Q Infinite stream of characters,
After a new character comes, check if the current stream of characters forms a palindromic string or not.

Google

can't store string

a	✓
ab	X
abc	X
abcb	X
abcba	✓
abcbak...	X

S = reverse(S) ⇒ palindrome Hash

S

FH BH

abcba

FH = a*p^0 + b*p^1 + c*p^2 + b*p^3 + a*p^4

BH = a*p^0 + b*p^1 + c*p^2 + b*p^3 + a*p^4

palindrome ⇒ FH = BH

		FH	BH
i=0	a	a*p^0	a*p^0
i=1	ab	a*p^0 + b*p^1	a*p^1 + b*p^0
i=2	abc	a*p^0 + b*p^1 + c*p^2	a*p^2 + b*p + c*p^0
i=3 ↓	abcb	a*p^0 + b*p^1 + c*p^2 + b*p^3	a*p^3 + b*p^2 + c*p + b

$$FH_{new} = FH_{old} + new\_char * p^{i}$$

$$BH_{new} = (BH_{old} * p) + new\_char$$

Q) given a string s, check if you can rearrange the string s such that there is no BORING substring in s.

Boring substring → Size = 2
→ both characters are adjacent to each other in English alphabet

s = a b c d

a c b d

c a d b

b d a c

a b ba boring

c d dc

aa ?

B.F :- generate all permutations and check !

$n! * n$

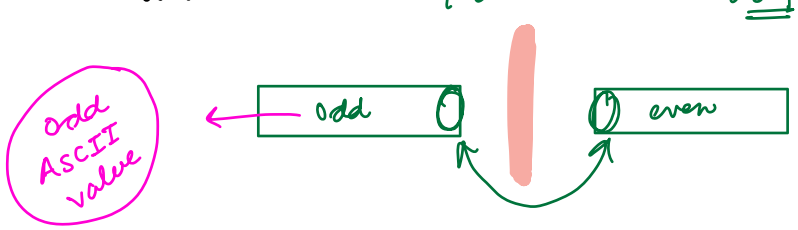
s

odd characters : Yes

even chars : Yes

a - 97 c = 99. --

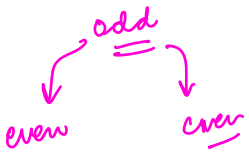
b d f ..

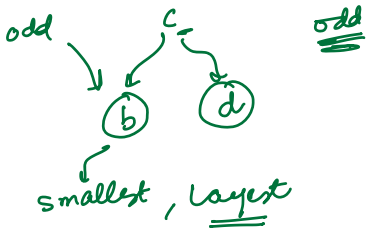


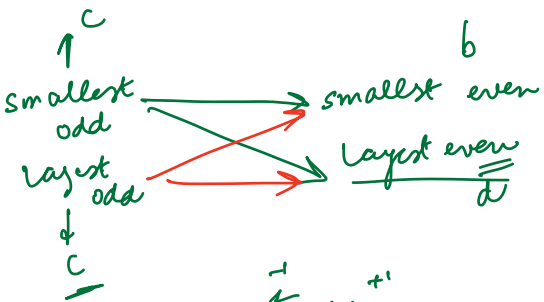
acabdac

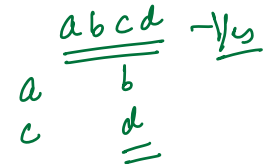
aacca — ~~bd~~ db

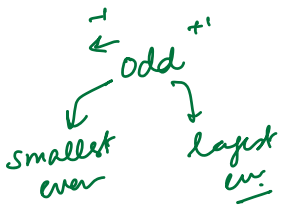
B * B = 0

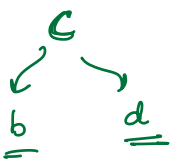

odd
even
even


odd
c
odd
b
d
smallest , layest


c
smallest
odd
layest
odd
c
b
smallest even
layest even
d


a b c d
a
b
c
d


odd
smallest
even
layest
even


c
b
d

Q Find length of longest substring which contains all unique / distinct characters.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a	e	b	c	a	b	g	e	b	@	#	g	b	k	d	b	#

B.F:- consider all possible substrings

BSG

(0,0) ✓ (1,1) X
(0,1) ✓ (1,2) X
(0,2) ✓ :
(0,3) ✓ (1,4)
(0,4) X

l=0, r=0 set

if(current element (arr[r]) is already present in set)
{
ans = max(ans, r-l);
while(arr[l] != curr_el)
{
l++; // remove your arr[l] from set
}
l++;
}
else{
// insert in set
r++;
}

T.C: O(n)
S.C: O(n)

optimi:-
store the prev occurence & then directly jump

optimisation with HM:

1) detect duplicate with right condition
   ↳ when your prev index is present inside the window
2) update your map
3) directly jump to prevIndex+1

Q. count all permutations of A in B as a substring.

A : a b c
B : a b c b a c a b c

ans = 5

permutation :- rearrangement

abc
bac
bca
cba

3!

