2559. Count Vowel Strings in Ranges

Prefix Sum

## **Problem Description**

**Array** 

Medium

String )

the first and the last character are both vowels. The range for each query is defined by two indices [1\_i, r\_i], where 1\_i is the start index and ri is the end index. Our goal is to answer each query by counting the relevant strings and returning the results in an array ans, where each element ans[i] corresponds to the count for the i-th query. The vowel characters are specifically 'a', 'e', 'i', 'o', and 'u'. The array words is 0-indexed, meaning the first element has an index of

The problem presents us with an array of strings words and a two-dimensional array queries. We need to determine how many

strings within a specific index range in the words array start and end with a vowel. A string is said to start and end with a vowel if

0, the second an index of 1, and so on. This indexing is important to consider when processing the ranges specified in each query within queries.

Intuition

#### Precomputation with Cumulative Sums: The idea is to precompute the number of valid strings (strings that start and end with a vowel) up to each index in the words array. This is achieved by creating an auxiliary array s where each element s[i]

stores the cumulative count of valid strings from words[0] to words[i-1]. This allows us to quickly calculate the number of valid strings in any query range with a simple subtraction: s[r + 1] - s[l], where l and r are the start and end indices of

To efficiently solve this problem, we leverage two techniques:

- the query. Checking String Endpoints: For each string in words, we only need to check the first and the last character to determine if it qualifies as a valid string. We define the set of vowels and then, for each word, check if both endpoints are in that set. The Python expression int(w[0] in vowels and w[-1] in vowels) evaluates to 1 if both conditions are true and 0 otherwise.
- cumulative sum array, resulting in a significant reduction of the time complexity, especially when dealing with a large number of queries over the words array. **Solution Approach**

By combining these two ideas, the solution efficiently computes the answer to each query by referencing the precomputed

The Solution class has a method named vowelStrings which takes in two parameters: words, a list of strings, and queries, a 2D list of integers. The output is a list of integers representing the counts of strings in words that satisfy the query conditions (start and end with a vowel).

#### **Defining a Set of Vowels:** A set vowels is created with all the vowel characters {'a', 'e', 'i', 'o', 'u'}. This allows for constant time complexity 0(1) for checking if a character is a vowel since set membership checking is typically 0(1) due to

hashing.

Calculating Cumulative Sums with accumulate: The accumulate function from Python's itertools module is used to generate the cumulative sum of valid strings. For every word w in words, an expression int(w[0] in vowels and w[-1] in vowels) is computed which results in 1 if the word starts and ends with a vowel and 0 otherwise. These values are accumulated to get cumulative sums. The initial=0 parameter is used to start the accumulation from 0 which implicitly

adds an extra 0 at the beginning of the accumulated list, making the cumulative sum array s one element longer than the words. This simplification helps in handling edge cases and simplifies the subtraction step in the queries.

Here's a closer look at how the solution is implemented:

time 0(1) for each query. The result of this subtraction gives the count of valid strings within the specified range of indices, which is then appended to the resulting list. Return the Results: A list comprehension is used to iterate through each query in queries, and the described subtraction operation is applied to produce the final answer list, which is then returned.

Handling Queries: With the cumulative sums array s ready, each query [1, r] is processed to find the count of valid strings

in the range by subtracting the cumulative sum up to 1 from the cumulative sum up to r + 1. This operation takes constant

Hence the entire solution has an overall time complexity of 0(N + Q) which is highly efficient, especially for large datasets. All in all, this approach utilizes space-time trade-offs (precomputing the cumulative sum) and efficient data structure access patterns (using a set and direct indexing) to provide an optimal solution to the problem.

The overall time complexities for initializing the cumulative sum and for processing each query are 0(N) and 0(1) respectively,

where N is the total number of words in words. Given Q queries, the total time complexity for the query processing part is O(Q).

• queries: [[1, 3], [0, 4], [2, 2]] First, we will start by identifying which words start and end with a vowel:

Let's go through a walkthrough example to illustrate the solution approach. Suppose we are given an array of strings words and a

### • "echo" starts with 'e' and ends with 'o', so it's valid. Next, we create a cumulative sum array s which will hold the number of valid strings at each index. It will look like this:

zero at the beginning):

For the query [2, 2], we do:

Solution Implementation

from itertools import accumulate

from typing import List

class Solution:

does not start and end with a vowel.

