Input: array of int, int S
Output: int

Return 0 if none exists.

$\boxed{\text{Ex}}$ [2, 1, 5, 2, 3, 2], S = 7

$\Rightarrow$ 2 ([5, 2])

Naive Brute force

↳ Use double or more loops to add up the elem. in the list and have a counter. and total. If total exceeds S then increment the counter.

↳ $O(n^2)$ time, $O(1)$ space

+

Sliding Window Approach

↳ Similar to the brute force, we will add and check the total of subarray w/ input S and as we traverse the ~~list~~ array, we will subtract the initial elem of the subarray and add the subsequent elem in the array.

1/2

$\boxed{\text{Ex}}$ $[2, 1, 5, 2, 3, 2]$, $S = 7$

$\hookrightarrow$ $= 8$, don't need to continue

$= 6$

$= 8$

$= 7$

$= 5$

$= 7$

$= 5$

9min

```
def f(k, A):
    w_st, w_sum, max_sum = 0, 0, 0
    for w_end in range(len(A)):
        w_sum += A[w_end]
        if w_end >= k-1:
            max_sum = max(max_sum, w_sum)
            w_sum -= A[w_st]
            w_st += 1
return max_sum
```

```
import math

def f(k, A):
    w_st, w_sum = 0, 0

    min_len = math.inf

    for w_end in range(len(A)):
        w_sum += A[w_end]
        while w_sum >= k :
            min_len = min(min_len, w_end-w_st - 1)
            w_sum -= A[w_st]
            w_st += 1

    return min_len
```

Correction

```
def longest_substring_with_k_distinct(str1, k):
    w_start, max_length = 0, 0
    char_freq = {}
    for w_end in range(len(str1)):
        right_char = str1[w_end]
        if right_char not in char_freq:
            char_freq[right_char] = 0
        char_freq[right_Char] += 1
        while len(char_freq) > k:
            left_char = str1[w_start]
            char_freq[left_char] -= 1
            if char_freq[left_char] == 0:
                del char_freq[left_char]
            w_start += 1
        max_length = max(max_length, w_end-w_start-1)
    return max_length
```

if not len(char_freq) > k

$\boxed{k=2}$

a r a a c i

w_st w_end

Max_length = 0

char_freq = { }

w_st w_end

Max_length = 1

cf = { a : 1 }

w_st w_end

m_l = 2

c_f = { a : 1 , r : 1 }

w_st w_end

m_l = 3

c_f = { a : 2 , r : 1 }

w_st w_end

m_l = 4

c_f = { a : 3 , r : 1 }

len(c_f) = 2

c_f = { a : 3 , r = 1 , c : 1 }

len(c_f) = 3 > k

w_st += 1
= 1

left_char = "a"

c_f = {a : 2 , r : 1 , c : 1}

---

left_char = "r"

c_f = {a : 2 , ~~r : 0~~ , c : 1} → del

w_st += 1
= 2

c_f = {a : 2 , c : 1}

→ At this point len(char_freq) > k

so end of loop

0	1	2	3	4	5
a	r	a	a	c	i
		↑ w_st		↑ w_end	

a r a a c i

w_st w_end

ml = 4

c_f = {a:2, c:1, i:1}

len > k

---

w_st

left_char = "a"

c_f = {a:1, c:1, i:1}

w_st += 1

= 3

---

w_st

left_char = "a"

c_f = {c:1, i:1}

w_st = 4

~~left~~

4/4

Longest substring w/k distinct

Input: String

Output: int

Find length of longest substring w/ no more than k distinct characters.

Naive App.:

- ~~Take For every unique char, we need to add to the result string~~

- "a r a a c i", $k = 2$

  ↳ For every unique char, decrement k.

(★) Need to use hashmap

→ Take a, k--, count++

Take r, k--, count++

a count++

a count++

⇒ count = $\boxed{4}$,

```
d = {}
for char in string1
    if char not in d:
        d[char] = 1
        k -= 1
        count += 1
    if char in d:
        count += 1
```

→ while $k \geq 0$:

↳ Naive $O(n)$ time, space

→ A better approach is to use sliding windows where

↳ start w/ substring w/ length k

"ar"

↳ If next elem is in substring add to str.

↳ Else, move to next substring of length k. → "ar" → "rb"

2/3

$\boxed{\text{Ex}}$ "cb$\underbrace{\text{atta}}_{\Rightarrow 4}$", $k = 2$

cb a tt a

→ ✗

→ ✗

→ ✓

→ ✓

= atta , return 4

→ $O(n)$, $O(1)$

↳ $O\left(\frac{n}{k}\right) \approx O(n)$

13 min , 44 min ~~33 min~~ Correction

3/3

Input : Array of Char (upper)
Output : int

Max number of fruit of unique type.

$\boxed{\text{Ex}}$ [A, B, C, A, C] $\Longrightarrow$ 3

Need to use sliding windows and <u>hashmap</u> to keep track of fruits.

↳ If we have more than two different fruit type (i.e. we have [A, B, C]), move the sliding window.

→ [A, B] → [B, C] → [C, A] → [C, A, C]

A B C A C

we will have to keep track of char from the start and end of the window.

1/1

7min design, 12min Total