://leetcode.com/problems/valid-parentheses)

## Solution

- Loop through the string
- If **open** brackets (**{** push to a stack
- If **closed** brackets:
  - **pop** the last added bracket
  - **check** if the **closed** bracket corresponds to the **popped** bracket
  - if not, return false
- after the loop, **return true** if the **size** of the stack is empty (all brackets closed)



# Code – Valid Parentheses

Easy



LeetCode

[leetcode.com/problems/valid-parentheses](https://leetcode.com/problems/valid-parentheses)

## Code Time: $O(n)$ Space: $O(n)$

```
bool isValid(string s) {  
    // stack (LIFO)  
    std::stack<char> brackets;  
    // O(n)  
    for (int i = 0; i < s.size(); ++i) {  
        char bracket = s[i];  
        if (bracket == '(' || bracket == '[' || bracket == '{') {  
            brackets.push(bracket);  
        } else {  
            if (brackets.size() == 0) return false;  
            char lastBracket = brackets.top();  
            if (bracket == ')' && lastBracket != '(') return false;  
            if (bracket == '}' && lastBracket != '{') return false;  
            if (bracket == ']' && lastBracket != '[') return false;  
            brackets.pop();  
        }  
    }  
    // all brackets must be closed  
    return brackets.size() == 0;  
}
```

# Problem – 125. Valid Palindrome

Easy

 [leetcode.com/problems/valid-palindrome](https://leetcode.com/problems/valid-palindrome)

**Problem Statement / Solution / Code**   Time:  $O(-)$    Space:  $O(-)$

■ ...

# Problem – Minimum Number of Increments on Subarrays

Hard



LeetCode

[leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array](https://leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array)

## Problem Statement

- You are given an array of integers initialized with zeros (e.g. **[0,0,0,0]**)
- The goal is to reach some target (e.g. **[1, 2, 2, 3]**)
- The valid operations is to increment a subarray by one
- The output is the total number of operations

In this case:

**[1,1,1,1]** → increment the subarray starting from 0 to total size

**[1,2,2,2]** → increment the subarray starting from 1 to total size

**[1,2,2,3]** → increment the subarray starting and ending from the last element

**Output:** 3 (total number of operations)

# Solution – Minimum Number of Increments on Subarrays

Hard



LeetCode

[leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array](https://leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array)

## Solution

- Explain...

# Code [2] – Minimum Number of Increments on Subarrays

Hard



LeetCode

[leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array](https://leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array)

## Code (optimized)

```
int minNumberOperations(vector<int>& target) {  
    return target[0] +  
        inner_product(target.begin() + 1, target.end(),  
            target.begin(), 0,  
            plus<int>(),  
            [](int curr, int prev) { return max(curr - prev, 0); });  
}
```

# Code – Minimum Number of Increments on Subarrays

Hard



LeetCode

[leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array](https://leetcode.com/problems/minimum-number-of-increments-on-subarrays-to-form-a-target-array)

## Code

```
int minNumberOperations(vector<int>& target) {  
    int totalOp = target[0];  
    for (int i = 1; i < target.size(); ++i) {  
        // can't reuse  
        if (target[i - 1] < target[i]) {  
            totalOp += target[i] - target[i - 1];  
        }  
    }  
    return totalOp;  
}
```