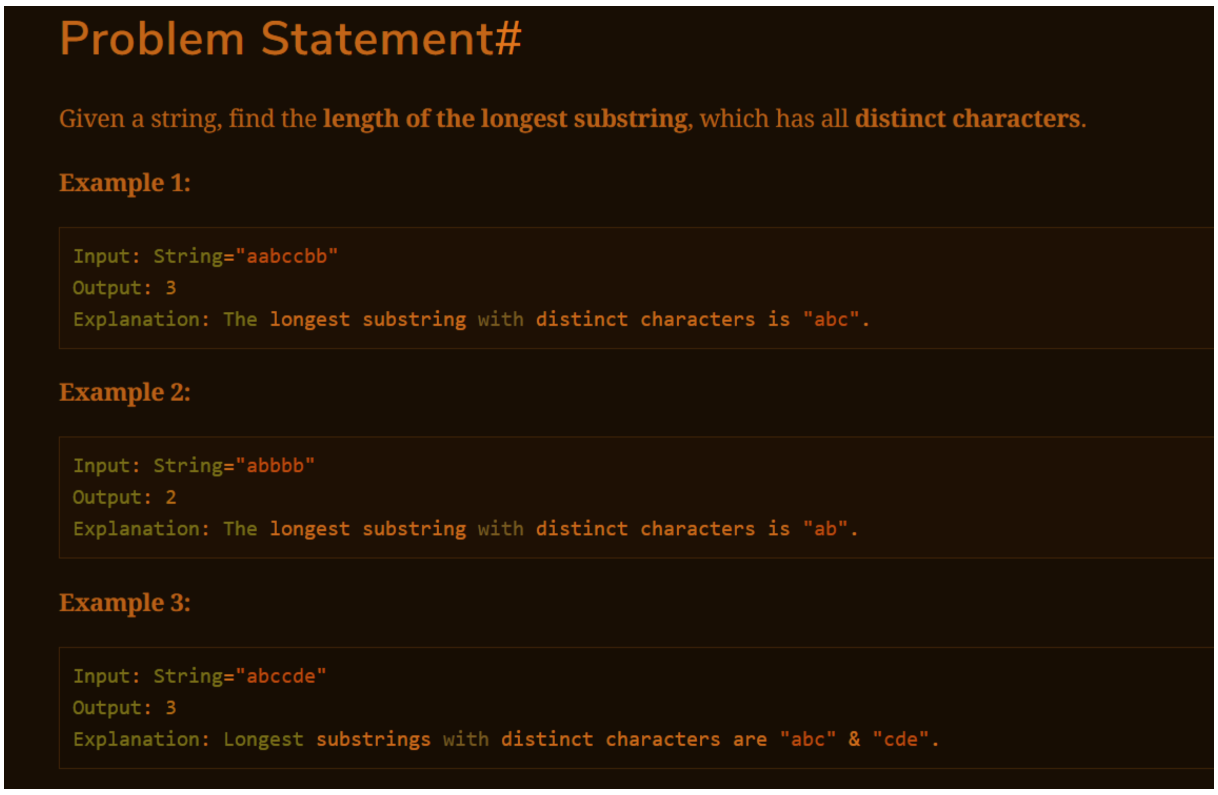
# Longest Substring with Distinct Characters

| | |
|---|---|
| SR Score | 1203 |
| Link | https://www.educative.io/courses/grokking-the-coding-interview/YMzBx1gE5EO |
| Last Reviewed | @April 8, 2022 |
| Time | 4 |
| Score | 3 |
| DS | arrays |
| Algo | sliding window |
| Stated | hard |
| Perceived | medium |
| List | TODO |
| Needs Review | ☑ |
| Repeat Offender | ☐ |
| Confident | ☐ |
| C_Date | 1 |
| C_Solution | 3 |
| C_Time | 400 |
| Frequency | |

▼ **Problem Statement**



## Problem Statement#

Given a string, find the **length of the longest substring**, which has all **distinct characters**.

**Example 1:**

```
Input: String="aabccbb"
Output: 3
Explanation: The longest substring with distinct characters is "abc".
```

**Example 2:**

```
Input: String="abbbb"
Output: 2
Explanation: The longest substring with distinct characters is "ab".
```

**Example 3:**

```
Input: String="abccde"
Output: 3
Explanation: Longest substrings with distinct characters are "abc" & "cde".
```

▼ **Intuition**

- recognize set trigger for "which has all distinct characters."
  - allows you to use set
  - now that we're using a set, we don't need to do $max(letterTracker.values())$, which is an addt'l $O(n)$ operation
- now that we're using a set, we can almost "prelook" ahead.. so if the current letter is already in the set, then we immediately get to work on shrinking the window and removing letters until that dupe is gone
  - once the dupe is gone, we can safely add the current letter to the set

- any time the curr letter is not in the set already, we can safely add it and check if the current window size is larger than the existing maximum size

▼ **Time & Space Considerations**

- Time: O(2n) → O(n)
    - for loop goes thru entire string 1 time → O(n)
    - while loop will process each character a max of 1 time → O(n)
    - $O(n+n) = O(2n) -- > O(n)$
- Space: O(n)
    - using a set, which can end up storing all the letters in the string if all characters are distinct → O(n)

▼ **Review Notes**

▼ [3/7/22]

- got solution with dict
- set trigger for "which has all **distinct characters.**"
    - all distinct in this case means there can only be a max of 1 of each letter, and sets are meant to only contain unique values
        - so we don't need a key for the letter and then separately tally its counts as the value; the fact that the letter exists in the set means it occurs once exactly, no more, no less

▼ [3/16/22]

- got the dict solution again
- alarm bells didn't go off about distinct

▼ [4/8/22]

- same as attempt 2.. stagnating here.. **remember** set trigger for "which has all distinct characters."

