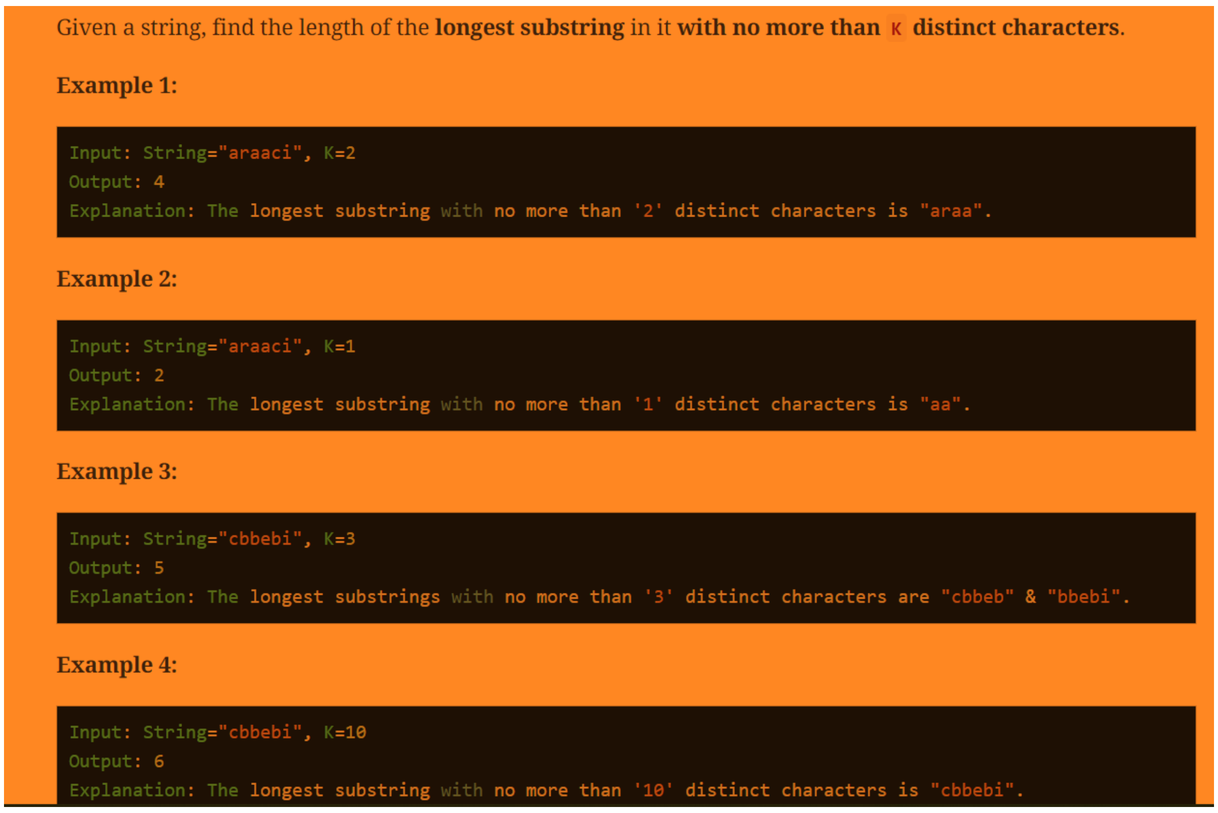
Longest Substring with maximum K Distinct Characters

SR Score	1002
Link	https://www.educative.io/courses/grokking-the-coding-interview/YQQwQMWLx80
Last Reviewed	@April 6, 2022
Time	5
Score	4
DS	arrays
Algo	sliding window
Stated	medium
Perceived	medium
List	REPEAT
Needs Review	✓
Repeat Offender	☐
Confident	☐
C_Date	1
C_Solution	2
C_Time	500
Frequency	

▼ Problem Statement



Given a string, find the length of the **longest substring** in it **with no more than `K` distinct characters**.

