Problem 2284: Sender With Largest Word Count

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You have a chat log of

n

messages. You are given two string arrays

messages

and

senders

where

messages[i]

is a

message

sent by

senders[i]

.

A

message

is list of

words

that are separated by a single space with no leading or trailing spaces. The

word count

of a sender is the total number of

words

sent by the sender. Note that a sender may send more than one message.

Return

the sender with the

largest

word count

. If there is more than one sender with the largest word count, return

the one with the

lexicographically largest

name

.

Note:

Uppercase letters come before lowercase letters in lexicographical order.

"Alice"

and

"alice"

are distinct.

Example 1:

Input:

messages = ["Hello userTwooo","Hi userThree","Wonderful day Alice","Nice day userThree"], senders = ["Alice","userTwo","userThree","Alice"]

Output:

"Alice"

Explanation:

Alice sends a total of 2 + 3 = 5 words. userTwo sends a total of 2 words. userThree sends a total of 3 words. Since Alice has the largest word count, we return "Alice".

Example 2:

Input:

messages = ["How is leetcode for everyone","Leetcode is useful for practice"], senders = ["Bob","Charlie"]

Output:

"Charlie"

Explanation:

Bob sends a total of 5 words. Charlie sends a total of 5 words. Since there is a tie for the largest word count, we return the sender with the lexicographically larger name, Charlie.

Constraints:

n == messages.length == senders.length

1 <= n <= 10

4

1 <= messages[i].length <= 100

1 <= senders[i].length <= 10

messages[i]

consists of uppercase and lowercase English letters and

' '

.

All the words in

messages[i]

are separated by

a single space

.

messages[i]

does not have leading or trailing spaces.

senders[i]

consists of uppercase and lowercase English letters only.

Code Snippets

C++:

```
class Solution {
public:
string largestWordCount(vector<string>& messages, vector<string>& senders) {

}
};
```

Java:

```
class Solution {
public String largestWordCount(String[] messages, String[] senders) {

}
}
```

Python3:

```
class Solution:
def largestWordCount(self, messages: List[str], senders: List[str]) -> str:
```

Python:

```
class Solution(object):
def largestWordCount(self, messages, senders):
"""
:type messages: List[str]
:type senders: List[str]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string[]} messages
* @param {string[]} senders
* @return {string}
*/
var largestWordCount = function(messages, senders) {
};
```

TypeScript:

```
function largestWordCount(messages: string[], senders: string[]): string {

};
```

C#:

```
public class Solution {
public string LargestWordCount(string[] messages, string[] senders) {

}
}
```

C:

```
char* largestWordCount(char** messages, int messagesSize, char** senders, int
sendersSize) {

}
```

Go:

```
func largestWordCount(messages []string, senders []string) string {

}
```

Kotlin:

```
class Solution {
fun largestWordCount(messages: Array<String>, senders: Array<String>): String
{

}
}
```

Swift:

```
class Solution {
func largestWordCount(_ messages: [String], _ senders: [String]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn largest_word_count(messages: Vec<String>, senders: Vec<String>) ->
String {

}
}
```

Ruby:

```
# @param {String[]} messages
# @param {String[]} senders
# @return {String}
def largest_word_count(messages, senders)

end
```

PHP:

```
class Solution {

/**
* @param String[] $messages
* @param String[] $senders
* @return String
*/
function largestWordCount($messages, $senders) {

}
}
```

Dart:

```
class Solution {
String largestWordCount(List<String> messages, List<String> senders) {


}
}
```

Scala:

```
object Solution {
def largestWordCount(messages: Array[String], senders: Array[String]): String
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec largest_word_count(messages :: [String.t], senders :: [String.t]) ::
String.t
def largest_word_count(messages, senders) do

end
end
```

Erlang:

```
-spec largest_word_count(Messages :: [unicode:unicode_binary()], Senders ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
largest_word_count(Messages, Senders) ->
.
```

Racket:

```
(define/contract (largest-word-count messages senders)
(-> (listof string?) (listof string?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sender With Largest Word Count
* Difficulty: Medium
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string largestWordCount(vector<string>& messages, vector<string>& senders) {

}
};
```

Java Solution:

```
/**
* Problem: Sender With Largest Word Count
* Difficulty: Medium
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Sender With Largest Word Count
Difficulty: Medium
Tags: array, string, graph, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def largestWordCount(self, messages: List[str], senders: List[str]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def largestWordCount(self, messages, senders):
"""
:type messages: List[str]
:type senders: List[str]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Sender With Largest Word Count
* Difficulty: Medium
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[]} messages
* @param {string[]} senders
* @return {string}
*/
var largestWordCount = function(messages, senders) {

};
```

TypeScript Solution:

```
/**
* Problem: Sender With Largest Word Count
* Difficulty: Medium
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function largestWordCount(messages: string[], senders: string[]): string {

};
```

C# Solution:

```
/*
* Problem: Sender With Largest Word Count
* Difficulty: Medium
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string LargestWordCount(string[] messages, string[] senders) {

}
}
```

C Solution:

```
/*
* Problem: Sender With Largest Word Count
* Difficulty: Medium
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* largestWordCount(char** messages, int messagesSize, char** senders, int
sendersSize) {

}
```

Go Solution:

```
// Problem: Sender With Largest Word Count
// Difficulty: Medium
// Tags: array, string, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func largestWordCount(messages []string, senders []string) string {

}
```

Kotlin Solution:

```
class Solution {
fun largestWordCount(messages: Array<String>, senders: Array<String>): String
{

}
}
```

Swift Solution:

```
class Solution {
func largestWordCount(_ messages: [String], _ senders: [String]) -> String {

}
}
```

Rust Solution:

```
// Problem: Sender With Largest Word Count
// Difficulty: Medium
// Tags: array, string, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn largest_word_count(messages: Vec<String>, senders: Vec<String>) ->
String {

}
}
```

Ruby Solution:

```
# @param {String[]} messages
# @param {String[]} senders
# @return {String}
def largest_word_count(messages, senders)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $messages
* @param String[] $senders
* @return String
*/
function largestWordCount($messages, $senders) {

}
}
```

Dart Solution:

```
class Solution {
String largestWordCount(List<String> messages, List<String> senders) {
}
}
```

Scala Solution:

```scala
object Solution {
def largestWordCount(messages: Array[String], senders: Array[String]): String
= {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec largest_word_count(messages :: [String.t], senders :: [String.t]) ::
String.t
def largest_word_count(messages, senders) do

end
end
```

Erlang Solution:

```erlang
-spec largest_word_count(Messages :: [unicode:unicode_binary()], Senders ::
[unicode:unicode_binary()]) -> unicode:unicode_binary().
largest_word_count(Messages, Senders) ->
.
```

Racket Solution:

```racket
(define/contract (largest-word-count messages senders)
(-> (listof string?) (listof string?) string?)
)
```