2586. Count the Number of Vowel Strings in Range

Easy Array String

Leetcode Link

Problem Description

In this problem, you're given an array of strings named `words`, and two integers `left` and `right` that represent the indices within the array. The task is to count how many strings from the sub-array starting at index `left` and ending at index `right` (both inclusive) are "vowel strings." A vowel string is defined as a string that both starts and ends with a vowel character. The vowels in this case are the characters 'a', 'e', 'i', 'o', and 'u'. The answer you need to provide is the total number of such vowel strings that exist within the specified range of indices in the `words` array.

Intuition

The solution approach is fairly straightforward and relies on a direct check of each string within the specified bounds (`left` to `right` inclusive). For each string, you need to perform two checks: (1) whether the first character is a vowel, and (2) whether the last character is a vowel. If both conditions are true, then that string qualifies as a vowel string.

To arrive at this solution, it's clear that you must visit each string within the given index range; hence, a loop or iterator is necessary. Within the loop, you need to examine the specific characters of the string - only the first and last characters since these are the ones that determine if a string is a vowel string according to the problem definition.

The use of the Python slicing notation `words[left:right+1]` simplifies accessing the sub-array we're interested in. The `sum` function is a concise way to count the number of `True` evaluations, thereby providing us the total count of vowel strings. By using a generator expression, it performs the check and counts in a single line without the need for an explicit loop or additional variables for counting.

Thus, the approach is to execute the steps as a single inline operation that efficiently traverses the relevant subset of strings and tallies those that meet the vowel string criteria.

Solution Approach

The implementation of the Reference Solution Approach leans on a simple but effective strategy. It is a simulation of the conditions directly stated in the problem description with a healthy usage of Python's expressive syntax and built-in functions.

Here's a step-by-step description of how the solution works:

1. **Slicing**: The first operation is to slice the `words` array to obtain the relevant sub-array. This is done using the Python slicing syntax `words[left:right+1]`. The `left:right+1` slice notation selects all the elements from index `left` up to and including the index `right`.

2. **Generator Expression**: Next, a generator expression is used to iterate over each word in the sliced sub-array of `words`. Generator expressions are a compact and memory-efficient way to work with sequences in Python. They allow for iterating over data without creating an intermediate list in memory, which would be the case if a list comprehension was used.

3. **Conditional Check**: During the iteration, the condition that each word `w` starts with a vowel (`w[0] in 'aeiou'`) and ends with a vowel (`w[-1] in 'aeiou'`) is checked. The `w[0]` and `w[-1]` syntax accesses the first and last characters of the string `w`, respectively. The `in 'aeiou'` part checks if a given character is among the characters 'a', 'e', 'i', 'o', 'u'.

4. **Summation**: Finally, the `sum()` function wraps the generator expression. `sum` adds up the `True` (which count as 1) evaluations of the conditional checks for every word in the sliced array. Since `False` is equivalent to 0, only the `True` cases contribute to the sum. The result is the count of all strings that are vowel strings within the given index range.

No additional data structures are used as this approach operates directly on the input data and produces the sum immediately, following an efficient memory and time usage pattern.

Here's a bit of code from the solution which encapsulates the process:

```
return sum(
    w[0] in 'aeiou' and w[-1] in 'aeiou' for w in words[left : right + 1]
)
```

Each `w` represents a string within the specified range. The `in` keyword is used twice to perform the vowel checks, and the `and` keyword connects the two conditions. Only when both conditions are satisfied does the expression evaluate to `True`, subsequently increasing the sum total. This neat inline loop-and-check code effectively solves the problem in an idiomatic and Pythonic way.

Example Walkthrough

Let's consider an example to illustrate the solution approach described above. Suppose we have an array of strings and we want to count the number of vowel strings within a specified range of indices:

`words` = ["apple", "banana", "anaconda", "eagle", "kiwi", "onion", "ubi", "orange"]

Let's say `left = 2` and `right = 5`. The sub-array we need to evaluate is from index 2 to index 5, which includes the words: ["anaconda", "eagle", "kiwi", "onion"]. Now, let's apply the solution step by step:

1. **Slicing**: We slice the `words` array using `words[left:right+1]`, which results in ["anaconda", "eagle", "kiwi", "onion"].

2. **Generator Expression**: Using the generator expression, we iterate over each word in this sliced sub-array.

3. **Conditional Check**:
   - "anaconda": Starts with 'a' and ends with 'a', both of which are vowels. Hence, this word satisfies the condition.
   - "eagle": Starts with 'e' and ends with 'e', both vowels. This word satisfies the condition.
   - "kiwi": Starts with 'k' and ends with 'i'. Since 'k' is not a vowel, this word does not satisfy the condition.
   - "onion": Starts with 'o' and ends with 'n', and 'n' is not a vowel. This word also does not satisfy the condition.
4. **Summation**: The expression will evaluate to `True` for "anaconda" and "eagle", resulting in a count of 2. For "kiwi" and "onion", the expression evaluates to `False`.