**Example Walkthrough** 

2D array queries as follows:

words: ["apple", "orange", "coop", "peach", "echo"]

• "apple" starts with 'a' and ends with 'e', so it's valid.

"coop" does not end with a vowel, so it's not valid.

• "peach" does not start with a vowel, so it's not valid.

• "orange" starts with 'o' and ends with 'e', so it's valid.

• s: [0, 1, 2, 2, 2, 3] Now, we process each query:

For the query [1, 3], we subtract the cumulative sum at index 1 from the cumulative sum at index 3 + 1 (since s has an extra

• s[2 + 1] - s[2] = s[3] - s[2] = 2 - 2 = 0. There are no valid strings at index 2 in words, because index 2 to 2 is the string "coop" which

We have used the vowels set to identify valid strings, the accumulate function to prepare the cumulative sum array s, and

• s[3 + 1] - s[1] = s[4] - s[1] = 2 - 1 = 1. So there is one valid string in the range from index 1 to 3 in words.

• s[4 + 1] - s[0] = s[5] - s[0] = 3 - 0 = 3. There are three valid strings in the range from index 0 to 4 in words.

- For the query [0, 4], we do:
- ans: [1, 3, 0]

**Python** 

simple subtraction for each query to get the counts efficiently. This outlines the solution approach and demonstrates how it would work with an actual set of inputs.

cumulative count = list( accumulate( [int(words[i][0] in vowels and words[i][-1] in vowels) for i in range(len(words))], initial=0

# A set of vowels is initialized for quick lookup.

# begin and end with a vowel up to the current index.

vowels = {'a', 'e', 'i', 'o', 'u'}

def vowel strings(self, words: List[str], queries: List[List[int]]) -> List[int]:

# This creates a cumulative sum list which indicates the count of words that

# and the last characters of word 'w' are vowels, contributing to the sum.

# 'int(w[0] in vowels and w[-1] in vowels)' evaluates to 1 when both the first

results = [cumulative\_count[r + 1] - cumulative\_count[l] for l, r in queries]

# print(result) # Output would be the number of words with vowel start and end in the given ranges

// Function to determine the number of words with a starting and ending vowel in given queries.

// Function to calculate the number of strings in subarrays that start and end with vowels.

// Initialize a prefix sum array with an extra element for simplifying calculations.

// Build the prefix sum array with the count of words starting and ending with a vowel.

prefixSum[i + 1] = prefixSum[i] + (vowels.count(firstChar) && vowels.count(lastChar));

// This gives the number of strings with first and last vowels in the range [l.r]

vector<int> vowelStrings(vector<string>& words, vector<vector<int>>& queries) {

char firstChar = words[i][0], lastChar = words[i].back();

// Retrieve the left and right boundaries for this query.

// Calculate the result for the current query using prefix sums.

ans.push\_back(prefixSum[rightBoundary + 1] - prefixSum[leftBoundary]);

int leftBoundary = query[0]. rightBoundary = query[1];

// Define the set of vowels for easy checking later.

// Get the size of the words array for future loops.

// Prepare a vector to store answers for queries.

// Return the final answers for all the queries.

// Iterate through the gueries to calculate the result.

int numWords = words.size();

for (auto& guery : gueries) {

vector<int> ans;

return ans;

vector<int> prefixSum(numWords + 1, 0);

for (int i = 0; i < numWords; ++i) {

unordered\_set<char> vowels = {'a', 'e', 'i', 'o', 'u'};

# result = solution.vowel strings(["apple", "banana", "kiwi"], [[0, 1], [1, 2]])

public int[] vowelStrings(String[] words, int[][] queries) {

// Set containing all the vowels for easy reference.

Therefore, the final result (counts of valid strings for each query) is:

```
# For each query, calculate the number of words that start and end with a vowel
# in the given range by subtracting the cumulative count at the start index from
# the cumulative count at the end index (+1 because of the way the cumulative
```

# Usage

Java

C++

public:

#include <vector>

#include <string>

class Solution {

using namespace std;

