Problem 3527: Find the Most Common Response

Problem Information

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

Problem Description

You are given a 2D string array

responses

where each

responses[i]

is an array of strings representing survey responses from the

i

th

day.

Return the

most common

response across all days after removing

duplicate

responses within each

responses[i]

. If there is a tie, return the

lexicographically smallest

response.

Example 1:

Input:

responses = [["good","ok","good","ok"],["ok","bad","good","ok","ok"],["good"],["bad"]]

Output:

"good"

Explanation:

After removing duplicates within each list,

responses = [["good", "ok"], ["ok", "bad", "good"], ["good"], ["bad"]]

.

"good"

appears 3 times,

"ok"

appears 2 times, and

"bad"

appears 2 times.

Return

"good"

because it has the highest frequency.

Example 2:

Input:

responses = [["good","ok","good"],["ok","bad"],["bad","notsure"],["great","good"]]

Output:

"bad"

Explanation:

After removing duplicates within each list we have

responses = [["good", "ok"], ["ok", "bad"], ["bad", "notsure"], ["great", "good"]]

.

"bad"

,

"good"

, and

"ok"

each occur 2 times.

The output is

"bad"

because it is the lexicographically smallest amongst the words with the highest frequency.

Constraints:

1 <= responses.length <= 1000

1 <= responses[i].length <= 1000

1 <= responses[i][j].length <= 10

responses[i][j]

consists of only lowercase English letters

Code Snippets

C++:

```
class Solution {
public:
string findCommonResponse(vector<vector<string>>& responses) {

}
};
```

Java:

```
class Solution {
public String findCommonResponse(List<List<String>> responses) {

}
}
```

Python3:

```
class Solution:
def findCommonResponse(self, responses: List[List[str]]) -> str:
```

Python:

```
class Solution(object):
def findCommonResponse(self, responses):
"""
:type responses: List[List[str]]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string[][]} responses
* @return {string}
*/
var findCommonResponse = function(responses) {

};
```

TypeScript:

```
function findCommonResponse(responses: string[][]): string {

};
```

C#:

```
public class Solution {
public string FindCommonResponse(IList<IList<string>> responses) {

}
}
```

C:

```
char* findCommonResponse(char*** responses, int responsesSize, int*
responsesColSize) {

}
```

Go:

```
func findCommonResponse(responses [][]string) string {

}
```

Kotlin:

```
class Solution {
fun findCommonResponse(responses: List<List<String>>): String {

}
}
```

Swift:

```
class Solution {
func findCommonResponse(_ responses: [[String]]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn find_common_response(responses: Vec<Vec<String>>) -> String {

}
}
```

Ruby:

```
# @param {String[][]} responses
# @return {String}
def find_common_response(responses)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $responses
* @return String
*/
function findCommonResponse($responses) {

}
}
```

Dart:

```
class Solution {
String findCommonResponse(List<List<String>> responses) {

}
}
```

Scala:

```
object Solution {
def findCommonResponse(responses: List[List[String]]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_common_response(responses :: [[String.t]]) :: String.t
def find_common_response(responses) do

end
end
```

Erlang:

```
-spec find_common_response(Responses :: [[unicode:unicode_binary()]]) ->
unicode:unicode_binary().
find_common_response(Responses) ->
.
```

Racket:

```
(define/contract (find-common-response responses)
(-> (listof (listof string?)) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Most Common Response
* 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 findCommonResponse(vector<vector<string>>& responses) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Most Common Response
* 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 findCommonResponse(List<List<String>> responses) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Most Common Response
```

```
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 findCommonResponse(self, responses: List[List[str]]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findCommonResponse(self, responses):
"""
:type responses: List[List[str]]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Most Common Response
* 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[][]} responses
* @return {string}
*/
var findCommonResponse = function(responses) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Most Common Response
* 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 findCommonResponse(responses: string[][]): string {

};
```

C# Solution:

```
/*
* Problem: Find the Most Common Response
* 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 FindCommonResponse(IList<IList<string>> responses) {

}
}
```

C Solution:

```
/*
* Problem: Find the Most Common Response
* 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* findCommonResponse(char*** responses, int responsesSize, int*
responsesColSize) {

}
```

Go Solution:

```
// Problem: Find the Most Common Response
// 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 findCommonResponse(responses [][]string) string {

}
```

Kotlin Solution:

```
class Solution {
fun findCommonResponse(responses: List<List<String>>): String {

}
}
```

Swift Solution:

```
class Solution {
func findCommonResponse(_ responses: [[String]]) -> String {

}
}
```

Rust Solution:

```
// Problem: Find the Most Common Response
// 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 find_common_response(responses: Vec<Vec<String>>) -> String {

}
}
```

Ruby Solution:

```
# @param {String[][]} responses
# @return {String}
def find_common_response(responses)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $responses
* @return String
*/
function findCommonResponse($responses) {

}
}
```

Dart Solution:

```
class Solution {
String findCommonResponse(List<List<String>> responses) {

}
}
```

Scala Solution:

```
object Solution {
def findCommonResponse(responses: List[List[String]]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_common_response(responses :: [[String.t]]) :: String.t
def find_common_response(responses) do

end
end
```

Erlang Solution:

```
-spec find_common_response(Responses :: [[unicode:unicode_binary()]]) ->
unicode:unicode_binary().
find_common_response(Responses) ->
.
```

Racket Solution:

```
(define/contract (find-common-response responses)
(-> (listof (listof string?)) string?)
)
```