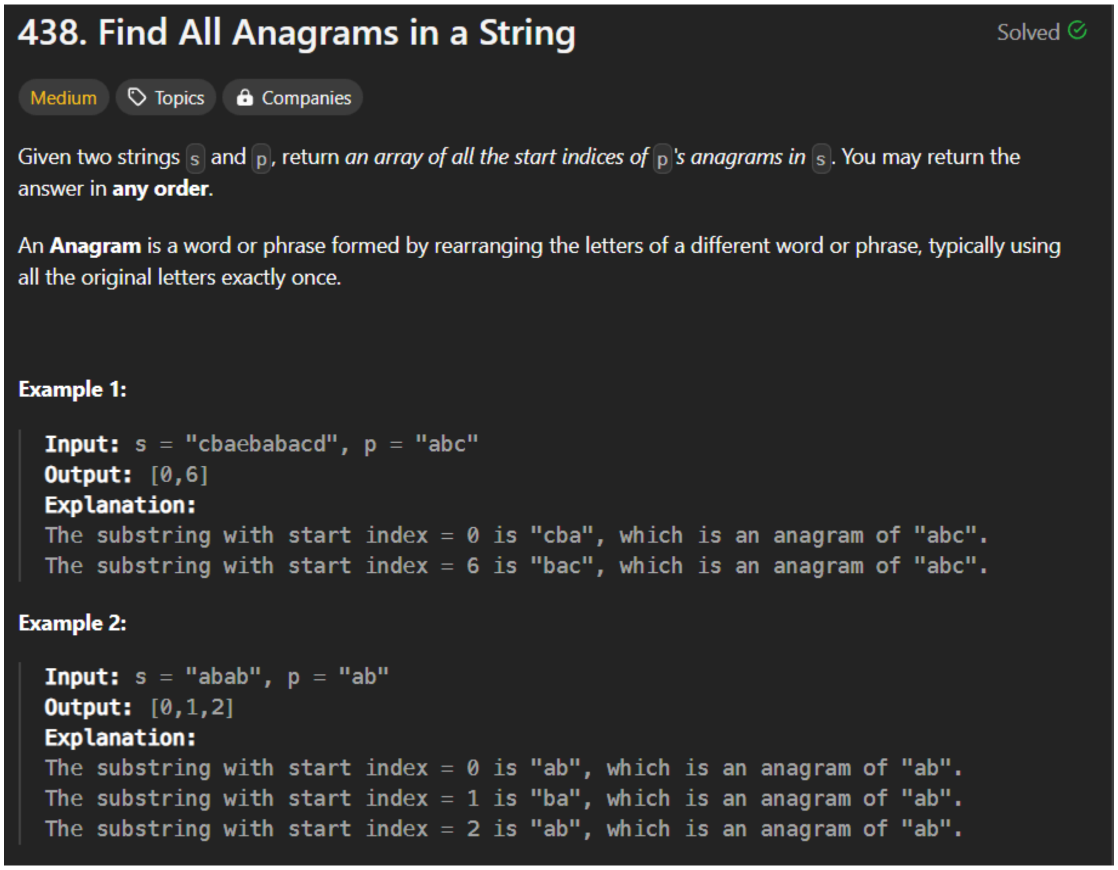
2: Leetcode 438: Find All Anagrams in a String

⇒ https://leetcode.com/problems/find-all-anagrams-in-a-string/description/



438. Find All Anagrams in a String

Solved ✓

Medium · Topics · Companies

Given two strings `s` and `p`, return *an array of all the start indices of* `p`*'s anagrams in* `s`. You may return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: s = "cbaebabacd", p = "abc"
Output: [0,6]
Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
```

Example 2:

