2284. Sender With Largest Word Count

Medium Array Hash Table String Counting Leetcode Link

Problem Description

In the given problem, we are working with a log of chat messages. We have two arrays: `messages` and `senders`. Each element of the `messages` array is a string representing a single chat message, and the corresponding element in the `senders` array is the name of the person who sent that message. It's important to note that these messages are well-formed, meaning they do not have any leading or trailing spaces and words in a message are separated by a single space.

The task is to determine which sender has the highest *word count* across all their messages. The word count for a sender is simply the total count of all words that sender has sent, considering all messages. If it turns out that multiple senders have the same highest word count, then we must return the sender whose name is the lexicographically largest among them. Here, lexicographical order refers to how words are sorted in a dictionary, with the catch that uppercase letters are considered to precede lowercase ones, thus `'A' < 'a'`.

In essence, we are asked to count words for each sender, compare the totals, and identify the sender with either the highest word count or, in case of a tie, the "largest" name.

Intuition

To approach this problem, we should think about how to effectively track the word count for each sender. A natural choice for this task is to use a hashmap (or a dictionary in Python), where the keys are the senders' names, and the values are their respective word counts.

Since we are provided pairs of messages and senders, we can iterate through these pairs and for each pair:

1. Count the words in the message. Since words are separated by spaces, we can simply count how many spaces are in the message and add one to that to get the word count.
2. Update the word count for the sender in the hashmap. If the sender isn't in the hashmap yet, we initialize their word count with the current count; otherwise, we add to the existing count.

After processing all pairs, we'll have a complete hashmap of senders and their total word counts.

The next step is to find the sender with the largest word count while keeping in mind the rules for breaking ties. To do this, we can iterate over the hashmap entries and keep track of the sender with the highest word count seen so far. If we find a sender with the same word count as our current highest, we check the lexicographical order between the names to decide if we should update our answer to this sender. Ultimately, we will arrive at the final answer after checking all the entries.

Solution Approach

The implemented solution uses Python's `Counter` class from the `collections` module, which is a specialized dictionary designed for counting hashable objects. It's an apt choice for maintaining the word counts for different senders in our problem.

Here's the step-by-step breakdown of the implementation:

1. Initialize a `Counter` object `cnt`. This will hold our senders as keys and their word counts as values.

2. Iterate over the zipped `messages` and `senders` arrays. The `zip` function is used to create pairs of message-sender which allows us to process them together.

3. In the loop, increment the word count for each `sender` in the `Counter` object by the number of words in their corresponding `msg`. Since words in a message are separated by spaces, `msg.count(' ') + 1` gives us the number of words in the message.

4. Create a variable `ans` to store the sender with the current largest word count, initialized to an empty string.

5. Iterate over the items in the `Counter` object to find the sender with the largest word count. For each sender, two conditions are checked:

   - If the current word count (`v`) is greater than the word count of the sender stored in `ans`, or
   - If the current word count (`v`) is equal to the word count of the sender stored in `ans`, but the name of the current sender comes later lexicographically (`ans < sender`),

   then we update `ans` to the current `sender`.

6. After examining all senders, `ans` contains the name of the sender with the largest word count. If there are multiple senders with the same word count, `ans` will be the lexicographically largest one among them, due to the way we perform the check in the previous step.

7. Finally, we return `ans`.

The clever use of the `Counter` class along with the `zip` function allows for a clean and efficient solution. The overall time complexity is O(N) where N represents the total number of words across all messages because we iterate through every word at least once. Handling the lexicographical comparison during the iteration over the `Counter` items ensures that we only need one pass to find the sender who satisfies both conditions in the problem statement.

Example Walkthrough

Let's use a small example to illustrate the solution approach with sample `messages` and `senders`.

Suppose we have the following inputs:

`messages` = ["Hello world", "Hi", "Good morning", "Good night"] `senders` = ["Alice", "Alice", "Bob", "Bob"]

We're tasked with determining who has the highest word count of all the messages they sent. Following the solution approach:

1. Initialize a `Counter` object `cnt`.
2. Zip `messages` and `senders` and iterate over them.
3. For each message-sender pair, use `msg.count(' ') + 1` to count the words and update the sender's total count in `cnt`.
4. Initialize a variable `ans` to store the sender with the highest word count so far.