A : a c a
B : a c a a

ans = 2

B.F :- generate all permutations $m! \times n$ → length of A

↓

same set of elements are just rearranged

⇓

just focus on freq of elements

A : a b c
B : a b c b a c a b c

a	b	c
1	1	1

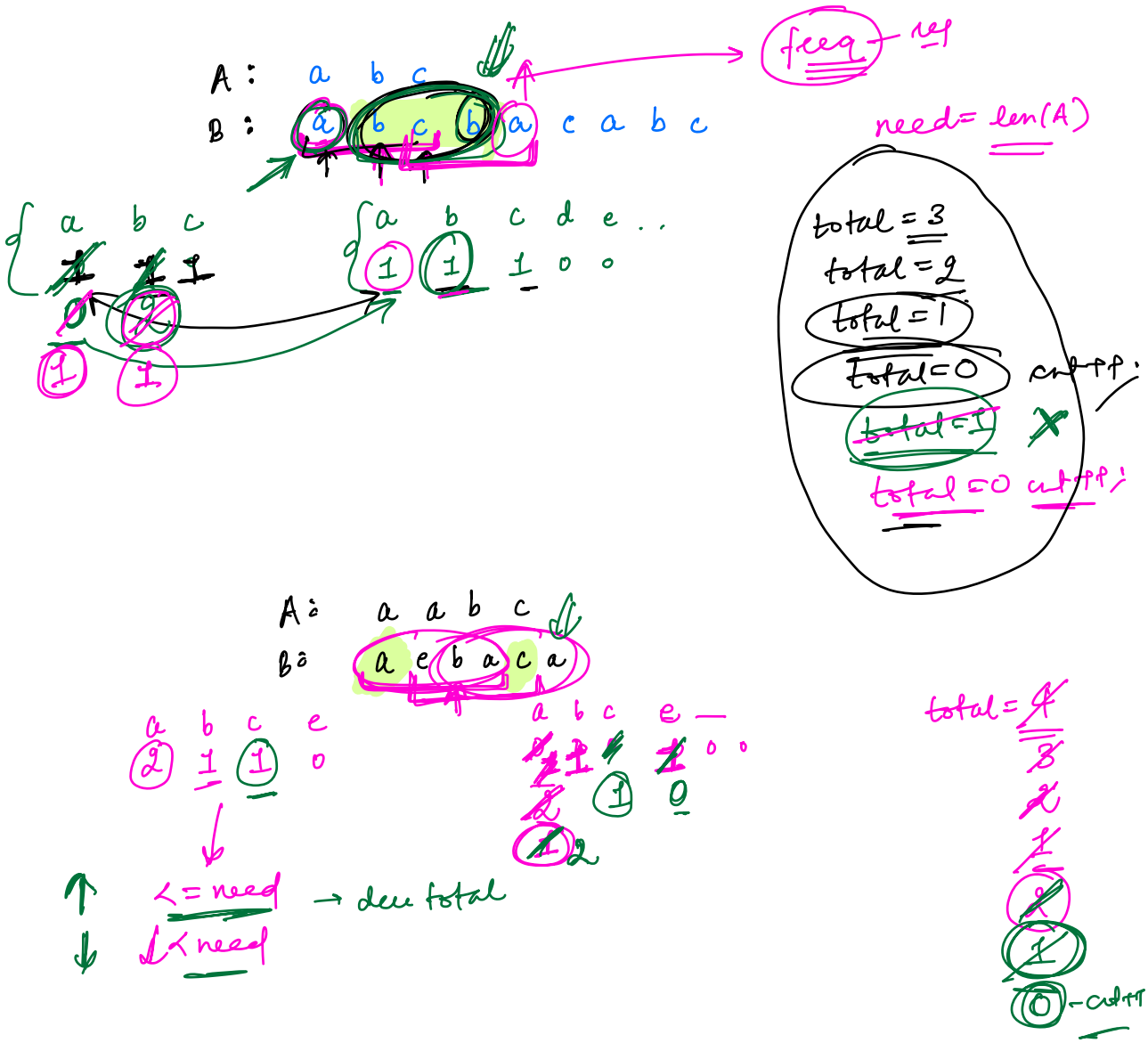
a	b	c
1	1	1

1) calculate freq array / HM for string A
2) build freq array for first window of size = length of A

traverse the freq & compare
↳ count++;

26 iteration

T.C :- $O(n*26) = O(n)$



element comes in → increase its freq

↓

<= needed freq ⇒ total--;

> freq ⇒ extra ✗

element goes out → decrease its freq

< needed freq total++;