Problem 2781: Length of the Longest Valid Substring

Problem Information

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

Problem Description

You are given a string

word

and an array of strings

forbidden

.

A string is called

valid

if none of its substrings are present in

forbidden

.

Return

the length of the

longest valid substring

of the string

word

.

A

substring

is a contiguous sequence of characters in a string, possibly empty.

Example 1:

Input:

word = "cbaaaabc", forbidden = ["aaa","cb"]

Output:

4

Explanation:

There are 11 valid substrings in word: "c", "b", "a", "ba", "aa", "bc", "baa", "aab", "ab", "abc" and "aabc". The length of the longest valid substring is 4. It can be shown that all other substrings contain either "aaa" or "cb" as a substring.

Example 2:

Input:

word = "leetcode", forbidden = ["de","le","e"]

Output:

4

Explanation:

There are 11 valid substrings in word: "l", "t", "c", "o", "d", "tc", "co", "od", "tco", "cod", and "tcod". The length of the longest valid substring is 4. It can be shown that all other substrings contain either "de", "le", or "e" as a substring.

Constraints:

1 <= word.length <= 10

5

word

consists only of lowercase English letters.

1 <= forbidden.length <= 10

5

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

forbidden[i]

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int longestValidSubstring(string word, vector<string>& forbidden) {

}
};
```

Java:

```
class Solution {
public int longestValidSubstring(String word, List<String> forbidden) {
}
}
```

Python3:

```
class Solution:
def longestValidSubstring(self, word: str, forbidden: List[str]) -> int:
```

Python:

```
class Solution(object):
def longestValidSubstring(self, word, forbidden):
"""
:type word: str
:type forbidden: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word
* @param {string[]} forbidden
* @return {number}
*/
var longestValidSubstring = function(word, forbidden) {

};
```

TypeScript:

```
function longestValidSubstring(word: string, forbidden: string[]): number {

};
```

C#:

```
public class Solution {
public int LongestValidSubstring(string word, IList<string> forbidden) {

}


}
```

C:

```
int longestValidSubstring(char* word, char** forbidden, int forbiddenSize) {


}
```

Go:

```
func longestValidSubstring(word string, forbidden []string) int {


}
```

Kotlin:

```
class Solution {
fun longestValidSubstring(word: String, forbidden: List<String>): Int {


}
}
```

Swift:

```
class Solution {
func longestValidSubstring(_ word: String, _ forbidden: [String]) -> Int {


}
}
```

Rust:

```
impl Solution {
pub fn longest_valid_substring(word: String, forbidden: Vec<String>) -> i32 {


}
}
```

Ruby:

```
# @param {String} word
# @param {String[]} forbidden
# @return {Integer}
def longest_valid_substring(word, forbidden)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @param String[] $forbidden
* @return Integer
*/
function longestValidSubstring($word, $forbidden) {

}
}
```

Dart:

```
class Solution {
int longestValidSubstring(String word, List<String> forbidden) {

}
}
```

Scala:

```
object Solution {
def longestValidSubstring(word: String, forbidden: List[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_valid_substring(word :: String.t, forbidden :: [String.t]) ::
integer
def longest_valid_substring(word, forbidden) do
```

```

end
end
```

Erlang:

```
-spec longest_valid_substring(Word :: unicode:unicode_binary(), Forbidden ::
[unicode:unicode_binary()]) -> integer().
longest_valid_substring(Word, Forbidden) ->
.
```

Racket:

```
(define/contract (longest-valid-substring word forbidden)
(-> string? (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Length of the Longest Valid Substring
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int longestValidSubstring(string word, vector<string>& forbidden) {

}
};
```

Java Solution:

```
/**
* Problem: Length of the Longest Valid Substring
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int longestValidSubstring(String word, List<String> forbidden) {

}
}
```

Python3 Solution:

```
"""
Problem: Length of the Longest Valid Substring
Difficulty: Hard
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def longestValidSubstring(self, word: str, forbidden: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestValidSubstring(self, word, forbidden):
"""
:type word: str
:type forbidden: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Length of the Longest Valid Substring
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} word
* @param {string[]} forbidden
* @return {number}
*/
var longestValidSubstring = function(word, forbidden) {

};
```

TypeScript Solution:

```
/**
* Problem: Length of the Longest Valid Substring
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function longestValidSubstring(word: string, forbidden: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Length of the Longest Valid Substring
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int LongestValidSubstring(string word, IList<string> forbidden) {

}
}
```

C Solution:

```
/*
* Problem: Length of the Longest Valid Substring
* Difficulty: Hard
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int longestValidSubstring(char* word, char** forbidden, int forbiddenSize) {

}
```

Go Solution:

```
// Problem: Length of the Longest Valid Substring
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func longestValidSubstring(word string, forbidden []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestValidSubstring(word: String, forbidden: List<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestValidSubstring(_ word: String, _ forbidden: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Length of the Longest Valid Substring
// Difficulty: Hard
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn longest_valid_substring(word: String, forbidden: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} word
# @param {String[]} forbidden
# @return {Integer}
def longest_valid_substring(word, forbidden)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @param String[] $forbidden
* @return Integer
*/
function longestValidSubstring($word, $forbidden) {

}
}
```

Dart Solution:

```
class Solution {
int longestValidSubstring(String word, List<String> forbidden) {

}
}
```

Scala Solution:

```
object Solution {
def longestValidSubstring(word: String, forbidden: List[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_valid_substring(word :: String.t, forbidden :: [String.t]) ::
integer
def longest_valid_substring(word, forbidden) do

end
end
```

Erlang Solution:

```
-spec longest_valid_substring(Word :: unicode:unicode_binary(), Forbidden ::
[unicode:unicode_binary()]) -> integer().
longest_valid_substring(Word, Forbidden) ->
.
```

Racket Solution:

```
(define/contract (longest-valid-substring word forbidden)
(-> string? (listof string?) exact-integer?)
)
```