438. Find All Anagrams in a String

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".
```

Constraints:

- `1 <= s.length, p.length <= 3 * 10<sup>4</sup>`
- `s` and `p` consist of lowercase English letters.

```
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        n = len(p)
        pattern = collections.Counter(p)
        checkPattern = collections.Counter(s[:n])
        ans = []
        j = 0
        i = n
        while i<len(s):
            if self.compare(pattern,checkPattern):
                ans.append(i-n)
```

```
            ch = s[i]
            checkPattern[ch] = checkPattern.get(ch,0)+1
            char = s[i-n]

            if checkPattern[char]==1:
                del checkPattern[char]
            else:
                checkPattern[char] = checkPattern.get(char,0)-1
            i = i+1
            # j = j+1
        if self.compare(pattern,checkPattern):
                ans.append(i-n)
        return ans




def compare(self,map1,map2):
    for key in map1.keys():
        if map1[key]!=map2[key]:
            return False
    return True
```