fast get two sum at a time.

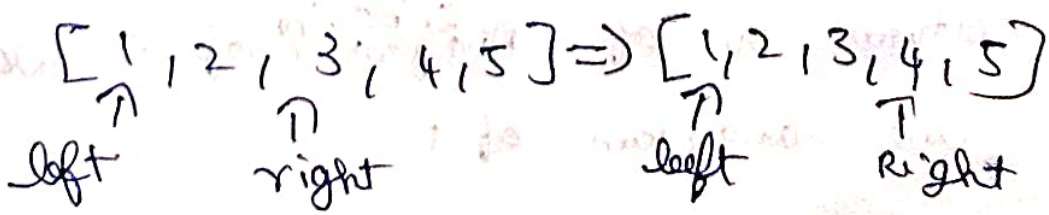
4) if fast meets slow then there is a loop so return as unhappy number.

5) if fast == 1 then return as happy number

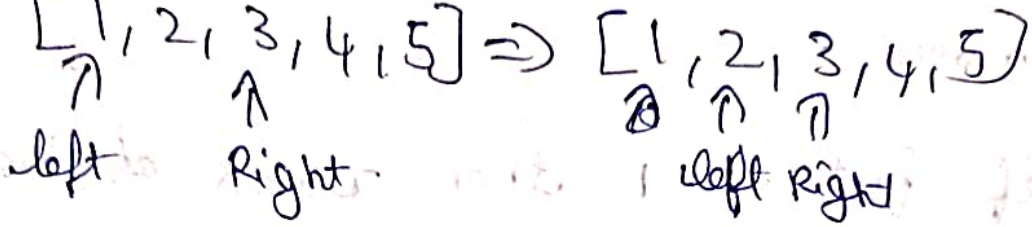
Sliding window

Sliding window is technique which is Subset of two pointers. window is a sub component in a data structure where slides in array to search a particular sub component (Sub array or Substring)

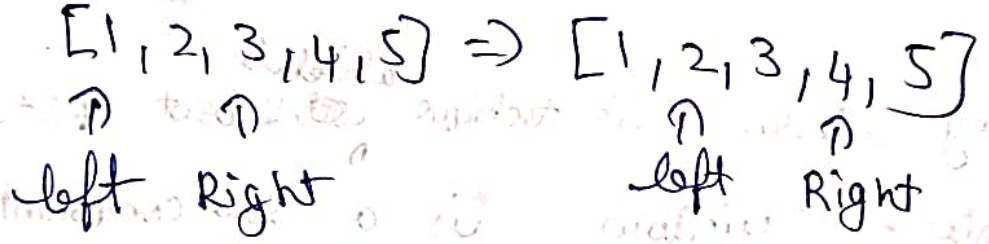
In Sliding window if we need to expand the window we have to move right pointer



if we need Shrink then move the left Pointer



if we need to slide the window move both Pointer



Types of sliding window

1) Fixed window
2) Variable window

Substring Anagram

Given String S & t we need return no. of Anagram Substring of S that are anagram of t

we are going to solve this problem by sliding window

Step by Step approach

1) here we need to find substring which is anagram of other string say t.

2) Then we can say substring should be length of t so we can solve this problem by fixed sliding window

3) we need to see the substring same frequencies of characters. To check we are going to initialize 26 char size array

4) each index in the array denotes a a char.

5) first iterate over the string t and populate the expected freq

6) then iterate over length t in given string & populate over

window freq

(7) check both expected freq & window freq are equal if yes then increment count

(8) move both pointer forward

s = "cbaebabacd" t = "abc"

expected freq

[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

window freq

[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

both array are equal so

count = 1

move left and Right one step forward

corresponding count need to be reduced in window freq

we need to decrement count at Index 2 which char C
and increment at index 4 which is char (E)

[1, 1, 0, 0, 1, ...] => window_freq

here window freq & expected freq are not equal

move forward both pointer
increment at index 1 which is char b
decrement at index 1 which is char b

[1, 1, 0, 0, 1, ...] => window_freq

here window freq & expected are not equal

move forward both pointer
increment at index 0 which is char a
increment at index 0 which is char a
decrement

repeat the same process till end of array

Longest Substring with unique characters

We need to find the longest substring with unique character

- we are solving this problem by Sliding window

Step by step approach

1) we need find a longest substring which means we dont know the size of window i.e dynamic Sliding window

2) here we need unique substring. To find unique character only substring we need to use hashset.

3) take two two pointer left and right starting at both index zero

4) declare hashset to check the substring is unique

5) check right pointer index value is present in hashset. if not move forward.

6) initially hashset is empty so index zero char is added to hashset

7) Then if char at right index is already found hashset move the left pointer to shrink the window. to make substring valid

$S = "abcbaba"$

hashset = {}

left = 0 Right = 0

initially hashset is empty we push the first char to hashset

Step = 1 hashset = {a}

left = 0 Right = 1

Step = 2 hashset = {a, b}

left = 0 Right = 2

Step3: {a, b, c}

left = 0 Right = 3

Step4: Since is Repeating again we need to shrink the window

{a b, c}

left = 1 Right = 3

Step 5; Since window is not valid shrink

{c}

left = 2 Right = 3

step6: hashset = {c, b}

left = 2 Right = 4

Step 7: hashset = {c, b, a}

left = 2 Right = 5

Step 8: again duplicate char is found so Shrink the window

hashset = {b, a}

left = 3 Right = 5

Step 4: still duplicate exist in window

So increase the window size by 1

hashset = {a}

left = 4 Right = 5

Repeat the same process till end of string

time complexity : O(n)

we can optimize the approch by creating dictionary

Store key value pair where key is char in string and its index in the string

if we encounter the duplicate which we have already

Longest substring after replacement

We are given with String and integer replacement k where we need to return longest length where replacing k character resulting uniform char

We can solve this problem by sliding window

Step by step approach

1) Here we need to find longest substring then we need to used dynamic sliding window

2) we need to find number of char replace to get uniform substring

3) for a window if you have highest freq char we can find no of char to replace by Substracting count of highest freq char from length of Substring

num of char to replace = len(string) − highest freq

4) if no of char replace is less than K than window is valid else window is not valid we have to shrink

5) if window is valid then we can expand the window

"aa b c d c c a" K=2

hashmap = { }. left = 0 , Right = 0

step 1 hashmap = {a:1} left = 0 Right = 0

no of char To replace = len(string) − high freq
= 1 − 1 = 0

Step 2 hashmap = {a:2} left = 0 Right = 1

No of char to Replace = 2 − 2 = 0

Step 3 hash map = {a:2 , b:1}.

left = 0 Right = 2

no of char to replace = 3 − 2 = 1 ≤ K

Step 4 : { a:2 b:1 c:1 }

left = 0 Right = 3

no of char to replace = 4 - 2 = 2 ≤ k

Step 5 : { a:2 b:1 c:1 d:1 }

left = 0 Right = 4

no of char to replace = 5 - 2 = 3 > k

So window is not valid
Shrink window

Step 6 : { a:1 b:1 c:1 d:1 }

left = 1 Right = 4

no of char = 4 - 1 = 3

Step { a:0 b:1 c:1 d:1 }

left = 2 Right = 4

no of char to replace = 3 - 1

= 2

Step 8 { a : 0 b : 1 c : 2 d : 1 }

left = 2 Right = 5

no of char to replace = 4 - 2 = 2

Step 9 { a : 0 b : 1 c : 3 d : 1 }

left = 2 Right = 6

no of char to replace = 5 - 3 = 2

step 10 { a : 0 1 b :

Repeat the same process till end

Time complexity : O(n)