Remove Duplicate

7/19

My Sol.

```
def remove_dup(A):
    l, r, c = 0, 0, 1
    while r < len(A):
        if A[l] ≠ A[r]:
            c += 1
            l = r
            r += 1
        else:
            r += 1
return c
```

Did not know why this was 1

originally had as left

Had c+=1 in the wrong place

Remove Duplicate

7/19

Input: sorted array

Output: length of array w/ distinct elem.

A naive approach is to check for every elem in the list if the subseq. elem is different and use a counter.

$\boxed{\text{Ex 1}}$

2 3 3 3 6 9 9 $\quad c = 0$

$c = 1$

$c = 2$

$c = 3$

l and r ptrs start at 0

if different
left = right
right ++

else:
right ++

use while loop

5min

9min code

Input : Sorted array , target int
Output : Array of indices

Naive → $O(n^2)$, $O(1)$

Pointers (Two)

Have one ptr from left and another from right
While the left ptr is less than the right ptr
(index pos.)
decrement and increment the ptrs until
the elements add up to the target.

```
def f(A, t):
    l, r = 0, len(A) - 1

    while l <= r:
        if t - A[l] < A[r]
            r -= 1
        if t - A[l] == A[r]:
            return [l, r]
        else
            l += 1
    return -1
```

7 min

7/19

I/O : Arr, int / int

This problem is exactly the same as previous but with array A where $A[i] \in [0,1] \ \forall i > 0$.

- We ~~might~~ not even need a hashmap.

$\Rightarrow O(n), O(1)$

$\approx O(2)$?

4min design

12min code total

```
def f(A, k):
    w_st, m_l, m_r = 0, 0, 0
    f_m = {0:0, 1:0}

    for w_end in range(len(A)):

        right = A[w_end]
        f_m[right] += 1
        m_r = max(m_r, f_m[right])

        if (w_end - w_st + 1 - m_r) > k:
            left = A[w_st]
            f_m[left] -= 1
            w_st += 1

        m_l = max(m_l, w_end - w_st + 1)

return m_l
```

Longest substring w/ same letters after replacement

4/19

→ Longest substring w/ distinct char.

① Iterate through string, add one letter (in the right) at a time in the window

② Keep track of max repeating letter in <u>any</u> window.

③ At any window, we have a window w/ one letter repeating max repeat count times, the we replace the remaining letters.

↳ If the count of remaining letters is k or less, replace all

↳ If not, shrink the window until we can replace k letters.

0/3

```
def length_of_longest_substring(s, k):
    w_st, max_length, max_repeat_letter_count = 0, 0, 0
    freq_map = {}

    for w_end in range(len(s)):
        right_ch = s[w_end]
        if right_ch not in freq_map:
            freq_map[right_ch] = 0
        freq_map[right_ch] += 1

        max_repeat_letter_count = (
            maxrepeat_letter_count,
            freq_map[right_ch] )
        if (w_end - w_st + 1 - m_r_l_c) > k:

            left_ch = s[w_st]
            freq_map[left_ch] -= 1
            w_st += 1
        max_length = max(max_length, w_end - w_st + 1)

    return max_length
```

Ex1

a a b c c b b , $k=2$

we ws

ws we

f_m = {a : 1}

m_r = 1

m_l = 2

[redacted]

[redacted]

f_m = {a : 2}

m_r = 2

m_l = 2

a a b

ws we

f_m = {a : 2, b : 1}

m_r = 2

m_l = 3 = w_end − w_st + 1

a a b c

ws we

f_m = {a : 2, b : 1, c : 1}

m_r = 2

m_l = 4

a a b c c

w_s w_e

f_m = {a:2, b:1, c:2}

m_r = 2

left_ch = a

f_m = {a:1, b:1, c:2}

w_s += 1 ⟺ w_s = 1

m_l = 4

a a b c c b b

w_s w_e

f_m = {a:1, b:2, c:2}

m_r = 2

left_ch = a

f_m = {a:0, b:2, c:2}

w_s += 1 ⟺ w_s = 2

m_l = 4

0 1 2 3 4 5 6

a a b c c b b

w_s w_e

f_m = {a:0, b:3, c:2}

m_r = 3

m_l = 5 = w_end + w_st + 1
= 6 − 2 + 1

⟹ $\boxed{5}$

3/3

Longest substring with same letters after replacement

7/19

$\boxed{\text{I/O : str, int / int}}$

$\boxed{\text{Ex1}}$ s = "aabccbb", k = 2

$\Rightarrow$ aa bbbbb $\Rightarrow \boxed{5}$

$\boxed{\text{Ex2}}$ s = "abccde", k = 1

$\Rightarrow$ ccc $\Rightarrow \boxed{3}$

We need to consider the cases when the repeating characters are in between or and the start or end of the ~~char~~ substring

↳ or we can use sliding window and keep count of the max length

In the string, we don't care what the other k next characters are unless they are the repeating characters.

↳ To check the char. on the right, we need to use hash map.

---

a a b c c b b , k = 2

w st w end

k -= 1

a a a c c b b , k = 1

w st w end

right ch

w end

a a a a c b b k = 2

max l = 4

w end

a a b c c b b

st end

= 5

There needs to be a while loop ?

~~while~~

Basically the hashmap must have only one key

We can iterate the w_end k amount of time and continue if the character is in the hashmap.

```
def f(s, k):
    w_st, max_length = 0, 0
    h = {}
    for w_end in range(len(s)):
        r_ch = s[w_end]
        if r_ch not in h:
            if k ≠ 0:
                w_end += 1
                k -= 1
                max_length += 1
        ~~h[r_ch] = w_end~~
        ~~while len(h) > 1:~~
        max_length = max(max_length, w_end - w_st + 1)
```

27 min attempt

4/4

```
def non_repeat_substring(s):
    w_st, max_length = 0, 0
    ch_index_map = {}
    for w_end in range(len(s)):
        right_ch = s[w_end]
        if right_ch in ch_index_map:
            w_st = max(w_st, ch_index_map[right_ch]+1)
        ch_index_map[right_ch] = w_end
        max_length = max(max_length, w_end - w_st + 1)
    return max_length
```

Shrink until there is only one right_ch from the beginning of the substring.

Longest substring w/ distinct characters. 7/19

Input: String, Output: int

Ex1 "aabcbb" ⟹ 3

Ex2 "abcde" ⟹ 3

- Hashmap, sliding windows

↳ Most likely O(n) runtime and O(longest substring) ≈ O(k)

↳ Since we have to take count of unique character in hashmap.

4 min design

15 min Total code

↳ correct ✓

```
def f( s : Str ) → int :
    w_st, max_length = 0 , 0
    char_freq = {}
    for w_end in range(len(s)):

        right_ch = s[w_end]
        if right_ch not in char_freq:
            char_freq[right_ch] = 0
        char_freq[r_ch] += 1
        while char_freq[r_ch] > 1 :
            left_ch = s[w_st]
            char_freq[left_ch] -= 1
            w_st += 1
        max_length = max(max_length,
                         w_end - w_st + 1 )

return max_length
```

The approach I used.

"a b c c d e"

w_st w_end

r_ch = "a"

c_f = {a : 1}

max_length = 1 = (0 - 0 + 1)

w_st w_end

r_ch = "b"

c_f = {a : 1 , b : 1}

max_l = 2

"a a b c c b b"

w_st w_end

r_ch = a

c_f = {a : 1}

m l = 1

---

w_st w_end

w_st w_end

right_ch = a

c_f = {a : 2}

l_ch = a

c_f = {a : 1}

w_st++ = 1

m l = ~~1~~ - 1 + 1 = 1