Problem 1170: Compare Strings by Frequency of the Smallest Character

Problem Information

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

Problem Description

Let the function

f(s)

be the

frequency of the lexicographically smallest character

in a non-empty string

s

. For example, if

s = "dcce"

then

f(s) = 2

because the lexicographically smallest character is

'c'

, which has a frequency of 2.

You are given an array of strings

words

and another array of query strings

queries

. For each query

queries[i]

, count the

number of words

in

words

such that

f(queries[i])

<

f(W)

for each

W

in

words

.

Return

an integer array

answer

, where each

answer[i]

is the answer to the

i

th

query

.

Example 1:

Input:

queries = ["cbd"], words = ["zaaaz"]

Output:

[1]

Explanation:

On the first query we have f("cbd") = 1, f("zaaaz") = 3 so f("cbd") < f("zaaaz").

Example 2:

Input:

queries = ["bbb","cc"], words = ["a","aa","aaa","aaaa"]

Output:

[1,2]

Explanation:

On the first query only f("bbb") < f("aaaa"). On the second query both f("aaa") and f("aaaa") are both > f("cc").

Constraints:

1 <= queries.length <= 2000

1 <= words.length <= 2000

1 <= queries[i].length, words[i].length <= 10

queries[i][j]

,

words[i][j]

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> numSmallerByFrequency(vector<string>& queries, vector<string>&
words) {

}
};
```

Java:

```
class Solution {
public int[] numSmallerByFrequency(String[] queries, String[] words) {

}
}
```

Python3:

```
class Solution:
def numSmallerByFrequency(self, queries: List[str], words: List[str]) ->
List[int]:
```

Python:

```
class Solution(object):
def numSmallerByFrequency(self, queries, words):
"""
:type queries: List[str]
:type words: List[str]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string[]} queries
* @param {string[]} words
* @return {number[]}
*/
var numSmallerByFrequency = function(queries, words) {

};
```

TypeScript:

```
function numSmallerByFrequency(queries: string[], words: string[]): number[]
{

};
```

C#:

```
public class Solution {
public int[] NumSmallerByFrequency(string[] queries, string[] words) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numSmallerByFrequency(char** queries, int queriesSize, char** words, int
wordsSize, int* returnSize) {

}
```

Go:

```
func numSmallerByFrequency(queries []string, words []string) []int {

}
```

Kotlin:

```
class Solution {
fun numSmallerByFrequency(queries: Array<String>, words: Array<String>):
IntArray {

}
}
```

Swift:

```
class Solution {
func numSmallerByFrequency(_ queries: [String], _ words: [String]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn num_smaller_by_frequency(queries: Vec<String>, words: Vec<String>) ->
```

```
Vec<i32> {

}
}
```

Ruby:

```
# @param {String[]} queries
# @param {String[]} words
# @return {Integer[]}
def num_smaller_by_frequency(queries, words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $queries
* @param String[] $words
* @return Integer[]
*/
function numSmallerByFrequency($queries, $words) {

}
}
```

Dart:

```
class Solution {
List<int> numSmallerByFrequency(List<String> queries, List<String> words) {

}
}
```

Scala:

```
object Solution {
def numSmallerByFrequency(queries: Array[String], words: Array[String]):
Array[Int] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec num_smaller_by_frequency(queries :: [String.t], words :: [String.t]) ::
[integer]
def num_smaller_by_frequency(queries, words) do

end
end
```

Erlang:

```
-spec num_smaller_by_frequency(Queries :: [unicode:unicode_binary()], Words
:: [unicode:unicode_binary()]) -> [integer()].
num_smaller_by_frequency(Queries, Words) ->
.
```

Racket:

```
(define/contract (num-smaller-by-frequency queries words)
(-> (listof string?) (listof string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Compare Strings by Frequency of the Smallest Character
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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:
vector<int> numSmallerByFrequency(vector<string>& queries, vector<string>&
words) {

}
};
```

Java Solution:

```
/**
* Problem: Compare Strings by Frequency of the Smallest Character
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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 int[] numSmallerByFrequency(String[] queries, String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Compare Strings by Frequency of the Smallest Character
Difficulty: Medium
Tags: array, string, graph, hash, sort, search

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 numSmallerByFrequency(self, queries: List[str], words: List[str]) ->
List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def numSmallerByFrequency(self, queries, words):
"""
:type queries: List[str]
:type words: List[str]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Compare Strings by Frequency of the Smallest Character
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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[]} queries
* @param {string[]} words
* @return {number[]}
*/
var numSmallerByFrequency = function(queries, words) {

};
```

TypeScript Solution:

```
/**
* Problem: Compare Strings by Frequency of the Smallest Character
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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 numSmallerByFrequency(queries: string[], words: string[]): number[]
{

};
```

C# Solution:

```
/*
* Problem: Compare Strings by Frequency of the Smallest Character
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* 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 int[] NumSmallerByFrequency(string[] queries, string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Compare Strings by Frequency of the Smallest Character
* Difficulty: Medium
* Tags: array, string, graph, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
int* numSmallerByFrequency(char** queries, int queriesSize, char** words, int
wordsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Compare Strings by Frequency of the Smallest Character
// Difficulty: Medium
// Tags: array, string, graph, hash, sort, search
//
// 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 numSmallerByFrequency(queries []string, words []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun numSmallerByFrequency(queries: Array<String>, words: Array<String>):
IntArray {

}
}
```

Swift Solution:

```
class Solution {
func numSmallerByFrequency(_ queries: [String], _ words: [String]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Compare Strings by Frequency of the Smallest Character
// Difficulty: Medium
```

```
// Tags: array, string, graph, hash, sort, search
//
// 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 num_smaller_by_frequency(queries: Vec<String>, words: Vec<String>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String[]} queries
# @param {String[]} words
# @return {Integer[]}
def num_smaller_by_frequency(queries, words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $queries
* @param String[] $words
* @return Integer[]
*/
function numSmallerByFrequency($queries, $words) {

}
}
```

Dart Solution:

```
class Solution {
List<int> numSmallerByFrequency(List<String> queries, List<String> words) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def numSmallerByFrequency(queries: Array[String], words: Array[String]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_smaller_by_frequency(queries :: [String.t], words :: [String.t]) ::
[integer]
def num_smaller_by_frequency(queries, words) do

end
end
```

Erlang Solution:

```
-spec num_smaller_by_frequency(Queries :: [unicode:unicode_binary()], Words
:: [unicode:unicode_binary()]) -> [integer()].
num_smaller_by_frequency(Queries, Words) ->
.
```

Racket Solution:

```
(define/contract (num-smaller-by-frequency queries words)
(-> (listof string?) (listof string?) (listof exact-integer?))
)
```