1641. Count Sorted Vowel Strings

Medium Math Dynamic Programming Combinatorics

Problem Description

The goal is to find the number of strings of a given length `n` that are composed solely of vowels (`a`, `e`, `i`, `o`, `u`) and are lexicographically sorted. A lexicographically sorted string means that each character in the string is in non-decreasing alphabetical order. For example, the string "aei" is sorted but "aie" is not because `i` comes before `e` alphabetically. Given that the strings can only contain vowels, and they must be in sorted order, we have to count all the possible combinations that meet these criteria for a string of length `n`.

Intuition

To understand the solution, let's consider a simple case with `n = 1`. There are obviously 5 strings that can be formed, each with one character: `a`, `e`, `i`, `o`, or `u`. Now consider `n = 2`. We can add each vowel to each of the strings of length 1, but because our strings must remain sorted, every character we add can only be the same or come after the character in our existing string. So for 'a', we can add `a`, `e`, `i`, `o`, or `u`, but for 'e', we can only add `e`, `i`, `o`, or `u`, and so on. This results in a total of 15 strings.

Intuitively, you can see that the problem is related to the concept of combinations with repetitions allowed. The code provided uses a dynamic programming approach to build a solution for a string of length `n` based on the solutions for strings of length `n-1`. Here's how the implementation works:

1. We start by initializing a list `f` with length 5, filled with 1s. This list keeps track of the number of ways to end a string of length `n` with each vowel. Initially, for the case where `n = 1`, we have 1 way to end a string with each vowel.
2. For each additional character in our string (from 2 to `n`), we iterate through our list `f`, and for each position, we calculate the cumulative sum up to the current position. This cumulative sum represents the number of ways we can form a string of the current length that ends with the vowel corresponding to the current position.
3. Why does this work? The cumulative sum works because it adds all the ways we could form strings ending with vowels that can lexicographically precede the current vowel. This is exactly what we need because earlier vowels can always be followed by later vowels to maintain the sorted order.
4. After we have iterated through all the characters (`n - 1` times), we sum up our list `f` to get the total number of sorted vowel strings of length `n`.

The elegance of the solution lies in the way it builds from simpler subproblems, exploiting the fact that every subproblem is related to smaller instances of itself; hence, building up to the solution rather than trying to compute it from scratch.

Solution Approach

The solution uses dynamic programming, a common technique for solving problems by breaking them down into simpler subproblems. The Python code's structure demonstrates how we can smartly leverage the combinatorial nature of the problem with a bottom-up approach. Here's a step-by-step walkthrough of how the algorithm operates:

1. **Initialize the State**: A list `f` of length 5 is initialized with ones, `f = [1, 1, 1, 1, 1]`. This represents the number of ways to form a lexicographically sorted string with `n = 1` for each vowel. Essentially, these are the base cases.
2. **Iterate Over String Length**: We iterate from 2 up to `n` (since `n = 1` is our base case). For each new length, we are trying to compute the number of sorted vowel strings of that length.
3. **Update State with Cumulative Sums**: For each vowel indexed by `j`, we update `f[j]` with the sum of counts up to `j`. In other words, `s` is a running total which is added to each `f[j]`. This represents all the ways we can end a string with the vowel at `f[j]` while maintaining lexicographic order.
   - For example, if `f` currently equals `[1, 2, 3, 4, 5]` (for 'a', 'e', 'i', 'o', 'u' respectively), for the next length, 'a' can be followed by any vowel so it takes the sum of all possibilities `1+2+3+4+5`, 'e' cannot be followed by 'a', so it gets `2+3+4+5`, and so on.
4. **Final Summation**: Once all iterations for lengths are completed, we sum up all the numbers in `f` to get the total count of valid lexicographically sorted strings of length `n`.

This solution effectively applies the principles of counting sort by accumulating the number of valid previous options and using them to form new strings. By the end of the iterations, `sum(f)` gives us the desired output — the number of lexicographically sorted strings of length `n`.

Here's how the code encapsulates this logic:

```
class Solution:
    def countVowelStrings(self, n: int) -> int:
        f = [1] * 5  # Base case initialization
        for _ in range(n - 1):  # Iterating over string length
            s = 0  # Reset cumulative sum
            for i in range(5):  # Iterating over vowels
                s += f[i]  # Cumulative sum represents the combinatorial options
                f[i] = s  # Store the new count back in f
        return sum(f)  # Final result is the sum of the last state
```

The algorithm's time and space complexity are both O(N), where N is the length of the string, making it efficient even for larger strings as it only requires iterating and updating a 5-element list N-1 times.

Example Walkthrough

Let's illustrate the solution approach with an example where `n = 3`, i.e., we want to find the count of all lexicographically sorted strings that can be formed using vowels and have a length of 3.

