[Click on the logo to find **Problem Statement**]

# Intuition

The given code aims to find the maximum area between two vertical lines represented by an array of heights. It utilizes a two-pointer approach, starting with the widest possible container (using the outermost left and right indices). The algorithm iteratively calculates the area formed by the minimum height of the two pointers and the width between them, updating the maximum area encountered so far. The key insight is that moving the pointers towards each other preserves or potentially increases the height of the container, maximizing the potential area. By iteratively adjusting the pointers based on the comparison of the heights at each position, the algorithm efficiently identifies the maximum area and returns the result. This approach ensures a time complexity of O(n) as it processes each element in the input array only once, and it achieves this with constant space usage (O(1)) by maintaining a minimal set of variables.

# Approach

1. Initialization:
    - Initialize two pointers, `left` and `right`, at the beginning and end of the height array, respectively.
    - Initialize a variable `max_area` to store the maximum area encountered so far, starting with 0.
2. Main Loop:
    - Use a while loop that continues as long as the `left` pointer is less than or equal to the `right` pointer.
    - Within the loop, calculate the height of the current container using the minimum height between the heights at the `left` and `right` pointers. Calculate the width as the difference between the `right` and `left` pointers.
    - Update the `max_area` with the maximum of its current value and the area calculated in the previous step.
3. Move Pointers:
    - Compare the heights at the `left` and `right` pointers.
    - If the height at the `left` pointer is less than the height at the `right` pointer, increment the `left` pointer.
    - If the height at the `left` pointer is greater than or equal to the height at the `right` pointer, decrement the `right` pointer.

- This step ensures that the algorithm explores different container configurations by adjusting the pointers based on the comparison of heights.

4. Result:
   - After the loop completes, the `max_area` variable holds the maximum area between the vertical lines.

# Complexity

- Time complexity: O(n)
- Space complexity: O(1)

# Code

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        set_of_char = set()
        left = 0
        result = 0

        for right in range(len(s)):
            while s[right] in set_of_char:
                set_of_char.remove(s[left])
                left += 1
            set_of_char.add(s[right])
            result = max(result, right - left + 1)
        return result
```

**If you want to see more solutions to coding problems, you can visit:**


GitHub