2284. Sender With Largest Word Count

```
Medium Array
               Hash Table
                           String
                                   Counting
```

**Problem Description** 

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

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

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

to that to get the word count.

allows us to process them together.

then we update ans to the current sender.

Solution Approach

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

count; otherwise, we add to the existing count.

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

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

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

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:

- Create a variable ans to store the sender with the current largest word count, initialized to an empty string.
- Iterate over the items in the Counter object to find the sender with the largest word count. For each sender, two conditions are checked:

Iterate over the zipped messages and senders arrays. The zip function is used to create pairs of message-sender which

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.

• If the current word count (v) is greater than the word count of the sender stored in lans, or o 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),</pre>

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, and will be the lexicographically largest one among them, due to the way we perform the check in the previous step.

pass to find the sender who satisfies both conditions in the problem statement.

4. Initialize a variable ans to store the sender with the highest word count so far.

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

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

3. For each message-sender pair, use msg.count(' ') + 1 to count the words and update the sender's total count in cnt.

- 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
- Suppose we have the following inputs:

1. Initialize a Counter object cnt.

2. Zip messages and senders and iterate over them.

**Example Walkthrough** 

Finally, we return ans.

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.

• 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

• 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

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

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

Current cnt status: {"Alice": 2}

count in cnt: cnt[Alice] = 2 + 1 = 3.

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

- 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.
- 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".

Solution Implementation

top\_sender = ''

return top sender

int senderCount = senders.length;

**Python** 

Java

C++

public:

#include <vector>

#include <string>

class Solution {

#include <algorithm>

#include <unordered map>

class Solution {

count in cnt: cnt[Bob] = 2 + 2 = 4.

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

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

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

from collections import Counter class Solution:

for message, sender in zip(messages, senders):

word\_count\_by\_sender = Counter()

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[sender] += message.count(' ') + 1

for sender, word count in word count by sender.items():

# Return the sender with the largest total word count

public String largestWordCount(String[] messages, String[] senders) {

// Function to find the sender with the largest word count in the sent messages.

// Calculate the word count for each message and aggregate it by sender.

// Count the words in the current message. Words are separated by spaces.

wordCountBySender[senders[i]] += wordCount; // Update the sender's total word count.

// Find the sender with the highest word count or lexicographically largest name on a tie.

int wordCount = count(messages[i].begin(), messages[i].end(), ' ') + 1;

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

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

string largestWordCount(vector<string>& messages, vector<string>& senders) {

int totalSenders = senders.size(); // Total number of senders.

for (const auto& [sender, wordCount] : wordCountBySender) {

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[sender] += message.count(' ') + 1

if word count by sender[top sender] < word count or \

for sender, word count in word count by sender.items():

# Return the sender with the largest total word count

# Iterate over the messages and their corresponding senders simultaneously

# Compare the current highest word count to this sender's count,

# or check alphabetical order if there's a tie on word count

# Count the number of words in the message and update the sender's count.

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

# Adding 1 because the number of words is one more than the number of spaces.

# Iterate over the senders in the counter to find the one with the highest word count

(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

largestWordCountSender = sender;

// Function to count the number of words in a message.

for (int i = 0; i < totalSenders: ++i) {</pre>

// Create a map to store the word count for each sender

Map<String. Integer> wordCountBySender = new HashMap<>();

# Iterate over the messages and their corresponding senders simultaneously

# Compare the current highest word count to this sender's count,

# Count the number of words in the message and update the sender's count.

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

# Adding 1 because the number of words is one more than the number of spaces.

# Iterate over the senders in the counter to find the one with the highest word count

sender with the highest word count from the given messages.

# 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

```
// Iterate over messages to count words and aggregate by sender
for (int i = 0; i < senderCount; ++i) {</pre>
    // 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) {</pre>
        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()</pre>
        || (wordCountBySender.get(senderWithMaxWords).equals(entry.getValue())
        && senderWithMaxWords.compareTo(currentSender) < 0)) {
        senderWithMaxWords = currentSender;
// Return the sender who has the maximum count of words
return senderWithMaxWords;
```

// If multiple senders have the same word count, the sender with the lexicographically largest name is returned.

unordered map<string, int> wordCountBySender; // Create a map to store the word counts for each sender.

if (wordCountBySender[largestWordCountSender] < wordCount || // Check for a higher word count.</pre>

(wordCountBySender[largestWordCountSender] == wordCount && largestWordCountSender < sender)) { // Check for a lexicog

**}**;

**TypeScript** 

export const messageAnalysis = {

```
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;</pre>
     const isSenderNameGreater =
       wordCountBySender[largestWordCountSender] === wordCount &&
        largestWordCountSender < sender;</pre>
     if (isWordCountHigher || isSenderNameGreater) {
        largestWordCountSender = sender;
   });
   return largestWordCountSender; // Return the sender with the largest word count.
};
from collections import Counter
```

## **Time and Space Complexity Time Complexity**

**Space Complexity** 

return top\_sender

top\_sender = ''

class Solution:

word\_count\_by\_sender = Counter()

top\_sender = sender

for message. sender in zip(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).

The given code performs the following operations:

• 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 5 be the number of unique senders, then this operation takes O(5) time. Note that in the worst case, 5 can be equal to N if all messages are from different senders.

• It zips and iterates through messages and senders, which takes O(N) time, where N is the number of messages/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.
- 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 0(1).

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