Problem 3485: Longest Common Prefix of K Strings After Removal

Problem Information

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

Problem Description

You are given an array of strings

words

and an integer

k

.

For each index

i

in the range

[0, words.length - 1]

, find the

length

of the

longest common

prefix

among any

k

strings (selected at

distinct indices

) from the remaining array after removing the

i

th

element.

Return an array

answer

, where

answer[i]

is the answer for

i

th

element. If removing the

i

th

element leaves the array with fewer than

k

strings,

answer[i]

is 0.

Example 1:

Input:

words = ["jump","run","run","jump","run"], k = 2

Output:

[3,4,4,3,4]

Explanation:

Removing index 0 (

"jump"

):

words

becomes:

["run", "run", "jump", "run"]

.

"run"

occurs 3 times. Choosing any two gives the longest common prefix

"run"

(length 3).

Removing index 1 (

"run"

):

words

becomes:

["jump", "run", "jump", "run"]

.

"jump"

occurs twice. Choosing these two gives the longest common prefix

"jump"

(length 4).

Removing index 2 (

"run"

):

words

becomes:

["jump", "run", "jump", "run"]

.

"jump"

occurs twice. Choosing these two gives the longest common prefix

"jump"

(length 4).

Removing index 3 (

"jump"

):

words

becomes:

["jump", "run", "run", "run"]

.

"run"

occurs 3 times. Choosing any two gives the longest common prefix

"run"

(length 3).

Removing index 4 ("run"):

words

becomes:

["jump", "run", "run", "jump"]

.

"jump"

occurs twice. Choosing these two gives the longest common prefix

"jump"

(length 4).

Example 2:

Input:

words = ["dog","racer","car"], k = 2

Output:

[0,0,0]

Explanation:

Removing any index results in an answer of 0.

Constraints:

1 <= k <= words.length <= 10

5

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

4

words[i]

consists of lowercase English letters.

The sum of

words[i].length

is smaller than or equal

10

5

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] longestCommonPrefix(String[] words, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def longestCommonPrefix(self, words, k):
```

```
"""
:type words: List[str]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function longestCommonPrefix(words: string[], k: number): number[] {

};
```

C#:

```
public class Solution {
public int[] LongestCommonPrefix(string[] words, int k) {

}
}
```

C:

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

}
```

Go:

```
func longestCommonPrefix(words []string, k int) []int {

}
```

Kotlin:

```
class Solution {
fun longestCommonPrefix(words: Array<String>, k: Int): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
```

```
* @param Integer $k
* @return Integer[]
*/
function longestCommonPrefix($words, $k) {

}
}
```

Dart:

```
class Solution {
List<int> longestCommonPrefix(List<String> words, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec longest_common_prefix(Words :: [unicode:unicode_binary()], K ::
integer()) -> [integer()].
longest_common_prefix(Words, K) ->
.
```

Racket:

```
(define/contract (longest-common-prefix words k)
(-> (listof string?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Common Prefix of K Strings After Removal
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> longestCommonPrefix(vector<string>& words, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Common Prefix of K Strings After Removal
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] longestCommonPrefix(String[] words, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Longest Common Prefix of K Strings After Removal
Difficulty: Hard
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def longestCommonPrefix(self, words: List[str], k: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Common Prefix of K Strings After Removal
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Common Prefix of K Strings After Removal
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function longestCommonPrefix(words: string[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Longest Common Prefix of K Strings After Removal
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] LongestCommonPrefix(string[] words, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Longest Common Prefix of K Strings After Removal
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

Go Solution:

```
// Problem: Longest Common Prefix of K Strings After Removal
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func longestCommonPrefix(words []string, k int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun longestCommonPrefix(words: Array<String>, k: Int): IntArray {
```

```
}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Common Prefix of K Strings After Removal
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn longest_common_prefix(words: Vec<String>, k: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
```

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

}
}
```

Dart Solution:

```
class Solution {
List<int> longestCommonPrefix(List<String> words, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec longest_common_prefix(Words :: [unicode:unicode_binary()], K ::
integer()) -> [integer()].
longest_common_prefix(Words, K) ->
.
```

Racket Solution:

```
(define/contract (longest-common-prefix words k)
(-> (listof string?) exact-integer? (listof exact-integer?))
)
```