2063. Vowels of All Substrings

Medium Math String Dynamic Programming Combinatorics

Problem Description

The task is to calculate the total number of vowels present in all possible substrings of a given string, `word`. Vowels are the characters 'a', 'e', 'i', 'o', and 'u'. A substring is defined as any continuous sequence of characters within the string. It's important to note that since we are counting vowels in every possible substring, a single vowel can contribute to the count multiple times, depending on its position in the string and the number of substrings that include it. Also, the total count may be very large, so special attention should be paid to avoid integer overflow during the calculation.

Intuition

To find the solution without checking every possible substring directly, we can consider how many times each vowel contributes to the total count. A character at position `i` in the string `word` of length `n` is part of `i + 1` substrings starting at or before position `i` and part of `n - i` substrings ending at or after position `i`. Therefore, that character is found in a total of `(i + 1) * (n - i)` substrings.

By looping through the characters of the word and checking if each character is a vowel, we can use the above formula to count how many times each vowel appears across all substrings. Then, summing these counts gives us the total number of vowels in all substrings. Python's list comprehension makes this process concise and efficient, as seen in the provided solution. This approach significantly reduces the computational complexity compared to the naive approach of generating and checking every possible substring individually.

Solution Approach

The solution to this problem utilizes a single-pass algorithm along with the mathematical concept that each character in a string contributes to several substrings based on its position.

Here's a walk-through of the implementation steps with reference to the provided Python code:

1. We determine the length `n` of the string `word`.
2. We initialize a total sum of vowel occurrences in all substrings, which will be our final result. This sum starts at 0.
3. We then iterate through the string `word` using a for loop with `enumerate`, which gives us each character `c` along with its index `i`.
4. Within the loop, we check if the current character `c` is a vowel. This is accomplished by checking if `c` is in the string `'aeiou'`.
5. For every vowel found, we calculate its contribution to the total count. The number of times this vowel appears in all substrings is given by `(i + 1) * (n - i)`, where `i + 1` is the number of possible starting points for substrings that include this character, and `n - i` is the number of possible ending points.
6. We add this contribution to the total sum for each vowel encountered.
7. Finally, the total sum is returned as the result.

The provided solution uses a concise list comprehension that combines these steps into a single line. The `sum` function calculates the aggregate total by adding up the contributions from each vowel character in the string. Here's the core logic in the code:

```
sum((i + 1) * (n - i) for i, c in enumerate(word) if c in 'aeiou')
```

This solution is efficient because it only involves processing each character of the string once and does not require generating all substrings explicitly. It has a time complexity of O(n), where n is the length of the string since it iterates through the string once, and each operation within the loop is constant time. There's no complex data structure used, and the space complexity is O(1) as it only keeps track of the total sum and loop variables.

Example Walkthrough

Let's go through an example to illustrate the solution approach using the string `word = "abcde"`.

1. First, we determine the length of the string `word`, which is `n = 5`.
2. We initialize the total sum of vowel occurrences in all substrings to 0. This will hold our result.
3. Now, we iterate through the string `word`. For each iteration, we consider both the character `c` and its index `i`.
4. Let's walk through each character of the string `abcde`:
   - For character `a` at index `i = 0`:
     1. We check if `a` is a vowel — yes, it is.
     2. We calculate its contribution: `(0 + 1) * (5 - 0) = 5`.
     3. We add this to the total sum, so the sum is now 5.
   - For character `b` at index `i = 1`:
     1. `b` is not a vowel, so we do not count it.
   - For character `c` at index `i = 2`:
     1. `c` is also not a vowel, so we continue.
   - For character `d` at index `i = 3`:
     1. `d` is not a vowel, so we skip it as well.
   - For character `e` at index `i = 4`:
     1. We check if `e` is a vowel — it is.
     2. We calculate its contribution: `(4 + 1) * (5 - 4) = 5`.
     3. We add this to the total sum, which becomes `5 (from a) + 5 (from e) = 10`.
5. Since there are no more characters in the string, we have completed our iteration. The total vowel occurrences in all possible substrings of `"abcde"` equal 10.
6. We return 10 as the final result.

This walkthrough demonstrates the core logic of the provided Python code:

```
sum((i + 1) * (5 - i) for i, c in enumerate("abcde") if c in 'aeiou')
```

By following these steps, we efficiently calculate the total number of vowel occurrences in all possible substrings of a given string.

