<https://leetcode.com/problems/longest-substring-without-repeating-characters/>

**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.

Constraints:

0 <= s.length <= 5 \* 104

s consists of English letters, digits, symbols and spaces.

**Method 1: (Brute Force)**

Find every substring and get their lengths

Time Complexity: O(n3) *[]*

Space Complexity: O(1)

**Method 2: (Sliding window with unordered\_map)**

Window points to substring and window is stored in an unordered\_set .

For every element x, check if set contains x.

If it contains x, slide left pointer of window till the duplicate while deleting them from set.

Simultaneously maintain a maximum substring length variable for storing the maxlength.

Time Complexity: O(n) *[]*

Space Complexity: O(n) *[set]*

int lengthOfLongestSubstring(string s) {

unordered\_set<char> st;

        int res = 0, l=0, r=0;

        while(r<s.size()){

            char x = s[r];

            if(st.count(x)!=0) {

                res = max(res, (int)st.size());

                //st.clear();

                while(s[l]!=x){

                    st.erase(s[l]);

                    l++;

                }

                st.erase(s[l++]);

            }

                st.insert(x);

                r++;

        }

        if(!st.empty()) res = max(res, (int)st.size());

        return res;

}

**Method 3: (Sliding window with visited vector)**

Window points to substring .

A visited vector will track if element was already visited, (replacement of unordered map; to check if window contains the element x)

If it contains x, slide left pointer of window till the duplicate.

Simultaneously maintain a maximum substring length variable for storing the maxlength.

Time Complexity: O(n) *[]*

Space Complexity: O(1) *[vector of constant size 256]*

int lengthOfLongestSubstring(string str) {

       // if string length is 0

        if (str.length() == 0)

            return 0;

        // if string length 1

        if (str.length() == 1)

            return 1;

        // if string length is more than 2

        int maxLength = 0;

        bool visited[256] = { false };

        // left and right pointer of sliding window

        int left = 0, right = 0;

        for (; right < str.length(); right++) {

            // if character is not visited then mark visited

            if (visited[str[right]] == false)

                visited[str[right]] = true;

            // if character is visited

            else {

                /\*    capture the unique string from [left ,

                right). Not including right since repeating

                character is at index right.

                \*/

                maxLength = max(maxLength, (right - left));

                /\*    Mark all characters until repeating

                    character as unvisited but not the repeating

                    character as it is in the new unique string.

                    However move window past the repeating

                    character.

                \*/

                while (left < right) {

                    if (str[left] != str[right])

                        visited[str[left]] = false;

                    else {

                        left++;

                        break;

                    }

                    left++;

                }

            }

        }

        // compare current left,right with previous result

        maxLength = max(maxLength, (right - left));

        return maxLength;

    }