Problem 748: Shortest Completing Word

Problem Information

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

Problem Description

Given a string

licensePlate

and an array of strings

words

, find the

shortest completing

word in

words

.

A

completing

word is a word that

contains all the letters

in

licensePlate

.

Ignore numbers and spaces

in

licensePlate

, and treat letters as

case insensitive

. If a letter appears more than once in

licensePlate

, then it must appear in the word the same number of times or more.

For example, if

licensePlate

= "aBc 12c"

, then it contains letters

'a'

,

'b'

(ignoring case), and

'c'

twice. Possible

completing

words are

"abccdef"

,

"caaacab"

, and

"cbca"

.

Return

the shortest

completing

word in

words

.

It is guaranteed an answer exists. If there are multiple shortest

completing

words, return the

first

one that occurs in

words

.

Example 1:

Input:

licensePlate = "1s3 PSt", words = ["step","steps","stripe","stepple"]

Output:

"steps"

Explanation:

licensePlate contains letters 's', 'p', 's' (ignoring case), and 't'. "step" contains 't' and 'p', but only contains 1 's'. "steps" contains 't', 'p', and both 's' characters. "stripe" is missing an 's'. "stepple" is missing an 's'. Since "steps" is the only word containing all the letters, that is the answer.

Example 2:

Input:

licensePlate = "1s3 456", words = ["looks","pest","stew","show"]

Output:

"pest"

Explanation:

licensePlate only contains the letter 's'. All the words contain 's', but among these "pest", "stew", and "show" are shortest. The answer is "pest" because it is the word that appears earliest of the 3.

Constraints:

1 <= licensePlate.length <= 7

licensePlate

contains digits, letters (uppercase or lowercase), or space

' '

.

1 <= words.length <= 1000

1 <= words[i].length <= 15

words[i]

consists of lower case English letters.

Code Snippets

C++:

```
class Solution {
public:
string shortestCompletingWord(string licensePlate, vector<string>& words) {

}
};
```

Java:

```
class Solution {
public String shortestCompletingWord(String licensePlate, String[] words) {

}
}
```

Python3:

```
class Solution:
def shortestCompletingWord(self, licensePlate: str, words: List[str]) -> str:
```

Python:

```
class Solution(object):
def shortestCompletingWord(self, licensePlate, words):
"""
:type licensePlate: str
:type words: List[str]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} licensePlate
* @param {string[]} words
* @return {string}
*/
var shortestCompletingWord = function(licensePlate, words) {

};
```

TypeScript:

```
function shortestCompletingWord(licensePlate: string, words: string[]):
string {

};
```

C#:

```
public class Solution {
public string ShortestCompletingWord(string licensePlate, string[] words) {

}
}
```

C:

```
char* shortestCompletingWord(char* licensePlate, char** words, int wordsSize)
{

}
```

Go:

```
func shortestCompletingWord(licensePlate string, words []string) string {

}
```

Kotlin:

```
class Solution {
fun shortestCompletingWord(licensePlate: String, words: Array<String>):
String {

}
}
```

Swift:

```
class Solution {
func shortestCompletingWord(_ licensePlate: String, _ words: [String]) ->
String {

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String} license_plate
# @param {String[]} words
# @return {String}
```

```
def shortest_completing_word(license_plate, words)

end
```

PHP:

```
class Solution {

/**
* @param String $licensePlate
* @param String[] $words
* @return String
*/
function shortestCompletingWord($licensePlate, $words) {

}
}
```

Dart:

```
class Solution {
String shortestCompletingWord(String licensePlate, List<String> words) {

}
}
```

Scala:

```
object Solution {
def shortestCompletingWord(licensePlate: String, words: Array[String]):
String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_completing_word(license_plate :: String.t, words ::
[String.t]) :: String.t
def shortest_completing_word(license_plate, words) do
```

```
end
end
```

Erlang:

```
-spec shortest_completing_word(LicensePlate :: unicode:unicode_binary(),
Words :: [unicode:unicode_binary()]) -> unicode:unicode_binary().
shortest_completing_word(LicensePlate, Words) ->
.
```

Racket:

```
(define/contract (shortest-completing-word licensePlate words)
(-> string? (listof string?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Completing Word
* Difficulty: Easy
* 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:
string shortestCompletingWord(string licensePlate, vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Completing Word
* Difficulty: Easy
* 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 String shortestCompletingWord(String licensePlate, String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Completing Word
Difficulty: Easy
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 shortestCompletingWord(self, licensePlate: str, words: List[str]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def shortestCompletingWord(self, licensePlate, words):
"""
:type licensePlate: str
:type words: List[str]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest Completing Word
* Difficulty: Easy
* 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} licensePlate
* @param {string[]} words
* @return {string}
*/
var shortestCompletingWord = function(licensePlate, words) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Completing Word
* Difficulty: Easy
* 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 shortestCompletingWord(licensePlate: string, words: string[]):
string {

};
```

C# Solution:

```
/*
* Problem: Shortest Completing Word
* Difficulty: Easy
* 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 string ShortestCompletingWord(string licensePlate, string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Completing Word
* Difficulty: Easy
* 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
*/

char* shortestCompletingWord(char* licensePlate, char** words, int wordsSize)
{

}
```

Go Solution:

```
// Problem: Shortest Completing Word
// Difficulty: Easy
// 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 shortestCompletingWord(licensePlate string, words []string) string {

}
```

Kotlin Solution:

```
class Solution {
fun shortestCompletingWord(licensePlate: String, words: Array<String>):
String {

}
}
```

Swift Solution:

```
class Solution {
func shortestCompletingWord(_ licensePlate: String, _ words: [String]) ->
String {

}
}
```

Rust Solution:

```
// Problem: Shortest Completing Word
// Difficulty: Easy
// 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 shortest_completing_word(license_plate: String, words: Vec<String>) ->
String {

}
}
```

Ruby Solution:

```
# @param {String} license_plate
# @param {String[]} words
# @return {String}
def shortest_completing_word(license_plate, words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $licensePlate
* @param String[] $words
* @return String
*/
function shortestCompletingWord($licensePlate, $words) {

}
}
```

Dart Solution:

```
class Solution {
String shortestCompletingWord(String licensePlate, List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def shortestCompletingWord(licensePlate: String, words: Array[String]):
String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_completing_word(license_plate :: String.t, words ::
```

```
[String.t]) :: String.t
def shortest_completing_word(license_plate, words) do

end
end
```

Erlang Solution:

```
-spec shortest_completing_word(LicensePlate :: unicode:unicode_binary(),
Words :: [unicode:unicode_binary()]) -> unicode:unicode_binary().
shortest_completing_word(LicensePlate, Words) ->
.
```

Racket Solution:

```
(define/contract (shortest-completing-word licensePlate words)
(-> string? (listof string?) string?)
)
```