2114. Maximum Number of Words Found in Sentences

Easy Array String

Leetcode Link

Problem Description

The problem provides us with an array of strings where each string represents a sentence. A sentence is defined to be a list of words that are separated by a single space and does not have leading or trailing spaces. Our task is to determine the maximum number of words that appear in any given sentence in the array.

Intuition

The key to solving this problem lies in understanding that within a properly spaced sentence, the number of spaces between words is exactly one less than the number of words. For instance, the sentence "I have a dream" contains three spaces but four words. Keeping this logic in mind, we can simply count the spaces in each sentence to find out how many words it has, and then add one to that count to account for the first word which precedes any spaces.

Here's the step by step approach:

1. Iterate over each sentence in our array: Each sentence is checked individually.
2. Count the spaces in the sentence: We go through a sentence counting every space (' ') encountered.
3. Since the number of spaces is always one less than the number of words, we add 1 to the count of spaces to get the number of words in the sentence.
4. Determine the maximum word count: Of all the word counts obtained for each sentence, we find and keep track of the largest one, which is the maximum number of words in a sentence.
5. Return the maximum word count: The highest number we found from the step above represents our answer.

The code `1 + max(s.count(' ') for s in sentences)` cleverly applies this approach using a generator expression, which avoids the need to create an intermediate list, hence making it a more memory-efficient solution. It loops over each sentence `s` counting spaces `' '` and adds 1 to each result to account for the number of words. The `max` function then pulls the largest number from these results, effectively finding the sentence with the most words.

Solution Approach

The solution provided is simple and efficient. It is implemented in Python, using what's known as a list comprehension combined with the `max` function. Here's the breakdown of the steps involved in the implementation and the concepts used:

1. **List Comprehension**: Instead of using a loop to iterate over each element, the solution uses a list comprehension, which is a concise way to create lists in Python. However, to further optimize it, a generator expression is used (which is like a list comprehension, but without creating a list in memory).

2. **String's Count Method**: The `count` method is called on each string (sentence) within the generator expression. This Python string method counts the number of occurrences of a substring (in this case, the space character `" "`) within the string.

3. **Adding Words**: As each sentence is guaranteed to have one more word than the number of spaces, the expression `s.count(' ')` is increased by 1. This gives us the total word count for each sentence.

4. **Max Function**: Python's built-in `max` function is then applied to the generator expression. This function iterates through all values produced by the generator expression and returns the largest one, which corresponds to the sentence with the most words.

The implementation does not explicitly use any complex data structures, algorithms, or design patterns; it is a straightforward application of built-in Python functions and syntax to solve the problem efficiently. The key to its efficiency lies in the use of a generator expression which ensures that only one item is processed at a time, thereby conserving memory, and the fact that `max` directly consumes these values to find the maximum.

The complete line of code `return 1 + max(s.count(' ') for s in sentences)` handles the problem at hand within a single return statement, showcasing the power of Python's expressive syntax and its standard library functions.

Example Walkthrough

Let's consider a small example to illustrate the solution approach:

Suppose we have an array of strings `sentences` representing various sentences as follows:

```
sentences = ["Alice loves Bob", "Keep calm and code on", "Life is simple"]
```

We want to find the maximum number of words found in any sentence within this array.

According to our solution approach, we loop through each sentence and count the spaces, then add one to that count to get the total number of words. Let's walk this through step by step:

1. Starting with the first sentence, `"Alice loves Bob"`, we count the spaces within this sentence. There are two spaces, which suggests there are three words in this sentence (`2 + 1 = 3` words).

2. Moving on to the second sentence, `"Keep calm and code on"`, we count the spaces again. We find four spaces, indicating there are five words in this sentence (`4 + 1 = 5` words).

3. Lastly, for the third sentence `"Life is simple"`, we count the spaces and we find two spaces, which means there are three words in this sentence (`2 + 1 = 3` words).

Now, we compare the word counts from each sentence: `3, 5, 3`. The maximum word count among these is `5`, which is from the second sentence.

