2255. Count Prefixes of a Given String

Easy Array String

Problem Description

You are provided with an array `words` containing a list of strings, and a single string `s`. Both the strings in `words` and the string `s` are made up exclusively of lowercase English letters. Your task is to find out how many of the provided strings in the `words` array are actually prefixes of the string `s`.

In the context of this problem, a prefix is defined as a substring that appears right at the beginning of another string. For example, "pre" is a prefix of "prefix". A substring is simply a sequence of characters that are found together in order, within a larger string.

The goal is to return a count of how many strings in `words` match the beginning of `s` exactly, character by character, from the first character onward.

Intuition

The intuition behind the solution is straightforward: You need to check every string in the `words` array and see if that string is a starting substring—or a prefix—of the given string `s`.

To arrive at the solution, you traverse through each word in the `words` array and use the function `startswith` to check if `s` begins with that word. The `startswith` function is handy as it does the job of comparing the characters of the potential prefix with the initial characters of the string `s`.

For each word in the `words` array, if `s` starts with that word, you consider it a match and include it in your count of prefix strings. By summing up the boolean values (True is treated as 1, and False is treated as 0 in Python) returned from `startswith` for each word, you get the total number of strings in the `words` array that are a prefix of `s`.

Solution Approach

The solution involves a very simple yet effective approach. It doesn't require complicated algorithms or data structures. Here, Python's built-in string method and list comprehension are used to achieve the goal in an efficient manner. The steps below break down the solution:

1. **Utilizing `startswith`:** The method `startswith` is used on the string `s` to check if it starts with a certain substring. This method returns `True` if `s` starts with the given substring, and `False` otherwise.
2. **List Comprehension for Iteration and Condition Check:** Instead of writing a loop to iterate over each word in `words`, list comprehension is used, which is a concise way to create lists based on existing lists. In this case, it's used to create a list of boolean values, where each boolean indicates whether the corresponding word in `words` is a prefix of `s`.
3. **Summation of Boolean Values:** In Python, the Boolean `True` has an integer value of `1`, and `False` has an integer value of `0`. Thus, by summing up the list created by list comprehension, it counts the number of `True` values, which represents the number of words that are prefixes of `s`.

Here's how the implementation looks:

```
class Solution:
    def countPrefixes(self, words: List[str], s: str) -> int:
        return sum(s.startswith(w) for w in words)
```

In the above implementation:

- `words: List[str]` is the input list of words to check.
- `s: str` is the string to compare against.
- The `countPrefixes` method returns an integer `int` which is the count of prefixes.
- The expression `s.startswith(w) for w in words` generates a generator having `True` or `False` for each word in `words`.
- The `sum(...)` function takes the generator and adds up all `True` values resulting in the count of words that are a prefix to `s`.

Despite the simplicity of this solution, it is very effective for this problem. No additional space is needed beyond the input, and each word is checked exactly once, resulting in a time complexity that is linear to the number of words times the average length of the words.

Example Walkthrough

Let's say we are given the following array of `words` and the string `s`:

```
words = ["a", "app", "appl"]
s = "apple"
```

We want to find out how many of these words are prefixes of the string `s`.

- The string `s` is "apple".
- The first word in our array is "a", which is indeed the first letter of "apple". So, "a" is a prefix of "apple".
- The second word is "app", which matches the first three letters of "apple". Therefore, "app" is also a prefix of "apple".
- The third word is "appl", which matches the first four letters of "apple". "appl" is a prefix as well.

The process for the solution is as follows:

1. **Utilizing `startswith`:** We use `startswith` to check if "apple" (`s`) starts with "a" (`words[0]`). It returns `True`.
2. We repeat this for "app" (`words[1]`) and "apple" starts with "app" as well, so it returns `True`.
3. Finally, we check "appl" (`words[2]`) and find that "apple" does start with "appl", hence it returns `True`.

Now, let's construct the list comprehension by evaluating `s.startswith(w)` for each `w` in `words`:

```
matches = [s.startswith(w) for w in words]
```

This list comprehension would result in:

```
matches = [True, True, True]
```

3. **Summation of Boolean Values:** We then sum up these boolean values:

```
count = sum(matches)
```

`count` would therefore be `3` since we have three `True` values.

