educative

No-repeat Substring (hard)

We'll cover the following

Problem Statement

Given a string, find the **length of the longest substring**, which has **no repeating characters**.

Example 1:

```
Input: String="aabccbb"
Output: 3
Explanation: The longest substring without any repeating character
s is "abc".
```

Example 2:

```
Input: String="abbbb"
Output: 2
Explanation: The longest substring without any repeating character
s is "ab".
```

Example 3:

```
Input: String="abccde"
Output: 3
Explanation: Longest substrings without any repeating characters a
re "abc" & "cde".
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```python
def non_repeat_substring(str):
  # TODO: Write your code here
  within_windows = {}
  start,ans = 0,0
  for i in range(len(str)):
    #expand
    if str[i] not in within_windows:
      within_windows[str[i]] = 1
    #shrink
    else:
      within_windows[str[i]]+=1
      target = str[i]
      while within_windows[target]>1:
        within_windows[str[start]]-=1
        start+=1
    ans = max(ans,i-start+1)
  #return
  return ans
```

Show Results | Show Console

3 of 3 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓	non_repeat_substring(aabccbb)	3	3	Succeeded
✓	non_repeat_substring(abbbb)	2	2	Succeeded
✓	non_repeat_substring(abccde)	3	3	Succeeded

0.15s

Solution

This problem follows the **Sliding Window** pattern, and we can use a similar dynamic sliding window strategy as discussed in Longest Substring with K Distinct Characters (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5698217712812032/). We can use a **HashMap** to remember the last index of each character we have processed. Whenever we get a repeating character, we will shrink our sliding window to ensure that we always have distinct characters in the sliding window.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
def non_repeat_substring(str1):
  window_start = 0
  max_length = 0
  char_index_map = {}

  # try to extend the range [windowStart, windov
  for window_end in range(len(str1)):
    right_char = str1[window_end]
    # if the map already contains the 'right_cha
    # we have only one occurrence of 'right_char
    if right_char in char_index_map:
      # this is tricky; in the current window, w
      # and if 'window_start' is already ahead c
      window_start = max(window_start, char_inde
    # insert the 'right_char' into the map
    char_index_map[right_char] = window_end
    # remember the maximum length so far
    max_length = max(max_length, window_end - wi
  return max_length


def main():
  print("Length of the longest substring: " + st
  print("Length of the longest substring: " + st
  print("Length of the longest substring: " + st


main()
```

Time Complexity

The above algorithm's time complexity will be $O(N)$, where 'N' is the number of characters in the input string.

Space Complexity

The algorithm's space complexity will be $O(K)$, where K is the number of distinct characters in the input string. This also means $K <= N$, because in the worst case, the whole string might not have any repeating character, so the entire string will be added to the **HashMap**. Having said that, since we can expect a fixed set of characters in the input string (e.g., 26 for English letters), we can say that the algorithm runs in fixed space $O(1)$; in this case, we can use a fixed-size array instead of the **HashMap**.

← Back

Fruits into Baskets (medium)

Next →

Longest Substring with Same Letters ...

✓ Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/no-repeat-substring-hard__pattern-sliding-window__grokking-the-coding-interview-patterns-for-coding-questions)