2185. Counting Words With a Given Prefix

`Easy` `Array` `String`

Problem Description

The task is to count how many strings from the given array `words` have the string `pref` as their prefix. A prefix is defined as a substring that appears at the start of another string. For example, if `pref` is "ab", and you have a string "abcd" in `words`, it counts because "abcd" starts with "ab".

Intuition

The intuitive solution to this problem is straightforward. You simply iterate through each string in the array `words` and check if it starts with the string `pref`. The method `startswith()` in Python is perfect for this task as it does exactly what's needed—it returns `True` if the string object it's called on starts with the specified prefix, which in this case, is `pref`.

By using a list comprehension, we effectively generate a list of boolean values, `True` for strings that start with `pref` and `False` otherwise. Wrapping this list comprehension with Python's `sum()` function cleverly counts the number of `True` values in that list, because `True` is equivalent to `1` and `False` is equivalent to `0` when it comes to addition. Hence, we get the total count of strings with the given prefix in one elegant line of code.

Solution Approach

The solution is implemented in Python and makes use of a list comprehension and the built-in `startswith()` method for strings. The elegance of the solution lies in its simplicity and efficient use of the language's features.

Here are the steps taken in the algorithm:

1. Loop through each string `w` in the array `words` using a list comprehension. For each string, check if it starts with the prefix `pref`. This is done using the `startswith()` method, which is called on each `w`.

2. The `startswith()` method returns a boolean value `True` or `False`. This list comprehension creates a list of boolean values corresponding to each string in `words`.

3. The `sum()` function is then used to add up all the boolean values in the list. In Python, `True` is treated as `1` and `False` as `0` when summed. So, `sum()` essentially counts all the `True` values, which represent the strings that start with `pref`.

4. The result of the `sum()` function gives us the total number of strings in `words` that contain `pref` as a prefix.

The code for this solution is concise and can be written as:

```
1  class Solution:
2      def prefixCount(self, words: List[str], pref: str) -> int:
3          return sum(w.startswith(pref) for w in words)
```

This code fragment is the complete implementation of the solution. It defines a class `Solution` with a method `prefixCount`, which takes an array of strings `words` and a string `pref` as arguments and returns an integer. The `prefixCount` method only contains the above-mentioned list comprehension wrapped in the `sum()` function, which calculates and returns the count directly.

The solution does not require additional data structures and is a good example of Python's expressive capability to solve problems in a very readable and succinct manner.

Example Walkthrough

Let's illustrate the solution with a small example. Suppose we have an array of strings `words` that contains `["apple", "appreciate", "appetite", "banana", "application"]` and our prefix `pref` is `"app"`.

As we iterate through `words`:

- The first word `"apple"` starts with `"app"`. startswith() returns `True`.
- The second word `"appreciate"` also starts with `"app"`. startswith() returns `True`.
- The third word `"appetite"` starts with `"app"` as well. startswith() returns `True`.
- The fourth word `"banana"` does not start with `"app"`. startswith() returns `False`.
- The fifth word `"application"` starts with `"app"`. startswith() returns `True`.

Using the list comprehension `[w.startswith(pref) for w in words]`, we get a list of boolean values: `[True, True, True, False, True]`.

Passing this list to the `sum()` function adds up the `True` values (with each `True` being equivalent to `1`): `sum([True, True, True, False, True])` which equals `1 + 1 + 1 + 0 + 1`, giving us a total of `4`.

So, the total count of strings with the prefix `"app"` in the `words` array is `4`. This demonstrates the simplicity and elegance of the solution using Python's built-in functions.

Python Solution

```python
1  from typing import List  # Import List from typing module for type annotation
2
3  class Solution:
4      def prefix_count(self, words: List[str], pref: str) -> int:
5          # Count the number of words that start with the given prefix
6          # Args:
7          #     words: A list of strings representing the words
8          #     pref: A string representing the prefix to be matched
9          # Returns:
10         #     The count of words starting with the given prefix
11
12         # Use a generator expression to iterate over 'words' list
13         # The 'startswith()' method is used to check if the word starts with 'pref'
14         # 'sum()' function adds up how many times True appears (i.e., where the condition is met)
15         return sum(word.startswith(pref) for word in words)
16
```

Java Solution

```java
1  class Solution {
2      // Method to count how many strings in the words array have the prefix 'pref'
3      public int prefixCount(String[] words, String pref) {
4          int count = 0; // Initialize counter to track number of words with the prefix
5          // Iterate through each word in the array
6          for (String word : words) {
7              // Check if the current word starts with the given prefix
8              if (word.startsWith(pref)) {
9                  // Increment the count if the word has the prefix
10                 count++;
11             }
12         }
13         // Return the total count of words with the prefix
14         return count;
15     }
16 }
17
```

C++ Solution

```cpp
1  #include <vector>
2  #include <string>
3
4  class Solution {
5  public:
6      // Function to count the number of words in 'words' that start with the given prefix 'pref'
7      int prefixCount(vector<string>& words, const string& pref) {
8          int count = 0; // Initialize a variable to store the count of words with the prefix
9          for (const auto& word : words) { // Iterate over each word in the vector 'words'
10             if (word.find(pref) == 0) { // Check if 'pref' is a prefix of 'word'
11                 count++; // Increment the count if the word starts with the given prefix
12             }
13         }
14         return count; // Return the total count of words with the given prefix
15     }
16 };
17
```

Typescript Solution

```typescript
1  // Counts the number of words in the array that start with the given prefix.
2  // @param words — An array of strings to be checked against the prefix.
3  // @param prefix — The prefix string to match at the beginning of each word.
4  // @returns The count of words starting with the specified prefix.
5  function prefixCount(words: string[], prefix: string): number {
6      // Use Array.prototype.reduce to accumulate the count of words starting with the prefix.
7      // — The callback function checks if the curent word starts with the prefix.
8      // — If it does, increment the accumulator.
9      // — The initial value of the accumulator is set to 0.
10     return words.reduce((accumulator, currentWord) => {
11         accumulator += currentWord.startsWith(prefix) ? 1 : 0
12     }, 0);
13 }
14
```

Time and Space Complexity

The given Python function `prefixCount` iterates over each word in the list `words` and checks if it starts with the prefix `pref` using the `startswith` method. The computation complexity analysis is as follows:

Time Complexity

The time complexity of the function is $O(n * k)$, where n is the number of words in the list and k is the length of the prefix `pref`. This is because for each of the n words, the `startswith` method checks up to k characters to determine if the word starts with the prefix.

Space Complexity

The space complexity of the function is $O(1)$. The function uses a generator expression within the `sum` function, which calculates the result on-the-fly and does not require additional space proportional to the input size. The only additional memory used is for the counter in the `sum` function, which is a constant space requirement.