String :-

↳ sequence of chars

↳ small set => 128 chars. (97 for 'a' to 'z', 65 for 'A' to 'Z')

↳ contiguous integer vals for 'a' to 'z' & 'A' to 'Z'

in both Ascii and UTF-16 encoding.

Ascii ~ older format.

↳ ord(char)

~ ordinal val.

↳ numerical representation of a char. in unicode / Ascii.

① Frequency of a char in a string.

↳ As it's eng we can go with ascii

↳ 0 - 127 chars

↳ check if the ord(char) is blw [0 to 127]

if yes increment it by 1

↳ for i in char freq:

if char freq > 0:

print chr(i) and charfreq[i] ~ times occured

Char

(2) is Subsequense of given string.

str1 => 'abc'

str2 => 'ahgbdc'

↳ get the minimum length of str1 & other of str2.

i=0 , j=0

abc

ahbgdc.

→ if both are same increment i

→ whatever string increment j

i => 1 2 3

1 2 3 4 5 6

if i == len(str1)

return True.

else

return False

→ Check for Anagram

↳ check if they are permutation of each other or not.

✓ s_1 = listen
s_2 = silent
} order doesn't matters

✗ s_1 = aab
s_2 = bab
} Not anagram because a appeared 2 times in first one while a appeared once in 2nd str.

① Sort both string & check if both are equal or not — $O(n \log n, n)$ - merge sort

② count Frequencies of each char and Check if they are equal

(11)

O (2n, 2n)

↳ 2 count Arrays to store freq of occurance of char.

↳ 1 loop to add them to count array.

↳ 2 loop to check if they are equal.

(3). (2n, n) / 2

Count = [0] * 128.

for i → 0 to len(str1):

count[ord(str1[i])] += 1

count[ord(str2[i])] -= 1

↳ check if every item in count zero.

→ Binary string

↳ Count num of sub strings that start & end with 1.

Str ⇒ 0 0 1 0 0 1 0 1

↳ if str[j] == 1 :

count += 1

Approach -2 :

S → 1 ⇒ 0

S → 11 ⇒ 1

S → 111 ⇒ 11, 11, 111 ⇒ 3.

S ⇒ 10101 ⇒ 101, 10101, 101 ⇒ 3.

so both give 3.

So based on num of one's we can predict answer.

S = 1111 ⇒ 1111, 11, 111, 111, 11, 11 ⇒ 6

S = 10111 ⇒ 101, 1011, 10111, 111, 11 11 ⇒ 6.

Approach 3

(Count of 1) * (Count of 1 - 1) / 2.

⇒ (3 * 2)/2 = 3

⇒ (4 * 3)/2 = 6

Approach 2 Continued.

If we carefully observe the strings we can get to know that the

For each one there are exactly ones count - 1 valid subsets - that gonna start & end with 1.

00 10011

↳ 1001, 10011

11

if we come from end.

if we encounter a one:

then increment one's count.

and increment count by one's count - 1

(1)