1. **Initialize the State**: We start by setting `f = [1, 1, 1, 1, 1]`, which represents there is 1 way to create a string of length 1 ending with each of the vowels 'a', 'e', 'i', 'o', 'u'.
2. **Iterate Over String Length**:
   - For `n = 2`:
     - We start by initializing a cumulative sum `s = 0`.
     - We then update this sum in a loop for each element in `f`:
       - For 'a' (`f[0]`), the sum becomes `s = 0 + 1 = 1`; we update `f[0] = 1`.
       - For 'e' (`f[1]`), the sum becomes `s = 1 + 1 = 2`; we update `f[1] = 2`.
       - For 'i' (`f[2]`), the sum becomes `s = 2 + 1 = 3`; we update `f[2] = 3`.
       - For 'o' (`f[3]`), the sum becomes `s = 3 + 1 = 4`; we update `f[3] = 4`.
       - For 'u' (`f[4]`), the sum becomes `s = 4 + 1 = 5`; we update `f[4] = 5`.
     - Now, `f = [1, 2, 3, 4, 5]` after the first pass which accounts for strings of length 2.
   - For `n = 3`:
     - Again, we initialize the cumulative sum `s = 0`.
     - We repeat the loop, updating `f` cumulatively:
       - For 'a', 'a' becomes `s = 0 + 1 = 1`; 'a' can precede any vowels, so we keep `f[0] = 1`.
       - For 'e', 'e' becomes `s = 1 + 2 = 3`; 'e' can only be followed by 'e', 'i', 'o', 'u', so `f[1] = 3`.
       - For 'i', 'i' becomes `s = 3 + 3 = 6`; 'i' can only be followed by 'i', 'o', 'u', so `f[2] = 6`.
       - For 'o', 'o' becomes `s = 6 + 4 = 10`; 'o' can only be followed by 'o', 'u', so `f[3] = 10`.
       - For 'u', 'u' becomes `s = 10 + 5 = 15`; 'u' can only be followed by 'u', so `f[4] = 15`.
     - Our array `f` is now `[1, 3, 6, 10, 15]`.
3. **Final Summation**: After completing iteration for `n = 3`, we calculate the sum of all elements in `f` to obtain the total count of sorted vowel strings of length `n`:
   - `sum = 1 + 3 + 6 + 10 + 15 = 35`

Therefore, the number of lexicographically sorted strings of length 3 that can be made using the vowels 'a', 'e', 'i', 'o', 'u' is `35`. This example demonstrates how the dynamic programming approach simplifies the counting process by breaking it down into manageable steps, building upon the result for each previous length to gradually find the total count for a string of length `n`.

Solution Implementation

Python

```
class Solution:
    def countVowelStrings(self, n: int) -> int:
        # Initialize a list 'vowel count' with 5 elements all set to 1.
        # This represents the count for each vowel: a, e, i, o, u
        vowel_count = [1] * 5

        # We loop over 'n - 1' times because we already initialized the first set of counts.
        for _ in range(n - 1):
            # Initialize a running sum to build the counts for the current position.
            running_sum = 0
            # Loop through the counts for each vowel.
            for i in range(5):
                # Add the count for the current vowel to the running sum.
                running_sum += vowel_count[i]
                # Update the count for the current vowel with the running sum.
                # This step is based on the dynamic programming principle, where
                # the count of vowel strings ending with the current vowel is equal to
                # all vowel strings ending with any vowel up to the current one.
                vowel_count[i] = running_sum

        # Return the sum of all counts, which represents all possible vowel strings of length 'n'.
        return sum(vowel_count)
```

Java

```
import java.util.Arrays;  // Import the Arrays class for the sum method

class Solution {

    // This method counts the number of strings consisting of vowels that can be formed of length `n`.
    public int countVowelStrings(int n) {
        // An array to represent the counts of vowel strings for each vowel ending
        // f[0] for 'a', f[1] for 'e', f[2] for 'i', f[3] for 'o', f[4] for 'u'
        int[] vowelCounts = {1, 1, 1, 1, 1};

        // Loop over the length n times, first length is already initialized, so we start from 0 to n-2
        for (int i = 0; i < n - 1; ++i) {
            // A temporary variable to keep track of the cumulative sum
            int cumulativeSum = 0;
            // This loop calculates the new counts by adding up counts from all previous vowels
            for (int j = 0; j < 5; ++j) {
                // Update cumulative sum with the current vowel count
                cumulativeSum += vowelCounts[j];
                // Update the count for the current vowel with the new cumulative sum
                vowelCounts[j] = cumulativeSum;
            }
        }

        // Sum up all the counts and return the result
        return Arrays.stream(vowelCounts).sum();
    }
}
```

