Documentation: Substring with Concatenation of All Words

Problem Description

You are given a string `s` and an array of strings `words`. All the strings in `words` are of the same length. A concatenated substring in `s` is a substring that contains all the strings of any permutation of `words` concatenated. You need to find and return the starting indices of all the concatenated substrings in `s`. The order of the output does not matter.

Examples

Example 1:

Input: s = "barfoothefoobarman", words = ["foo","bar"]

Output: [0,9]

Explanation:

- The substring starting at index 0 is "barfoo", which is the concatenation of ["bar","foo"] (a permutation of words).
- The substring starting at index 9 is "foobar", which is the concatenation of ["foo","bar"] (a permutation of words).

Example 2:

Input: s = "wordgoodgoodgoodbestword", words = ["word","good","best","word"]

Output: []

Explanation:

- There is no substring of length 16 in s that is equal to the concatenation of any permutation of words.
- Thus, an empty array is returned.

Example 3:

Input: s = "barfoofoobarthefoobarman", words = ["bar","foo","the"]

Output: [6,9,12]

Explanation:

- The substring starting at index 6 is "foobarthe", which is the concatenation of ["foo","bar","the"] (a permutation of words).

- The substring starting at index 9 is "barthefoo", which is the concatenation of ["bar","the","foo"] (a permutation of words).

- The substring starting at index 12 is "thefoobar", which is the concatenation of ["the","foo","bar"] (a permutation of words).

Constraints

- 1 <= s.length <= 10^4
- 1 <= words.length <= 5000
- 1 <= words[i].length <= 30
- `s` and `words[i]` consist of lowercase English letters.

Approach

1. Check if either `s` or `words` is empty. If so, return an empty list.
2. Determine the length of each word in `words`, the total length of all the words combined (`total_len`), and count the occurrences of each word in `words`.
3. Iterate through each index `i` in `s` up to `len(s) - total_len + 1`.
4. At each index `i`, initialize an empty Counter `seen` to count the occurrences of words encountered so far.
5. Iterate through each word in `s` starting from index `i` with a step of `word_len`.
6. Check if the current word is in `words_count`. If so, increment its count in `seen`.
7. If the count of the current word in `seen` exceeds its count in `words_count`, break the loop.
8. If `seen` matches `words_count` after iterating through all words, add `i` to the result.
9. Finally, return the result.