<https://leetcode.com/problems/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 \* 10^4

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

**Attempt 1: 2022-09-09**

**Solution 1: Not fixed length Sliding Window (30min)**

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int len = s.length();

        Map<Character, Integer> freq = new HashMap<Character, Integer>();

        int maxLen = Integer.MIN\_VALUE;

        // 'i' means left end pointer, 'j' means right end pointer

        int i = 0;

        for(int j = 0; j < len; j++) {

            char c = s.charAt(j);

            freq.put(c, freq.getOrDefault(c, 0) + 1);

            int size = freq.size();

            // When map size less than substring length means repeating characters exist

            // To resolve repeating character requires shrink subtring by moving left end

            // pointer to right

            while(size < j - i + 1) {

                // Remove 'c1' and update its frequency on map

                char c1 = s.charAt(i);

                freq.put(c1, freq.get(c1) - 1);

                // If 'c1' frequency drop to 0 means not exist on current substring, remove

                // 'c1' from map key, update map size

                if(freq.get(c1) == 0) {

                    freq.remove(c1);

                    size--;

                }

                // Shrink left end pointer one position

                i++;

            }

            maxLen = Math.max(maxLen, j - i + 1);

        }

        return maxLen == Integer.MIN\_VALUE ? 0 : maxLen;

    }

}

Space Complexity: O(n)

Time Complexity: O(n)

<https://leetcode.com/problems/minimum-size-subarray-sum/discuss/59078/Accepted-clean-Java-O(n)-solution-(two-pointers)/924580>

For those trying to figure out how is it O(n):

Here we have defined 2 index i & j,

In case of O(n^2) for each outer loop, inner loop runs some n or m number of times to make it O(nm), that means, as soon as the outer loop finishes one iteration, inner loop resets itself.

In case of O(n2), as in this case, we are not resetting the inner inner variable i, it's just incrementing each time. It is like 2 loops one after another and both runs n number of time.

**Solution 2 (180min, too long for recollecting how to truncate duplicate characters by tracking left end pointer without removing actual map key character like Solution 1)**

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int len = s.length();

        // Map store different content -> {key, value} => {character, position} not {character, frequency}

        Map<Character, Integer> char\_index\_map = new HashMap<Character, Integer>();

        int maxLen = Integer.MIN\_VALUE;

        // 'i' means left end pointer, 'j' means right end pointer

        int i = 0;

        for(int j = 0; j < len; j++) {

            char c = s.charAt(j);

            // Once we've landed on a character we've seen before,

            // we want to move the left pointer of our window to the

            // index after the "LAST" occurrence of that character.

            // e.g "abba"

            // Initially left end pointer i = 0, right end pointer j keep

            // moving to right, when j = 2, find second 'b', since

            // we have seen 'b' when j = 1, 'b' is a duplicate character,

            // to remove duplication, we can move the left pointer of our

            // window as i = 0 to the index after the "LAST" occurrence

            // of 'b', which means update i from 0 to 2 ('LAST' occurrence

            // index = 1, plus 1 to skip it to next index, then 1 + 1 = 2),

            // when j = 3, find second 'a', since we have seen 'a' when

            // j = 0, 'a' is another duplicate character, we can move

            // the left pointer i from 2 back to 1 (0 + 1 = 1) ??

            // That's the issue, when current left end pointer i's index

            // is larger than what it suppose to be, that means current

            // duplicate character (e.g 'a' here) already truncated during

            // previous index update for other duplicate character

            // (e.g 'b' here), we just need to keep current position to

            // avoid left end pointer drop back.

            // In simple, we don't need to move back left end pointer i

            // index from 2 to 1 for duplicate 'a' because the first 'a'

            // already truncated when handling duplicate 'b' by updating

            // i from 0 to 2 (The index = 0 and index = 1 two characters

            // "ab" has been removed, remain substring as "ba", actually

            // no duplicate 'a' when we see the second 'a')

            if(char\_index\_map.containsKey(c)) {

                i = Math.max(i, char\_index\_map.get(c) + 1);

            }

            char\_index\_map.put(c, j);

            maxLen = Math.max(maxLen, j - i + 1);

        }

        return maxLen == Integer.MIN\_VALUE ? 0 : maxLen;

    }

}

Space Complexity: O(n)

Time Complexity: O(n)

<https://leetcode.com/problems/minimum-size-subarray-sum/discuss/59078/Accepted-clean-Java-O(n)-solution-(two-pointers)/924580>

For those trying to figure out how is it O(n):

Here we have defined 2 index i & j,

In case of O(n^2) for each outer loop, inner loop runs some n or m number of times to make it O(nm), that means, as soon as the outer loop finishes one iteration, inner loop resets itself.

In case of O(n2), as in this case, we are not resetting the inner inner variable i, it's just incrementing each time. It is like 2 loops one after another and both runs n number of time.