Example 1:

```
Input: String="araaci", K=2
Output: 4
Explanation: The longest substring with no more than '2' distinct characters is "araa".
```

Example 2:

```
Input: String="araaci", K=1
Output: 2
Explanation: The longest substring with no more than '1' distinct characters is "aa".
```

Example 3:

```
Input: String="cbbebi", K=3
Output: 5
Explanation: The longest substrings with no more than '3' distinct characters are "cbbeb" & "bbebi".
```

Example 4:

```
Input: String="cbbebi", K=10
Output: 6
Explanation: The longest substring with no more than '10' distinct characters is "cbbebi".
```

▼ Intuition

- ok so create two windowStart and windowEnd pointers to shrink & grow a sliding window as usual
- key diff is this time, we aren't just tracking the length of the longest window; the longest window depends on the counts of the number of times every character occurs
  - maintaining counts of an entity is perfectly suited for a dictionary, so we'll be using one!
- so the loop continues with the pausing condition being that the $len(letterTracker)$ (which is the same as the # of keys in the dict) goes over k.. at this we know we have more than k distinct characters and the window needs to shrink

- so we shrink the window until the # of keys in the dict —-> $len(letterTracker)$ —-> is back equal to k
- we keep repeating this process and for every element by which the window grows, without the # of distinct letters going above k, we can check if this new window size is larger than our current max and update accordingly!

▼ **Time & Space Considerations**

- Time: O(2n) > O(n)
    - for loop ensures iteration over all characters in the string → O(n)
    - with the $while$ loop, each letter is processed exactly once, so it’s like iterating thru the string again with the $windowStart$ pointer → O(n)

> 💡 looks like len(letterTracker) is actually O(1) bc the length (# of keys in this case) is automatically tracked as the dict shrinks & grows.. ***len(—built-in DS-) in python is constant time***

- Space: O(K + 1)
    - max number of keys $letterTracker$ can have is $k+1$ since as soon as we add the third distinct character, we shrink the window until there are only 2 distinct characters again

▼ **Review Notes**

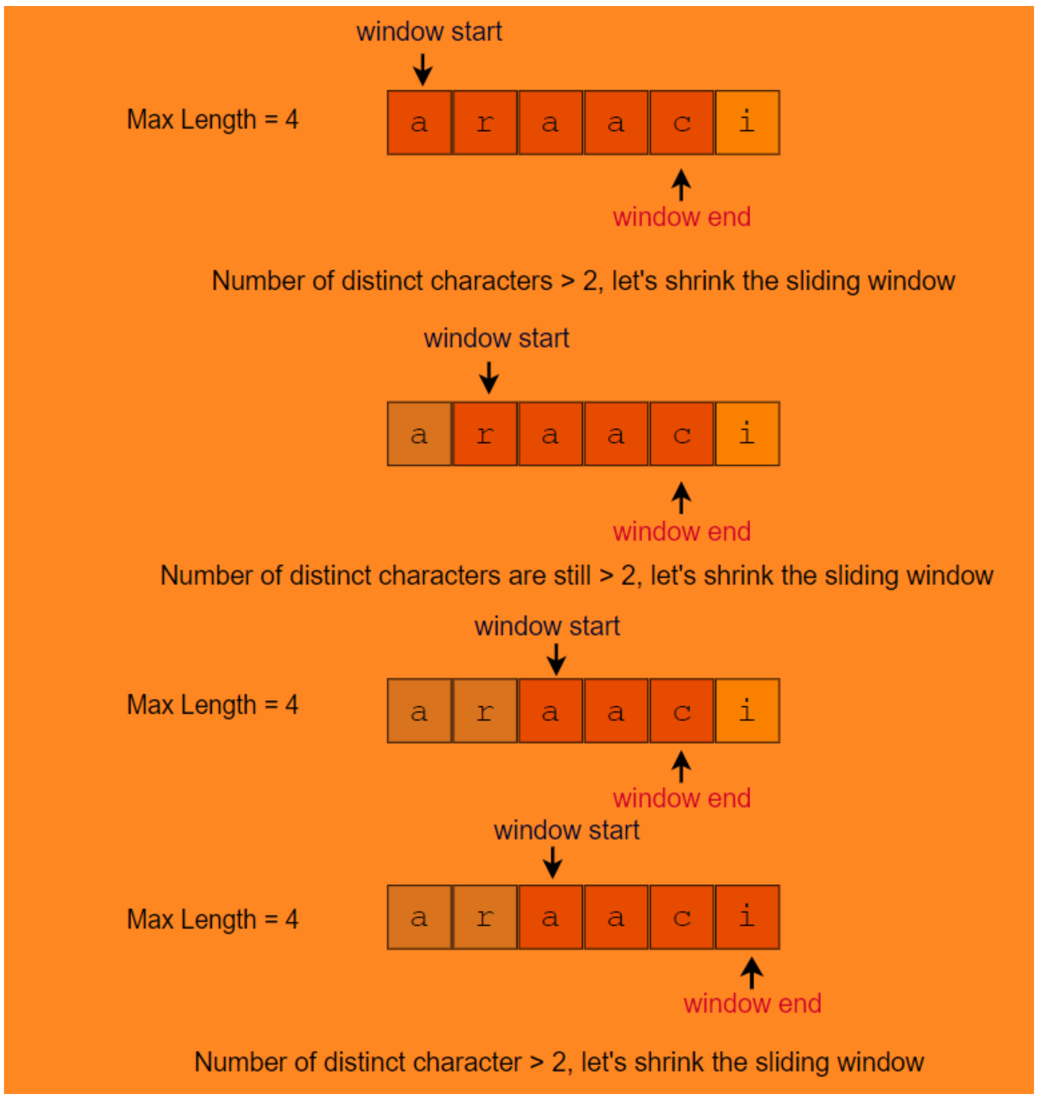
▼ [3/4/22]