Therefore, according to our generator expression, we have:

```
sum(
    w[0] in 'aeiou' and w[-1] in 'aeiou' for w in ["anaconda", "eagle", "kiwi", "onion"]
)
```

The sum would be calculated as `1 + 1 + 0 + 0 = 2`. Hence, there are 2 vowel strings in the array within the range of indices 2 to 5.

The inline code for this example simply counts the occurrences where both conditions are met and provides us with the final answer. In this case, the function call would look like this:

```
count_vowel_strings(words, 2, 5)
```

And it would return `2`, which is exactly the number of strings that start and end with a vowel in the specified range.

Python Solution

```
# Import the List type from the typing module to enable type annotations for lists
from typing import List

class Solution:
    def vowelStrings(self, words: List[str], left: int, right: int) -> int:
        # Initialize a counter for the number of strings that meet the criteria.
        number_of_vowel_strings = 0

        # Loop through the slice of the words list starting from `left` and ending with `right` (inclusive).
        for word in words[left:right + 1]:
            # Check if the first and last characters of word are both vowels.
            if word[0] in 'aeiou' and word[-1] in 'aeiou':
                # If they are, increment the counter.
                number_of_vowel_strings += 1

        # Return the total count of strings that start and end with a vowel.
        return number_of_vowel_strings
```

Java Solution

```
class Solution {
    // Method to count words with vowels at start and end positions within a given range
    public int vowelStrings(String[] words, int left, int right) {
        // Initialize the answer to count valid words
        int validWordCount = 0;

        // Iterate over the specified range in the array
        for (int i = left; i <= right; ++i) {
            String word = words[i]; // Current word being checked
            // Check if both the first and last characters of the word are vowels
            if (isVowel(word.charAt(0)) && isVowel(word.charAt(word.length() - 1))) {
                // Increment the count of valid words
                ++validWordCount;
            }
        }

        // Return the count of valid words
        return validWordCount;
    }

    // Helper method to check if a character is a vowel
    private boolean isVowel(char character) {
        // Check for all lowercase vowels (this code assumes input will be in lowercase)
        return character == 'a' || character == 'e' || character == 'i' || character == 'o' || character == 'u';
    }
}
```

C++ Solution

```
class Solution {
public:
    // Function to count strings in a vector of strings where each string
    // starts and ends with a vowel.
    // The function checks elements from index 'left' to 'right' (inclusive).
    int vowelStrings(vector<string>& words, int left, int right) {
        // Lambda function to check whether a character is a vowel
        auto isVowel = [](char c) -> bool {
            return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
        };

        // Initialize count of valid strings to 0
        int validStringCount = 0;

        // Iterate over the specified range of indices
        for (int i = left; i <= right; ++i) {
            // Get the current word
            auto currentWord = words[i];

            // Increment count if the first and last characters of the current word are vowels.
            validStringCount += isVowel(currentWord[0]) && isVowel(currentWord[currentWord.size() - 1]);
        }

        // Return the total count of strings starting and ending with a vowel
        return validStringCount;
    }
};
```

Typescript Solution

```
function vowelStrings(words: string[], left: number, right: number): number {
    // Initialize ans variable to hold the count of strings meeting the condition.
    let ans = 0;
    // Define the array of vowel characters for easy checking.
    const vowels: string[] = ['a', 'e', 'i', 'o', 'u'];
    // Iterate over the specified range of indices from left to right inclusive.
    for (let i = left; i <= right; ++i) {
        // Get the current word to check based on the index.
        const currentWord = words[i];
        // Check if the first and last characters of the current word are vowels.
        if (vowels.includes(currentWord[0]) && vowels.includes(currentWord[currentWord.length - 1])) {
            // Increment the count if both the first and last characters are vowels.
            ++ans;
        }
    }
    // Return the total count of strings that start and end with a vowel.
    return ans;
}
```

Time and Space Complexity

The time complexity of the provided function `vowelStrings` is `O(m)`, where `m` is the number of elements processed by the sum operation, calculated by the given range `right - left + 1`. The function iterates through each element in the slice `words[left : right + 1]` once to compute the sum, resulting in a linear relationship between the number of elements and the time taken, hence `O(m)`.

The space complexity of the code is `O(1)` as there are no additional data structures used that grow with the size of the input. The variables used in the list comprehension within the sum function do not require extra space that depends on the size of the input list, thus it remains constant regardless of the input size.