In the actual implementation using the provided solution approach, it would look like this:

```
class Solution:
    def countPrefixes(self, words: List[str], s: str) -> int:
        return sum(s.startswith(w) for w in words)
```

When calling the `countPrefixes` method with our `words` and `s`:

```
solution = Solution()
result = solution.countPrefixes(words, "apple")
```

`result` would be equal to `3`, indicating that there are three prefixes of "apple" in the given list of `words`. This example illustrates the efficacy and simplicity of the solution when applied to a small set of data.

Solution Implementation

Python

```
from typing import List

class Solution:
    # Function to count the number of words that are prefixes of a given string
    def count_prefixes(self, words: List[str], s: str) -> int:
        # Initialize the count to 0
        count = 0

        # Iterate over each word in the list
        for word in words:
            # Check if the current word is a prefix of the string s
            if s.startswith(word):
                # If it is a prefix, increment the count
                count += 1

        # After the loop, return the total count
        return count

# Example of how to use the class:
# solution = Solution()
# result = solution.count_prefixes(["a", "b", "c", "a"], "ac")
# print(result)  # Should print the number of prefixes found in the string "ac"
```

Java

```
class Solution {
    // Method to count how many strings in the array are prefixes of the given string
    public int countPrefixes(String[] words, String s) {
        int count = 0;  // Initialize a counter to keep track of prefix matches

        // Iterate through each word in the array
        for (String word : words) {
            // Check if the string 's' starts with the current word
            if (s.startsWith(word)) {
                count++;  // If it does, increment the counter
            }
        }

        // Return the final count of prefix matches
        return count;
    }
}
```

C++

```
#include <vector>
#include <string>
using namespace std;

class Solution {
public:
    // Function to count the number of words in 'words' that are prefixes of the string 's'.
    // Params:
    // words - vector of strings that we will check against the string 's'
    // s - the string against which we are checking the prefixes
    // Returns the count of prefixes from 'words' that are found at the start of string 's'.
    int countPrefixes(vector<string>& words, string s) {
        int count = 0; // Initialize a counter to keep track of the prefixes
        // Iterate through each word in the vector 'words'
        for (const string& word : words) {
            // Increase the count if the string 's' starts with the current word
            // Note: starts_with is a standard C++20 string method
            count += s.starts_with(word);
        }
        // Return the final count of prefix occurrences
        return count;
    }
};
```

TypeScript

```
/**
 * Counts the number of strings in a given array that are prefixes of a particular string.
 *
 * @param {string[]} words - An array of words to check for prefix status.
 * @param {string} targetString - The string to check prefixes against.
 * @return {number} - The count of words that are prefixes of the target string.
 */
function countPrefixes(words: string[], targetString: string): number {
    // Filter the array of words, keeping only those that are prefixes of targetString
    const prefixWords = words.filter(word => targetString.startsWith(word));

    // Return the length of the filtered array, which represents the number of prefixes
    return prefixWords.length;
}
```

```
from typing import List

class Solution:
    # Function to count the number of words that are prefixes of a given string
    def count_prefixes(self, words: List[str], s: str) -> int:
        # Initialize the count to 0
        count = 0

        # Iterate over each word in the list
        for word in words:
            # Check if the current word is a prefix of the string s
            if s.startswith(word):
                # If it is a prefix, increment the count
                count += 1

        # After the loop, return the total count
        return count

# Example of how to use the class:
# solution = Solution()
# result = solution.count_prefixes(["a", "b", "c", "a"], "ac")
# print(result)  # Should print the number of prefixes found in the string "ac"
```

Time and Space Complexity

The time complexity of the given code is `O(m * n)`, where `m` is the number of words in the list `words` and `n` is the length of the string `s`. This is because for each word in `words`, the `startswith` method checks if the string `s` starts with that word, and this check can take up to `O(n)` time in the worst case for each word.

The space complexity of the code is `O(1)` as it does not allocate any additional space that is dependent on the size of the input—the only extra space used is for the sum operation, which is a constant space usage not affected by the input size.