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

Intuition

The provided code aims to find the length of the longest substring without repeating characters in a given string `s`. It utilizes a sliding window approach with two pointers, `left` and `right`, to dynamically adjust the substring under consideration. The set `set_of_char` keeps track of unique characters within the current window.

As the `right` pointer iterates through the string, the code checks for repeating characters. If a repetition is found, the `left` pointer is moved forward until the substring becomes unique again. At each step, the code updates the `result` variable with the maximum length of the unique substring encountered so far (`right - left + 1`).

Approach

1. **Initialization:**
    - Initialize an empty set `set_of_char` to keep track of unique characters within the current substring.
    - Set two pointers, `left` and `right`, both initially pointing to the start of the string (`left = 0` and `right = 0`).
    - Initialize a variable `result` to store the length of the longest substring without repeating characters.
2. **Sliding Window Iteration:**
    - Iterate over the characters of the string using the `right` pointer.
    - Check if the character at the current `right` position is already in `set_of_char`.
    - If it is, remove the character at the `left` position from `set_of_char` and increment `left` until the substring becomes unique again.
3. **Updating Result:**
    - After making the substring unique, add the current character at `right` to `set_of_char`.
    - Update the `result` with the maximum length of the substring so far (`right - left + 1`).
4. **Final Result:**

  - After iterating through the entire string, the `result` variable holds the length of the longest substring without repeating characters.

Complexity

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

Code

```python
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