The single line in the solution approach:

```
return 1 + max(s.count(' ') for s in sentences)
```

This line effectively replicates what we did above in the example, but it does so using a generator expression, `s.count(' ') for s in sentences`, which counts the spaces for each sentence `s` and adds `1` to each. The `max` function is then called to return the maximum value generated, which corresponds to the maximum number of words in any sentence.

Therefore, when the code is run on our example array, it will return `5`, which is the maximum number of words in the sentence "Keep calm and code on".

Python Solution

```python
class Solution:
    def mostWordsFound(self, sentences):
        """
        Function to determine the maximum number of words in any sentence from the list of sentences.

        :param sentences: List of strings where each string represents a sentence.
        :return: The maximum number of words found in any sentence.
        """

        # We initialize the maximum word count to 0.
        max_word_count = 0

        # Iterate through each sentence in the list.
        for sentence in sentences:
            # Count the number of spaces in the sentence to determine the number of words.
            # Since the number of words is one more than the number of spaces, we add 1.
            word_count = 1 + sentence.count(' ')

            # Update the maximum word count if the current sentence has more words.
            max_word_count = max(max_word_count, word_count)

        # Return the maximum word count found.
        return max_word_count

```

Java Solution

```java
class Solution {
    public int mostWordsFound(String[] sentences) {
        int maxWords = 0; // This variable will hold the maximum number of words found.

        // Iterate over each sentence in the sentences array.
        for (String sentence : sentences) {
            int wordCount = 1; // Initialize word count to 1 because each sentence has at least one word.

            // Iterate over each character in the sentence.
            for (int i = 0; i < sentence.length(); ++i) {
                // Check for spaces, as each space indicates an additional word.
                if (sentence.charAt(i) == ' ') {
                    wordCount++; // Increment the word count.
                }
            }

            // Update maxWords with the larger value between current maxWords and wordCount.
            maxWords = Math.max(maxWords, wordCount);
        }

        return maxWords; // Return the maximum number of words found in any sentence.
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <string>
#include <algorithm> // Include algorithm header for count

class Solution {
public:
    // Function to find the maximum number of words in any given sentence.
    int mostWordsFound(vector<string>& sentences) {
        int maxWords = 0; // This will hold the maximum number of words found

        // Iterate through each sentence in the vector of sentences
        for (const auto& sentence : sentences) {
            // Count the number of spaces in the sentence, add 1 for the number of words
            // (assuming each word is separated by a single space and there is no trailing space)
            int wordCount = 1 + std::count(sentence.begin(), sentence.end(), ' ');

            // Update maxWords if the current sentence's word count is greater
            maxWords = std::max(maxWords, wordCount);
        }

        // Return the maximum word count found in any sentence
        return maxWords;
    }
};

```

Typescript Solution

```typescript
// Function to find the maximum number of words in any given sentence from the array.
// It takes an array of sentences and returns the maximum number of words found.
function mostWordsFound(sentences: string[]): number {
    // Use the `reduce` function to iterate over each sentence to find the one with
    // the most words by comparing the current maximum with the number of words in each sentence.
    return sentences.reduce((maxWordCount, currentSentence) => {
        // Split the current sentence by spaces to get an array of words,
        // then count the number of words in the current sentence.
        // We add 1 because the number of spaces is one less than the number of words.
        const wordCount = currentSentence.split(' ').length;

        // Compare and return the greater number between the current maxWordCount and the wordCount of the current sentence.
        return Math.max(maxWordCount, wordCount);
    }, 0); // Initialize the maxWordCount with 0.
}

```

Time and Space Complexity

Time Complexity

The given code snippet iterates over the list of sentences exactly once and performs the count operation for each sentence that effectively counts the number of spaces in that sentence. This count operation has a time complexity of `O(n)` where `n` is the length of the sentence. If there are `m` sentences and if `k` is the length of the longest sentence, the overall time complexity of the code will be `O(m * k)` since it has to count spaces for each sentence and the maximum count operation will take `O(k)` in the worst case when the sentence has the maximum length.

Space Complexity

The space complexity of the given code is `O(1)`. This is because the code only uses a constant amount of additional space. The additional space is used for the accumulation of the maximum count value. The input space (the list of sentences) is not considered in the space complexity analysis as it is the input to the function and is not part of the space used by the algorithm itself.