We start by zipping and iterating over the pairs:

- **First iteration**: The message is "Hello world", and the sender is Alice. The word count for this message is 2 (`"Hello world".count(' ') + 1`). Update Alice's count in `cnt`: `cnt[Alice] = 2`.

Current `cnt` status: `{"Alice": 2}`

- **Second iteration**: The message is "Hi", and the sender is also Alice. The word count for this message is 1 (`"Hi".count(' ') + 1`). Update Alice's count in `cnt`: `cnt[Alice] = 2 + 1 = 3`.

Current `cnt` status: `{"Alice": 3}`

- **Third iteration**: The message is "Good morning", and the sender is Bob. The word count is 2 (`"Good morning".count(' ') + 1`). Update Bob's count in `cnt`: `cnt[Bob] = 2`.

Current `cnt` status: `{"Alice": 3, "Bob": 2}`

- **Fourth iteration**: The message is "Good night", and the sender is Bob again. The word count is 2 (`"Good night".count(' ') + 1`). Update Bob's count in `cnt`: `cnt[Bob] = 2 + 2 = 4`.

Final `cnt` status: `{"Alice": 3, "Bob": 4}`

5. Iterate over `cnt` to find the sender with the largest word count. We compare values and update `ans` accordingly:

- Since `Bob` has a higher word count (4) than `Alice` (3), we set `ans = "Bob"`.

6. After checking all the entries in `cnt`, we find that `Bob` has the largest word count.

7. Return `ans`, which is "Bob".

Through this example, we have walked through the steps outlined in the solution approach, leading us to conclude that Bob is the sender with the highest word count from the given messages.

Python Solution

```python
from collections import Counter

class Solution:
    def largestWordCount(self, messages: List[str], senders: List[str]) -> str:
        # Create a counter for tracking number of words sent by each sender
        word_count_by_sender = Counter()

        # Iterate over the messages and their corresponding senders simultaneously
        for message, sender in zip(messages, senders):
            # Count the number of words in the message and update the sender's count.
            # Adding 1 because the number of words is one more than the number of spaces.
            word_count_by_sender[sender] += message.count(' ') + 1

        # Initialize a variable to keep track of the sender with the highest word count
        top_sender = ''

        # Iterate over the senders in the counter to find the one with the highest word count
        for sender, word_count in word_count_by_sender.items():
            # Compare the current highest word count to this sender's count,
            # or check alphabetical order if there's a tie on word count
            if word_count_by_sender[top_sender] < word_count or \
               (word_count_by_sender[top_sender] == word_count and top_sender < sender):
                # Update the sender with the largest word count or lexicographically smaller sender on tie
                top_sender = sender

        # Return the sender with the largest total word count
        return top_sender

```

Java Solution

```java
class Solution {
    public String largestWordCount(String[] messages, String[] senders) {
        // Create a map to store the word count for each sender
        Map<String, Integer> wordCountBySender = new HashMap<>();
        int senderCount = senders.length;

        // Iterate over messages to count words and aggregate by sender
        for (int i = 0; i < senderCount; ++i) {
            // Start word count from 1 since each message has at least one word
            int wordCount = 1;
            // Count words in the message (each space signifies a new word)
            for (int j = 0; j < messages[i].length(); ++j) {
                if (messages[i].charAt(j) == ' ') {
                    ++wordCount;
                }
            }
            // Merge the word count with the existing count in the map for the sender
            wordCountBySender.merge(senders[i], wordCount, Integer::sum);
        }

        // Initial sender to compare with others using first sender's name
        String senderWithMaxWords = senders[0];
        // Iterate through the map to find the sender with the highest word count
        for (var entry : wordCountBySender.entrySet()) {
            String currentSender = entry.getKey();
            // Compare word counts and sender names to find the sender with the maximum words
            if (wordCountBySender.get(senderWithMaxWords) < entry.getValue()
                || (wordCountBySender.get(senderWithMaxWords).equals(entry.getValue())
                && senderWithMaxWords.compareTo(currentSender) < 0)) {
                senderWithMaxWords = currentSender;
            }
        }
        // Return the sender who has the maximum count of words
        return senderWithMaxWords;
    }
}

