Problem 1178: Number of Valid Words for Each Puzzle

Problem Information

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

Problem Description

With respect to a given

puzzle

string, a

word

is

valid

if both the following conditions are satisfied:

word

contains the first letter of

puzzle

.

For each letter in

word

, that letter is in

puzzle

.

For example, if the puzzle is

"abcdefg"

, then valid words are

"faced"

,

"cabbage"

, and

"baggage"

, while

invalid words are

"beefed"

(does not include

'a'

) and

"based"

(includes

's'

which is not in the puzzle).

Return

an array

answer

, where

answer[i]

is the number of words in the given word list

words

that is valid with respect to the puzzle

puzzles[i]

.

Example 1:

Input:

words = ["aaaa","asas","able","ability","actt","actor","access"], puzzles = ["aboveyz","abrodyz","abslute","absoryz","actresz","gaswxyz"]

Output:

[1,1,3,2,4,0]

Explanation:

1 valid word for "aboveyz" : "aaaa" 1 valid word for "abrodyz" : "aaaa" 3 valid words for "abslute" : "aaaa", "asas", "able" 2 valid words for "absoryz" : "aaaa", "asas" 4 valid words for

"actresz" : "aaaa", "asas", "actt", "access" There are no valid words for "gaswxyz" cause none of the words in the list contains letter 'g'.

Example 2:

Input:

words = ["apple","pleas","please"], puzzles = ["aelwxyz","aelpxyz","aelpsxy","saelpxy","xaelpsy"]

Output:

[0,1,3,2,0]

Constraints:

1 <= words.length <= 10

5

4 <= words[i].length <= 50

1 <= puzzles.length <= 10

4

puzzles[i].length == 7

words[i]

and

puzzles[i]

consist of lowercase English letters.

Each

puzzles[i]

does not contain repeated characters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> findNumOfValidWords(String[] words, String[] puzzles) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findNumOfValidWords(words: string[], puzzles: string[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindNumOfValidWords(string[] words, string[] puzzles) {

}
}
```

C:

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

}
```

Go:

```
func findNumOfValidWords(words []string, puzzles []string) []int {

}
```

Kotlin:

```
class Solution {
fun findNumOfValidWords(words: Array<String>, puzzles: Array<String>):
```

```
List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn find_num_of_valid_words(words: Vec<String>, puzzles: Vec<String>) ->
Vec<i32> {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

```
class Solution {
List<int> findNumOfValidWords(List<String> words, List<String> puzzles) {

}
}
```

Scala:

```
object Solution {
def findNumOfValidWords(words: Array[String], puzzles: Array[String]):
List[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec find_num_of_valid_words(Words :: [unicode:unicode_binary()], Puzzles ::
[unicode:unicode_binary()]) -> [integer()].
find_num_of_valid_words(Words, Puzzles) ->
.
```

Racket:

```
(define/contract (find-num-of-valid-words words puzzles)
(-> (listof string?) (listof string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Valid Words for Each Puzzle
* Difficulty: Hard
* Tags: array, string, 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:
vector<int> findNumOfValidWords(vector<string>& words, vector<string>&
puzzles) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Valid Words for Each Puzzle
* Difficulty: Hard
* Tags: array, string, 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 List<Integer> findNumOfValidWords(String[] words, String[] puzzles) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Valid Words for Each Puzzle
Difficulty: Hard
Tags: array, string, 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 findNumOfValidWords(self, words: List[str], puzzles: List[str]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Valid Words for Each Puzzle
* Difficulty: Hard
* Tags: array, string, 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[]} words
* @param {string[]} puzzles
* @return {number[]}
```

```
*/
var findNumOfValidWords = function(words, puzzles) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Valid Words for Each Puzzle
* Difficulty: Hard
* Tags: array, string, 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 findNumOfValidWords(words: string[], puzzles: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Valid Words for Each Puzzle
* Difficulty: Hard
* Tags: array, string, 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 IList<int> FindNumOfValidWords(string[] words, string[] puzzles) {

}
}
```

C Solution:

```
/*
* Problem: Number of Valid Words for Each Puzzle
* Difficulty: Hard
* Tags: array, string, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findNumOfValidWords(char** words, int wordsSize, char** puzzles, int
puzzlesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Valid Words for Each Puzzle
// Difficulty: Hard
// Tags: array, string, 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 findNumOfValidWords(words []string, puzzles []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findNumOfValidWords(words: Array<String>, puzzles: Array<String>):
List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Valid Words for Each Puzzle
// Difficulty: Hard
// Tags: array, string, 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_num_of_valid_words(words: Vec<String>, puzzles: Vec<String>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @param String[] $puzzles
* @return Integer[]
*/
function findNumOfValidWords($words, $puzzles) {
```

```
}
}
```

Dart Solution:

```
class Solution {
List<int> findNumOfValidWords(List<String> words, List<String> puzzles) {

}
}
```

Scala Solution:

```
object Solution {
def findNumOfValidWords(words: Array[String], puzzles: Array[String]):
List[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec find_num_of_valid_words(Words :: [unicode:unicode_binary()], Puzzles ::
[unicode:unicode_binary()]) -> [integer()].
find_num_of_valid_words(Words, Puzzles) ->
.
```

Racket Solution:

```
(define/contract (find-num-of-valid-words words puzzles)
(-> (listof string?) (listof string?) (listof exact-integer?))
)
```