LeetCode 438. Find All Anagrams in a String

1. Problem Title & Link

- **438. Find All Anagrams in a String**
- 🔗 https://leetcode.com/problems/find-all-anagrams-in-a-string/

2. Problem Statement (Short Summary)

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

3. Examples (Input → Output)

Input: s = "cbaebabacd", p = "abc"
Output: [0,6]
Explanation:
- Substring "cba" starting at index 0 is an anagram of "abc".
- Substring "bac" starting at index 6 is an anagram of "abc".

Input: s = "abab", p = "ab"
Output: [0,1,2]

4. Constraints

- 1 <= s.length, p.length <= $3 * 10^4$
- s and p consist of lowercase English letters.

5. Thought Process (Step by Step)

This problem extends **Valid Anagram (LC 242)** but applied to **every substring** of length len(p) in s.
We can't sort every substring — that's O(n·m·logm).
Instead, we use a **sliding window with frequency counters** ❤️

🧩 Step 1: Frequency Target

Build a frequency array p_count[26] for string p.

🧩 Step 2: Sliding Window

Maintain a window of size len(p) in string s:

- Add new character (right pointer).
- Remove old character (left pointer when window too big).
- Compare window count with p_count.

If counts match → it's an anagram → record the start index.

🧩 Step 3: Efficient Comparison

We can avoid full array comparison each time using a matches counter:

- Keep track of how many characters currently match between s_count and p_count.
- Update only when a character enters or leaves the window.

But for simplicity (especially for students), we can start with the direct count comparison version.

6. Pseudocode

```
initialize p_count[26] = 0, s_count[26] = 0
for c in p:
    p_count[ord(c) - ord('a')]++

result = []
for i in range(len(s)):
    add s[i] to window count
    if i >= len(p):
        remove s[i - len(p)] from window
    if s_count == p_count:
        add (i - len(p) + 1) to result
return result
```

7. Code Implementation

✅ **Python**

```python
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        if len(p) > len(s):
            return []

        p_count = [0] * 26
        s_count = [0] * 26
        res = []

        for ch in p:
            p_count[ord(ch) - ord('a')] += 1

        for i in range(len(s)):
            s_count[ord(s[i]) - ord('a')] += 1

            if i >= len(p):
                s_count[ord(s[i - len(p)]) - ord('a')] -= 1

            if s_count == p_count:
                res.append(i - len(p) + 1)
```

```
        return res
```

✅ **Java**

```
class Solution {
    public List<Integer> findAnagrams(String s, String p) {
        List<Integer> res = new ArrayList<>();
        if (p.length() > s.length()) return res;

        int[] pCount = new int[26];
        int[] sCount = new int[26];

        for (char c : p.toCharArray()) pCount[c - 'a']++;

        for (int i = 0; i < s.length(); i++) {
            sCount[s.charAt(i) - 'a']++;

            if (i >= p.length())
                sCount[s.charAt(i - p.length()) - 'a']--;

            if (Arrays.equals(pCount, sCount))
                res.add(i - p.length() + 1);
        }
        return res;
    }
}
```

8. Time & Space Complexity

- **Time:** O(n + 26·n) → O(n)
- **Space:** O(1) — fixed-size frequency arrays (26 letters)

9. Dry Run (Step-by-Step Execution)

👉 Input:
s = "cbaebabacd", p = "abc"
p_count = [a:1, b:1, c:1]

Step	i	Window	s_count	Match?	Output
0	0	"c"	c:1	❌	[]
1	1	"cb"	c:1,b:1	❌	[]
2	2	"cba"	c:1,b:1,a:1	✅	[0]
3	3	"bae"	b:1,a:1,e:1	❌	[0]

4	4	"aeb"	a:1,e:1,b:1	❌	[0]
5	5	"eba"	e:1,b:1,a:1	❌	[0]
6	6	"bac"	b:1,a:1,c:1	✅	[0,6]

✅ Output: [0,6]

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Sliding Window + Frequency Array — track character counts dynamically as window slides.	- Find permutations/anagrams in streams - Compare character multisets efficiently - Frequency window matching	- Forgetting to decrement count when window slides - Comparing arrays inefficiently - Edge cases when p > s	🔷 Builds to **LC 567 (Permutation in String)** 🔷 Connects to **LC 76 (Minimum Window Substring)** 🔷 Introduces "moving frequency window" pattern

11. Common Mistakes / Edge Cases

- Missing check when len(p) > len(s) → must return [].
- Forgetting to remove old character as window slides.
- Using dictionary comparison (slower than arrays).

12. Variations / Follow-Ups

- **LC 567:** Check if any permutation of p exists in s (same logic, return True/False).
- **LC 76:** Minimum Window Substring → inverse problem (cover instead of match).
- Use same logic to detect **anagram clusters in a stream**.