#include <unordered set>

# count is constructed).

return results

# solution = Solution()

class Solution {

```
Set<Character> vowels = Set.of('a', 'e', 'i', 'o', 'u');
// The number of words in the input array.
int wordCount = words.length;
// An array to store the cumulative count of words that start and end with a vowel.
int[] cumulativeVowelCount = new int[wordCount + 1];
// Loop through each word to populate the `cumulativeVowelCount` array.
for (int i = 0; i < wordCount; ++i) {</pre>
    char firstChar = words[i].charAt(0); // First character of the current word.
    char lastChar = words[i].charAt(words[i].length() - 1); // Last character of the current word.
    // Update the cumulative count. Increment if both first and last chars are vowels.
    cumulativeVowelCount[i + 1] = cumulativeVowelCount[i] + (vowels.contains(firstChar) && vowels.contains(lastChar) ? 1 : 0)
// The number of queries to process.
int queryCount = queries.length;
// Array to hold the answers to each query.
int[] answer = new int[queryCount];
// Process each query to get the number of words that start and end with a vowel.
for (int i = 0; i < queryCount; ++i) {
    int leftIndex = queries[i][0], rightIndex = queries[i][1]; // Extracting the range from the query.
    // Answer for this guery is the difference in the cumulative counts.
    answer[i] = cumulativeVowelCount[rightIndex + 1] - cumulativeVowelCount[leftIndex];
// Return the array containing the answer to each query.
return answer;
```

**}**;

**TypeScript** // Function to calculate the number of strings within a given range // where both the first and last characters are vowels function vowelStrings(words: string[], queries: number[][]): number[] { // Create a set of vowel characters for easy checking const vowels = new Set(['a', 'e', 'i', 'o', 'u']); // Get the length of the words array const wordsCount = words.length; // Initialize an array to store the cumulative count of words // that start and end with vowels const cumulativeVowelCounts: number[] = new Array(wordsCount + 1).fill(0); // Iterate over the words to populate the cumulative count array for (let index = 0; index < wordsCount; ++index) {</pre> // Check if both the first and last character of the current word are vowels if (vowels.has(words[index][0]) && vowels.has(words[index][words[index].length - 1])) { // Increment the cumulative count at the next index cumulativeVowelCounts[index + 1] = cumulativeVowelCounts[index] + 1; } else { // If not, just carry over the previous cumulative count cumulativeVowelCounts[index + 1] = cumulativeVowelCounts[index]; // Use map to transform the queries into the result by taking the // difference of cumulative counts, thus obtaining the count for each range return queries.map(([left, right]) => { // Subtract the cumulative count at the start of the range from the // cumulative count at the end of the range to get the count of valid words within the range return cumulativeVowelCounts[right + 1] - cumulativeVowelCounts[left]; }); from itertools import accumulate from typing import List class Solution: def vowel strings(self, words: List[str], queries: List[List[int]]) -> List[int]: # A set of vowels is initialized for quick lookup. vowels = {'a', 'e', 'i', 'o', 'u'} # This creates a cumulative sum list which indicates the count of words that # begin and end with a vowel up to the current index. # 'int(w[0] in vowels and w[-1] in vowels)' evaluates to 1 when both the first # and the last characters of word 'w' are vowels, contributing to the sum.

# Time and Space Complexity

return results

# solution = Solution()

**Time Complexity** 

**Space Complexity** 

# Usage

cumulative count = list(

initial=0

# count is constructed).

accumulate(

The time complexity of the code mainly comes from three parts: 1. Creating the list comprehension which checks every word in the input list words. This process is O(n) where n is the number of words in words.

Since accumulate goes through each element in the provided list, this is O(n).

queries, this part has a complexity of O(q). Combining these gives a total time complexity of O(n + q).

[int(words[i][0] in vowels and words[i][-1] in vowels) for i in range(len(words))],

# For each query, calculate the number of words that start and end with a vowel

# the cumulative count at the end index (+1 because of the way the cumulative

results = [cumulative\_count[r + 1] - cumulative\_count[l] for l, r in queries]

# result = solution.vowel strings(["apple", "banana", "kiwi"], [[0, 1], [1, 2]])

# in the given range by subtracting the cumulative count at the start index from

# print(result) # Output would be the number of words with vowel start and end in the given ranges

2. The use of accumulate from the itertools module to create a prefix sum array of the boolean values obtained from the list comprehension.

3. Finally, answering the queries involves iterating over each query and performing constant-time subtraction for the prefix sums. If there are q

1. Storage of the boolean list comprehension which requires O(n) space. 2. The prefix sum list s which also needs O(n) space. 3. The space required for the output list which will store q integers, hence O(q) space.

The space complexity considerations include:

The total space complexity is thus the sum of the above, which is O(n + q).