▼ **Tracking**

**Scores**

| Attempt # | Date | Time | Score |
|---|---|---|---|
| 3 | @April 8, 2022 | 4 | 4 |
| 2 | @March 16, 2022 | 3 | 3 |
| 1 | @March 7, 2022 | 2 | 3 |

▼ **Solutions**

```
# ---------------------------------------------------------------------------------
# attempt 3: got close to optimal but not quite
#time: O(2n) -> O(n)
#space: O(n)
def non_repeat_substring(str):
    # ideal solution
    letterTracker = set()
    longest = windowStart = 0
    for windowEnd in range(len(str)):
        letter = str[windowEnd]
        while letter in letterTracker:
            letterTracker.remove(str[windowStart])
            windowStart += 1
        letterTracker.add(letter)

        longest = max(longest, windowEnd - windowStart + 1)

    return longest
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # my attempt
    letterTracker = dict()
    longest = windowStart = 0
    for windowEnd in range(len(str)):
        letter = str[windowEnd]
        letterTracker[letter] = letterTracker.get(letter, 0) + 1

        while max(letterTracker.values()) > 1:
            leftLetter = str[windowStart]
            letterTracker[leftLetter] -= 1
            windowStart += 1

        longest = max(longest, windowEnd - windowStart + 1)

    return longest


def main():
  print("Length of the longest substring: " + str(non_repeat_substring("aabccbb")))
  print("Length of the longest substring: " + str(non_repeat_substring("abbbb")))
  print("Length of the longest substring: " + str(non_repeat_substring("abccde")))
```

```
main()
# ----------------------------------------------------------------------------------
# attempt 2: got the close to optimal but not quite solution again
def non_repeat_substring(str):
    tracker = dict()
    longestSubstr = windowStart = 0
    for windowEnd in range(len(str)):
        letter = str[windowEnd]
        tracker[letter] = tracker.get(letter, 0) + 1
        while max(tracker.values()) > 1:
            leftmostLetter = str[windowStart]
            tracker[leftmostLetter] -= 1
            if tracker[leftmostLetter] == 0:
                del tracker[leftmostLetter]
            windowStart += 1
        longestSubstr = max(longestSubstr, windowEnd - windowStart + 1)
    return longestSubstr

def main():
  print("Length of the longest substring: " + str(non_repeat_substring("aabccbb")))
  print("Length of the longest substring: " + str(non_repeat_substring("abbbb")))
  print("Length of the longest substring: " + str(non_repeat_substring("abccde")))

main()
# --------------------------------------------------------------------------------
# solve 1: 3/7 (got a solution :)), but not ideal with set
def non_repeat_substring(str):
    letterTracker = set()
    longestSubstring = windowStart = 0
    for windowEnd in range(len(str)):
        letter = str[windowEnd]
        while letter in letterTracker:
            letterTracker.remove(str[windowStart])
            windowStart += 1
        letterTracker.add(letter)
        longestSubstring = max(longestSubstring, len(letterTracker))

    return longestSubstring
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    # my solution
    letterTracker = dict()
    longestSubstring = windowStart = 0
    for windowEnd in range(len(str)):
        letter = str[windowEnd]
        letterTracker[letter] = letterTracker.get(letter, 0) + 1
        while max(letterTracker.values()) > 1:
            leftLetter = str[windowStart]
            letterTracker[leftLetter] -= 1
            if letterTracker[leftLetter] == 0:
                del letterTracker[leftLetter]
            windowStart += 1
        longestSubstring = max(longestSubstring, windowEnd - windowStart + 1)

    return longestSubstring

def main():
  print("Length of the longest substring: " + str(non_repeat_substring("aabccbb")))
  print("Length of the longest substring: " + str(non_repeat_substring("abbbb")))
  print("Length of the longest substring: " + str(non_repeat_substring("abccde")))

main()
```

## ▼ Resources

Longest Substring Without Repeating Characters - Leetcode 3 - Python

🎓 Get 10% off EducativeIO today ▸ https://www.educative.io/neetcode🚀 https://neetcode.io/ - A better way to prepare for Coding Interviews🟣 Get 10% off Alg...

https://www.youtube.com/watch?v=wiGpQwVHdE0

## ▼ GitHub

GCI/Pattern 1 - Sliding Window/Longest Substring with Distinct Characters at main · psdev30/GCI

Contribute to psdev30/GCI development by creating an account on GitHub.

https://github.com/psdev30/GCI/tree/main/Pattern%201%20-%20Sliding%20Window/Longest%20Substring%20with%20Distinct%20Characters