Problem 3598: Longest Common Prefix Between Adjacent Strings After Removals

Problem Information

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

Problem Description

You are given an array of strings

words

. For each index

i

in the range

[0, words.length - 1]

, perform the following steps:

Remove the element at index

i

from the

words

array.

Compute the

length

of the

longest common

prefix

among all

adjacent

pairs in the modified array.

Return an array

answer

, where

answer[i]

is the length of the longest common prefix between the adjacent pairs after removing the element at index

i

. If

no

adjacent pairs remain or if

none

share a common prefix, then

answer[i]

should be 0.

Example 1:

Input:

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

Output:

[3,0,0,3,3]

Explanation:

Removing index 0:

words

becomes

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

Longest adjacent pair is

["run", "run"]

having a common prefix

"run"

(length 3)

Removing index 1:

words

becomes

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

No adjacent pairs share a common prefix (length 0)

Removing index 2:

words

becomes

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

No adjacent pairs share a common prefix (length 0)

Removing index 3:

words

becomes

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

Longest adjacent pair is

["run", "run"]

having a common prefix

"run"

(length 3)

Removing index 4:

words becomes

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

Longest adjacent pair is

["run", "run"]

having a common prefix

"run"

(length 3)

Example 2:

Input:

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

Output:

[0,0,0]

Explanation:

Removing any index results in an answer of 0.

Constraints:

1 <= 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) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Common Prefix Between Adjacent Strings After Removals
* Difficulty: Medium
* 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) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Common Prefix Between Adjacent Strings After Removals
* Difficulty: Medium
* 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) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Common Prefix Between Adjacent Strings After Removals
```

```
Difficulty: Medium
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]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Common Prefix Between Adjacent Strings After Removals
* Difficulty: Medium
* 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
* @return {number[]}
*/
var longestCommonPrefix = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Common Prefix Between Adjacent Strings After Removals
* Difficulty: Medium
* 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[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Longest Common Prefix Between Adjacent Strings After Removals
* Difficulty: Medium
* 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) {

}
}
```

C Solution:

```
/*
* Problem: Longest Common Prefix Between Adjacent Strings After Removals
* Difficulty: Medium
* 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* returnSize) {

}
```

Go Solution:

```
// Problem: Longest Common Prefix Between Adjacent Strings After Removals
// Difficulty: Medium
// 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) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Common Prefix Between Adjacent Strings After Removals
// Difficulty: Medium
// 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>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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