- had no idea, looked at Solutions

▼ [4/6/22]

- got optimally very fast (sub 5 min)
- good to see that I knew immediately a dict was needed (since it was clear we needed to keep track of letter counts in the input string)

▼ **one minor but costly bug was using if instead of $while$; space complexity goes above $K+1$ with if statement, & more impt. flaw seen here:**


window start
Max Length = 4
a r a a c i
window end
Number of distinct characters > 2, let's shrink the sliding window
window start
a r a a c i
window end
Number of distinct characters are still > 2, let's shrink the sliding window
window start
Max Length = 4
a r a a c i
window end
window start
Max Length = 4
a r a a c i
window end
Number of distinct character > 2, let's shrink the sliding window

shrinking window by 1 character could still result in there being k + 1 distinct characters (raac still has 3 after removing the “a” @ index 0)

- if $maxLen > k$, this implies that at least one of the two distinct characters is repeating, so if the letter removed as part of the window shrinking repeats later in the substring, we haven’t actually shrunk the # of distinct characters yet (there are still $k + 1$ distinct characters in $letterTracker$); as such, we need a $while$ to keep shrinking the window & removing letters until there are actually only 2 distinct characters in the new substring

▼ **Tracking**

Scores

Attempt #	Date	Time	Score
2	@April 6, 2022	5	4
1	@March 4, 2022	1	1
Untitled			

▼ **Solutions**

```
# solve 2: 4/6/22 (one bug --> if instead of while)
#time: O(2n) -> O(n)
#space: O(K + 1)
# https://www.notion.so/Longest-Substring-with-maximum-K-Distinct-Characters-38076943c5234420a230a239c6d89d64
def longest_substring_with_k_distinct(string, k):
    letterTracker = dict()
    longestSubstr = windowStart = 0
    for windowEnd in range(len(string)):
        letter = string[windowEnd]
        letterTracker[letter] = letterTracker.get(letter, 0) + 1
        while len(letterTracker) > k:
            leftLetter = string[windowStart]
            letterTracker[leftLetter] -= 1
            if letterTracker[leftLetter] == 0:
                del letterTracker[leftLetter]
            windowStart += 1

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

    return longestSubstr

def main():
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("araaci", 2)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("araaci", 1)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("cbbebi", 3)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("cbbebi", 10)))

main()
# ----------------------------------------------------------------------------------
# solve 1: 3/4/22 (looked at solution)
#time: O(2n) -> O(n)
#space: O(K + 1)
# https://www.notion.so/Longest-Substring-with-maximum-K-Distinct-Characters-38076943c5234420a230a239c6d89d64
def longest_substring_with_k_distinct(string, k):
    letterTracker = dict()
    longestSubstr = windowStart = 0
    for windowEnd in range(len(string)):
        letter = string[windowEnd]
        letterTracker[letter] = letterTracker.get(letter, 0) + 1
        while len(letterTracker) > k:
            leftLetter = string[windowStart]
            letterTracker[leftLetter] -= 1
            if letterTracker[leftLetter] == 0:
                del letterTracker[leftLetter]
            windowStart += 1

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

    return longestSubstr

def main():
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("araaci", 2)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("araaci", 1)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("cbbebi", 3)))
  print("Length of the longest substring: " + str(longest_substring_with_k_distinct("cbbebi", 10)))

main()
```

▼ Resources

Longest Substring with maximum K Distinct Characters (medium) - Grokking the Coding Interview: Patterns for Co

https://www.educative.io/courses/grokking-the-coding-interview/YQQwQMWLx80

▼ GitHub

GCI/Pattern 1 - Sliding Window/Longest Substring with maximum K 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%20maximum%20K%20Distinct%20Characters