Solution Implementation

Python

```
class Solution:
    def count_vowels(self, word: str) -> int:
        # Calculate the length of the word
        word_length = len(word)

        # Initialize a variable to hold the total count of vowels in all substrings
        vowel_count = 0

        # Iterate through each character in the word along with its index
        for index, char in enumerate(word):
            # Check if the current character is a vowel
            if char in 'aeiou':
                # If it's a vowel, calculate the contribution of this vowel to the total count
                # The vowel contributes to all substrings that start before (or at) this character
                # and end after (or at) this character. The number of such substrings can be calculated
                # by multiplying the number of ways to choose a start point (index + 1 choices) by
                # the number of ways to choose an end point (word_length - index choices).
                contribution = (index + 1) * (word_length - index)
                # Add the contribution of this vowel to the total count
                vowel_count += contribution

        # Return the total count of vowels in all substrings
        return vowel_count
```

Java

```
class Solution {
    public long countVowels(String word) {
        long totalCount = 0; // Initialize total count of vowels in substrings
        int wordLength = word.length(); // Cache the length of the word for efficiency

        // Iterate over each character in the word
        for (int i = 0; i < wordLength; ++i) {
            char currentChar = word.charAt(i); // Get the character at the current index

            // Check if the current character is a vowel
            if (currentChar == 'a' || currentChar == 'e' || currentChar == 'i' || currentChar == 'o' || currentChar == 'u') {
                // If it's a vowel, add to the total count based on its position and remaining length
                // The count is the product of the number of times this vowel can appear in
                // the start of a substring (i+1 times) and in the end of a substring (wordLength - i times)
                totalCount += (long) (i + 1) * (wordLength - i);
            }
        }

        // Return the total count of vowels found in all the substrings
        return totalCount;
    }
}
```

C++

```
class Solution {
public:
    // Function to count the total number of vowel appearances in all substrings of the word.
    long long countVowels(string word) {
        long long totalVowels = 0; // Initialize a variable to keep track of the total vowels count.
        int wordLength = word.size(); // Find the length of the word once to avoid multiple calls to `size()`.

        // Loop through each character of the word.
        for (int i = 0; i < wordLength; ++i) {
            char currentChar = word[i]; // Store the current character to compare it against vowels.

            // Check if the character is a vowel - 'a', 'e', 'i', 'o', or 'u'.
            if (currentChar == 'a' || currentChar == 'e' || currentChar == 'i' ||
                currentChar == 'o' || currentChar == 'u') {
                // The number of times this vowel character will appear in all possible substrings
                // is the product of its position (1-indexed) and its "rightward reach" (n - i).
                totalVowels += (static_cast<long long>(i) + 1) * (wordLength - i);
            }
        }

        // Return the total count of vowel appearances in all substrings.
        return totalVowels;
    }
};
```

TypeScript

```
// Function to count the total number of vowels in a given string, where each vowel
// is counted as many times as it contributes to each possible substring.
// @param {string} word - The word to count vowels in
// @returns {number} - The total count of vowels contributing to substrings
function countVowels(word: string): number {
    const wordLength = word.length;
    let vowelCount = 0;

    // Loop over all characters in the word
    for (let index = 0; index < wordLength; ++index) {
        // Check if the current character is a vowel
        if (['a', 'e', 'i', 'o', 'u'].includes(word[index])) {
            // If it is a vowel, add the count to the running total, factoring in
            // the number of possible substrings it can be part of.
            // Multiply the position of the vowel in the string (index + 1)
            // by the number of ways to choose the ending of the substring (wordLength - index)
            vowelCount += (index + 1) * (wordLength - index);
        }
    }

    // Return the total count of vowels
    return vowelCount;
}


class Solution:
    def count_vowels(self, word: str) -> int:
        # Calculate the length of the word
        word_length = len(word)

        # Initialize a variable to hold the total count of vowels in all substrings
        vowel_count = 0

        # Iterate through each character in the word along with its index
        for index, char in enumerate(word):
            # Check if the current character is a vowel
            if char in 'aeiou':
                # If it's a vowel, calculate the contribution of this vowel to the total count
                # The vowel contributes to all substrings that start before (or at) this character
                # and end after (or at) this character. The number of such substrings can be calculated
                # by multiplying the number of ways to choose a start point (index + 1 choices) by
                # the number of ways to choose an end point (word_length - index choices).
                contribution = (index + 1) * (word_length - index)
                # Add the contribution of this vowel to the total count
                vowel_count += contribution

        # Return the total count of vowels in all substrings
        return vowel_count
```

Time and Space Complexity

The given Python code snippet defines a function that counts the number of times a vowel appears in a string, such that each occurrence of a vowel is counted as many times as the number of substrings that include it.

Time Complexity: O(n)

The time complexity of the function is O(n) where `n` is the length of the input string `word`. This is because the function contains a single loop (created by the `sum` function and list comprehension) that iterates over each character in the string exactly once. Inside the loop, each step involves simple arithmetic operations that are constant time, namely `(i + 1) * (n - i)`, and a membership test `c in 'aeiou'`, which is also constant time since the set of vowels is of fixed size.

The calculation of the arithmetic progression is done for each vowel character in the string, thus the total operations scale linearly with the size of the input `n`.

Space Complexity: O(1)

The space complexity of the function is O(1). There are no data structures used that grow with the input size. Temporary variables like `i` and `c` are reused for each character in the string. The string `'aeiou'` is a constant space that does not change with the input. Thus, the algorithm uses a fixed amount of space regardless of the size of the input.