```
Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
```

A: PYTHON doubts:

→

- hashmap counter initialize ⇒ **sCounter, p Counter = {}, {}**
- for loop going till len (p) ⇒ **for i in range(len(p)):**
- counter increment also if already not present how to add ⇒ **sCounter[s[x]] = 1 + sCounter.get(s[x], 0)**
- adding 0 in a list ⇒ **res[0] (directly added to list)**
- pop out from a list ⇒ **sCount.pop(s[l])**

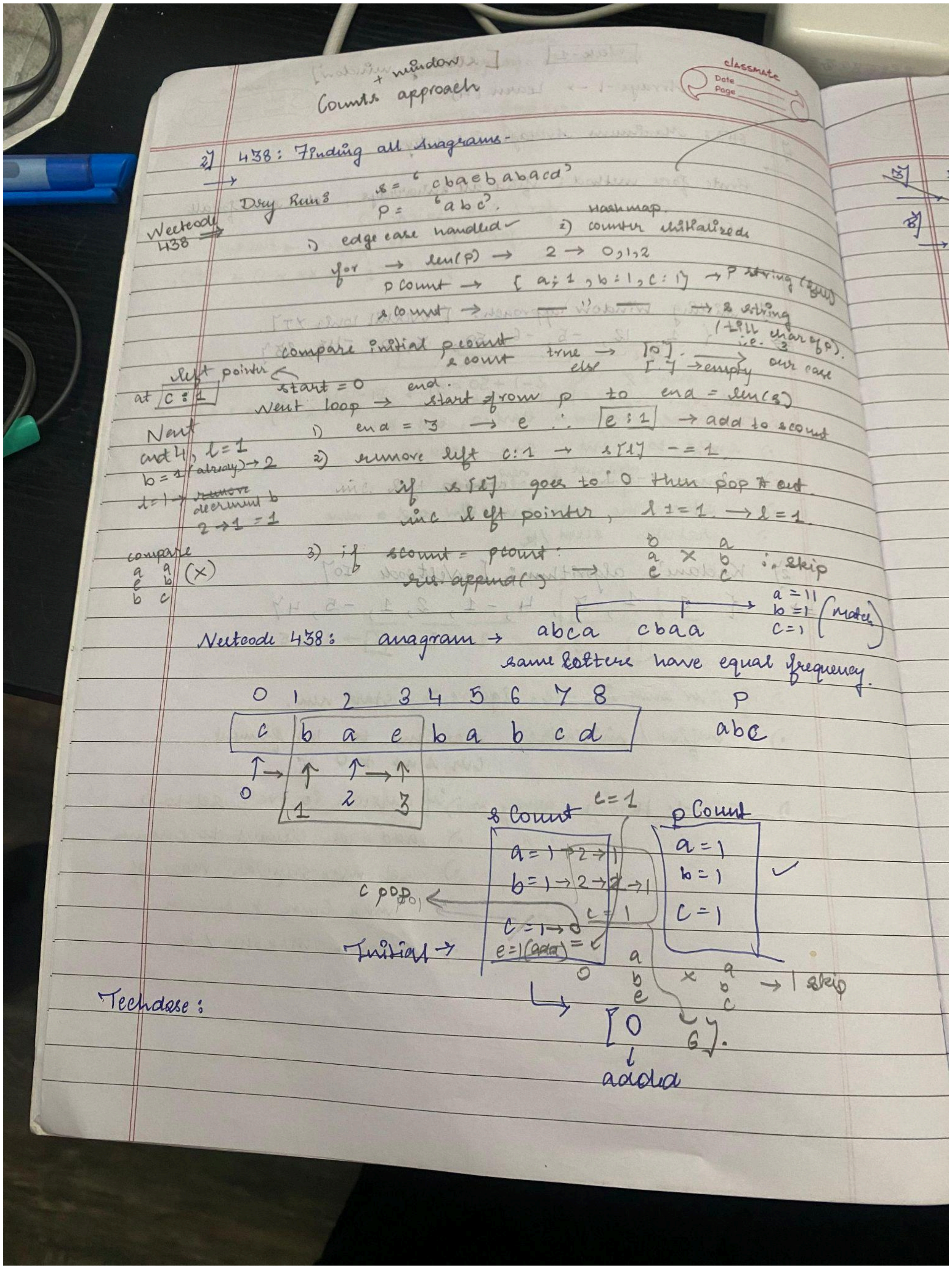
B: Code -

→

```
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        if len(p) > len(s):
            return []
        sCounter, pCounter = {}, {}
        for x in range(len(p)):
            sCounter[s[x]] = 1 + sCounter.get(s[x], 0)
            pCounter[p[x]] = 1 + pCounter.get(p[x], 0)

        result = [0] if sCounter == pCounter else []

        leftPtr = 0
        for rightPtr in range(len(p), len(s)):
            sCounter[s[rightPtr]] = 1 + sCounter.get(s[rightPtr], 0)
            sCounter[s[leftPtr]] -= 1

            if sCounter[s[leftPtr]] == 0:
                sCounter.pop(s[leftPtr])

            leftPtr += 1
            if sCounter == pCounter:
                result.append(leftPtr)

        return result
```

C: Written notes -

→



+ window
Counts approach

2] 438: Finding all Anagrams-
→
Dry Run: s = 'cbaebabacd'
p = 'abc'.

Neetcode 438 ⇒
1) edge case handled ✓ 2) counter initialized: Hashmap.
for → len(p) → 2 → 0,1,2
p count → {a: 1, b: 1, c: 1} → p string (full)
s count → → s string (till char of p). i.e. 3
compare initial p count, s count true → [0]. → our case
else [ ] → empty

left pointer at $\boxed{c : 1}$
start = 0 end.
Next loop → start from p to end = len(s)
1) end = 3 → e ∴ $\boxed{e : 1}$ → add to s count
2) remove left c : 1 → s[l] -= 1
if s[l] goes to 0 then pop it out.
inc left pointer, l += 1 → l = 1
3) if scount = pcount: ~~res.append( )~~ → b a e × a b c ∴ skip

Next
end 4, l = 1
b = 1 (already) → 2
l = 1 → ~~remove~~ decrement b
2 → 1 = 1

compare
a e b | a b c (×)

Neetcode 438: anagram → abca cbaa → a = 1 b = 1 c = 1 (match)
same letters have equal frequency.

0	1	2	3	4	5	6	7	8
c	b	a	e	b	a	b	c	d

p
abc

0 → 1 2 → 3

s Count
a = 1 → 2 → 1
b = 1 → 2 → 2 → 1
c = 1 → 0
e = 1 (add) = c
c = 1
c pop
Initial →
c = 1

p Count
a = 1
b = 1
c = 1
✓

a b e × a b c → skip

[0 6].
↓
added

Techdose:

Rewrite nicely

~~8) 567 : Permutation in string:~~

8) Code:

```
1) handle edge case → $n_1 > n_2$ , return 0.
2) Initialize Hashmap counter : s Counter = { } , ______
3) Get Initial counts → till p
     add 1 for char present → use .get (if already not present)
4) if both counter match then add 0 to result list
   (as it is the start index of matching anagram)
5) Left / start = 0.
6)   Right → len(p) to len(s) . (for)
   →   add ~~the~~ 1 to the letter correspond to index at right
       dec 1 from left
        ↳ if left index value = 0 ⇒ pop it out.
                                        .pop.
7) Left ++
8) if scounter = pcounter → .append to result list
```