one's count += 1

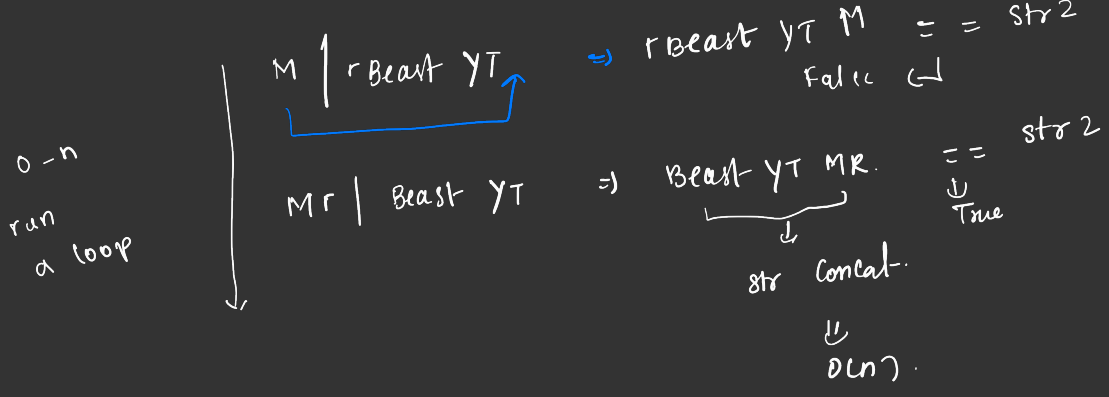
count += one's count - 1

We can also go from start but from end makes sense..

- Check if strings are rotated of each other or not.

i/p → Mr Beast YT ← S_1 } rotated 2 times.
Beast YT Mr ← S_2

o/p → True.



for ; → 0 - n : → O(n) } => $O(n^2)$.

O(n)
Concat
O(n) space.

° Case Specific Sorting of strings ..

lowercase , uppercase = [ ] , [ ] ⇒ 2 arrays.

add lower case & upper case letter accordingly.

res = ''

Sort both arrays

→ lower_index , upper_index = 0,0

⇒ for i in string:

if lower case :

res += lower case [lower_index]

lower_idx += 1

else :

res += uppercase [upper_idx]

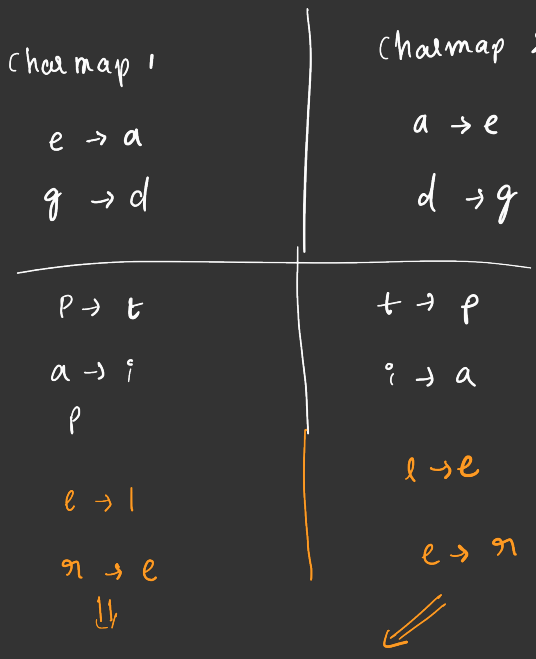
upper_idx += 1.

8. is isomorphic :

→ egg, add

→ paper, title.

2 charmaps ..



i=0

As p is already available in Charmap 1 we need to check if the value of p is t or not in str 2 //.

if item present in Charmap 1 we need to check if it exists in Charmap 2.

if not then add them to Charmap 1 & 2.

if yes return false.

→ left most.

① 2 loops → $O(n^2)$

```
for i 0 → n:
    j: i+1 → n:
        if _ _:
            retur i
```

geeks for geeks

i=0, j=1 → n.

```
if str[i] == str[j]:
    return i
return -1.
```

② $O(2n, n)$

↳ for i 0 → n:

char Count[ord(str[i])] += 1.

for i 0 → n:

if charCount[ord(str[i])] > 1:

retur i.

return −1.

③ Efficient Approach:

Start from End. so that we get the min index.

for i → n−1 → −1 : Exclusive.

if visited[ord(str[i])]:

res = i

else:

visited[ord(str[i])] = True

return res.

- reverse words.

{ i/p ⇒ i love programming very much
o/p ⇒ much very programming love i

Hi geeks Hello ⇒ Hello geeks Hi

iH skeeg olleH

↳ reverse this whole string

{ Hello geeks Hi }

How to reverse these parts ??

→ reverse the part of string till u encounter a ' ' space.

→ start again from next space
reverse till you encounter space.

As we wont be having space at last that we need to reverse.

reverse (start, end, str)

reverse (0, strlen(str) - 1, str)