C++

```
#include <numeric> // Required for std::accumulate

class Solution {
public:
    int countVowelStrings(int n) {
        // Initialize an array with 5 elements corresponding to the 5 vowels
        // and assign them the base case value, which is 1 for each vowel.
        int vowelCounts[5] = {1, 1, 1, 1, 1};

        // Run the loop for 'n - 1' times starting from 0 since the base case
        // for the first element is already set.
        for (int i = 0; i < n - 1; ++i) {
            // Initialize the sum which holds the running number of strings
            // ending with the current vowel.
            int runningSum = 0;

            // Iterate through the vowelCounts to update the counts for each vowel.
            // To prevent the new updates for vowels as we progress through the arrays,
            // we use a running sum variable to keep track of the updated counts.
            for (int j = 0; j < 5; ++j) {
                // Add the current vowel's count to the running sum.
                runningSum += vowelCounts[j];

                // Update the count for the current vowel to be the running sum,
                // which is effectively the count of vowel strings ending with the
                // current vowel or before.
                vowelCounts[j] = runningSum;
            }
        }

        // Return the sum of all the vowel string counts which we have
        // accumulated in our vowelCounts array for 'n' positions.
        // std::accumulate performs a fold operation over the entire range
        // (first, last) and the initial sum value of 0.
        return std::accumulate(vowelCounts, vowelCounts + 5, 0);
    }
};
```

TypeScript

```
/**
 * Count the number of strings of length `n` composed of vowels (a, e, i, o, u)
 * that are lexicographically sorted.
 *
 * @param {number} n - The length of the vowel strings to count.
 * @returns {number} - The total count of lexicographically sorted vowel strings.
 */
function countVowelStrings(n: number): number {
    // Initialize an array with 5 elements corresponding to the counts for a, e, i, o, u.
    let vowelCounts: number[] = [1, 1, 1, 1, 1];

    // Loop for 'n - 1' times since we start the counts with the base case of length 1.
    for (let i = 0; i < n - 1; ++i) {
        // Initialize the running sum, which holds the cumulative counts.
        let runningSum = 0;

        // Update the counts for each position in the vowel string.
        for (let j = 0; j < 5; ++j) {
            // Update the running sum with the current vowel count.
            runningSum += vowelCounts[j];

            // Assign the running sum as the new vowel count for the current vowel.
            // This represents the total count of strings ending with this vowel or before.
            vowelCounts[j] = runningSum;
        }
    }

    // Calculate and return the total count of vowel strings of length `n`.
    // Use the reduce function to sum all elements in `vowelCounts`.
    return vowelCounts.reduce((acc, count) => acc + count, 0);
}

class Solution:
    def countVowelStrings(self, n: int) -> int:
        # Initialize a list 'vowel count' with 5 elements all set to 1.
        # This represents the count for each vowel: a, e, i, o, u
        vowel_count = [1] * 5

        # We loop over 'n - 1' times because we already initialized the first set of counts.
        for _ in range(n - 1):
            # Initialize a running sum to build the counts for the current position.
            running_sum = 0
            # Loop through the counts for each vowel.
            for i in range(5):
                # Add the count for the current vowel to the running sum.
                running_sum += vowel_count[i]
                # Update the count for the current vowel with the running sum.
                # This step is based on the dynamic programming principle, where
                # the count of vowel strings ending with the current vowel is equal to
                # all vowel strings ending with any vowel up to the current one.
                vowel_count[i] = running_sum

        # Return the sum of all counts, which represents all possible vowel strings of length 'n'.
        return sum(vowel_count)
```

Time and Space Complexity

The given Python code is used to count the number of strings of length `n` that consist only of vowels (`a`, `e`, `i`, `o`, `u`), where each string is lexicographically greater than the previous one (i.e., we can consider each vowel as a number, and these strings represent non-decreasing sequences).

Time Complexity

To analyze the time complexity, we observe the following steps in the code:

1. An initial list `f` is created with 5 elements, all of which are set to `1`. This represents the number of vowel strings of length `1`.
2. The outer loop runs `(n - 1)` times which is `O(n)` since `n` is the length of the string to be formed.
3. Inside the outer loop, there is an inner loop that iterates over the 5 vowels.
4. For each vowel, it accumulates the values from the previous counts in the list `f`, effectively computing the cumulative sum.

Since there are 5 vowels, the inner loop runs at constant time, `O(5)`, which simplifies to `O(1)`. The cumulative sum in the inner loop does not alter the time complexity.

Thus, combining the complexity of the outer and inner loop, the final time complexity comes out to be `O(n) * O(1) = O(n)`.

Space Complexity

The space complexity is determined by the storage utilized by the algorithm which remains constant irrespective of the input size:

1. A list `f` containing 5 elements is used to store the count of each type of vowel strings ending in a particular vowel, which takes up constant space `O(5)`.
2. The variable `s` is used to keep the cumulative sum within the inner loop. This is just an integer, which also uses constant space `O(1)`.

Therefore, the space complexity of the algorithm is `O(5) + O(1)`, which simplifies to `O(1)` indicating constant space complexity.