```

C++ Solution

```cpp
#include <vector>
#include <string>
#include <unordered_map>
#include <algorithm>

class Solution {
public:
    // Function to find the sender with the largest word count in the sent messages.
    // If multiple senders have the same word count, the sender with the lexicographically largest name is returned.
    string largestWordCount(vector<string>& messages, vector<string>& senders) {
        unordered_map<string, int> wordCountBySender; // Create a map to store the word counts for each sender.
        int totalSenders = senders.size(); // Total number of senders.

        // Calculate the word count for each message and aggregate it by sender.
        for (int i = 0; i < totalSenders; ++i) {
            // Count the words in the current message. Words are separated by spaces.
            int wordCount = count(messages[i].begin(), messages[i].end(), ' ') + 1;
            wordCountBySender[senders[i]] += wordCount; // Update the sender's total word count.
        }

        string largestWordCountSender = senders[0]; // Initialize with the first sender.

        // Find the sender with the highest word count or lexicographically largest name on a tie.
        for (const auto& [sender, wordCount] : wordCountBySender) {
            if (wordCountBySender[largestWordCountSender] < wordCount || // Check for a higher word count.
                (wordCountBySender[largestWordCountSender] == wordCount && largestWordCountSender < sender)) { // Check for a lexicog
                largestWordCountSender = sender;
            }
        }

        return largestWordCountSender; // Return the sender with the largest word count.
    }
};

```

Typescript Solution

```typescript
export const messageAnalysis = {
  // Function to count the number of words in a message.
  countWordsInMessage(message: string): number {
    return message.split(' ').length; // Words are separated by spaces.
  },

  // Function to find the sender with the largest word count in the sent messages.
  // If multiple senders have the same word count, the sender with the lexicographically largest name is returned.
  largestWordCount(messages: string[], senders: string[]): string {
    const wordCountBySender: { [sender: string]: number } = {}; // Map to store the word counts for each sender.

    // Calculate the word count for each message and aggregate it by sender.
    messages.forEach((message, index) => {
      const wordCount = this.countWordsInMessage(message); // Get the word count for the current message.
      const sender = senders[index]; // Corresponding sender for the message.

      if (wordCountBySender[sender]) {
        wordCountBySender[sender] += wordCount; // Update the sender's total word count.
      } else {
        wordCountBySender[sender] = wordCount; // Initialize the sender's word count.
      }
    });

    let largestWordCountSender = senders[0]; // Initialize with the first sender.

    // Find the sender with the highest word count or lexicographically largest name on a tie.
    Object.keys(wordCountBySender).forEach(sender => {
      const wordCount = wordCountBySender[sender];
      const isWordCountHigher =
        wordCountBySender[largestWordCountSender] < wordCount;
      const isSenderNameGreater =
        wordCountBySender[largestWordCountSender] === wordCount &&
        largestWordCountSender < sender;

      if (isWordCountHigher || isSenderNameGreater) {
        largestWordCountSender = sender;
      }
    });

    return largestWordCountSender; // Return the sender with the largest word count.
  }
};

```

Time and Space Complexity

Time Complexity

The given code performs the following operations:

- It zips and iterates through `messages` and `senders`, which takes `O(N)` time, where `N` is the number of messages/senders.
- Within the iteration, for each message, it counts the number of spaces and adds one to get the number of words in the message. If we assume `M` is the maximum length of a message, this operation takes `O(M)` for each message. So, for all messages, it is `O(N*M)`.
- After populating the counter, it iterates over each unique sender to find the sender with the highest word count or the lexicographically greatest sender in case of a tie. Let `S` be the number of unique senders, then this operation takes `O(S)` time. Note that in the worst case, `S` can be equal to `N` if all messages are from different senders.

To summarize, the overall time complexity is `O(N*M + S)`, which simplifies to `O(N*M)` since `M` can vary independently of `N` and `S`.

Space Complexity

The space complexity consists of the following:

- The counter hash map `cnt` that may at most contain `S` unique senders, where `S` can be at most `N`. Therefore, it is `O(S)`. In the worst case, this is `O(N)`.
- The temporary variables used for iteration and comparison use constant space, which is `O(1)`.

Combining these, the overall space complexity is `O(S)` which, in the worst case, simplifies to `O(N)`.