Problem 2746: Decremental String Concatenation

Problem Information

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

Problem Description

You are given a

0-indexed

array

words

containing

n

strings.

Let's define a

join

operation

join(x, y)

between two strings

x

and

y

as concatenating them into

xy

. However, if the last character of

x

is equal to the first character of

y

, one of them is

deleted

.

For example

join("ab", "ba") = "aba"

and

join("ab", "cde") = "abcde"

.

You are to perform

n - 1

join

operations. Let

str

0

= words[0]

. Starting from

i = 1

up to

i = n - 1

, for the

i

th

operation, you can do one of the following:

Make

str

i

= join(str

i - 1

, words[i])

Make

str

i

= join(words[i], str

i - 1

)

Your task is to

minimize

the length of

str

n - 1

.

Return

an integer denoting the minimum possible length of

str

n - 1

.

Example 1:

Input:

words = ["aa","ab","bc"]

Output:

4

Explanation:

In this example, we can perform join operations in the following order to minimize the length of str

2

: str

0

= "aa" str

1

= join(str

0

, "ab") = "aab" str

2

= join(str

1

, "bc") = "aabc" It can be shown that the minimum possible length of str

2

is 4.

Example 2:

Input:

words = ["ab","b"]

Output:

2

Explanation:

In this example, str

0

= "ab", there are two ways to get str

1

: join(str

0

, "b") = "ab" or join("b", str

0

) = "bab". The first string, "ab", has the minimum length. Hence, the answer is 2.

Example 3:

Input:

words = ["aaa","c","aba"]

Output:

6

Explanation:

In this example, we can perform join operations in the following order to minimize the length of str

2

: str

0

= "aaa" str

1

= join(str

0

, "c") = "aaac" str

2

= join("aba", str

1

) = "abaaac" It can be shown that the minimum possible length of str

2

is 6.

Constraints:

1 <= words.length <= 1000

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

Each character in

words[i]

is an English lowercase letter

Code Snippets

C++:

```
class Solution {
public:
int minimizeConcatenatedLength(vector<string>& words) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} words
```

```
 * @return {number}
 */
var minimizeConcatenatedLength = function(words) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minimizeConcatenatedLength(char** words, int wordsSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun minimizeConcatenatedLength(words: Array<String>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimize-concatenated-length words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Decremental String Concatenation
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public:
int minimizeConcatenatedLength(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Decremental String Concatenation
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimizeConcatenatedLength(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Decremental String Concatenation
Difficulty: Medium
Tags: array, string, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Decremental String Concatenation
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Decremental String Concatenation
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimizeConcatenatedLength(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Decremental String Concatenation
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimizeConcatenatedLength(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Decremental String Concatenation
* Difficulty: Medium
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimizeConcatenatedLength(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Decremental String Concatenation
// Difficulty: Medium
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimizeConcatenatedLength(words []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimizeConcatenatedLength(words: Array<String>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Decremental String Concatenation
// Difficulty: Medium
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimize-concatenated-length words)
(-> (listof string?) exact-integer?)
)
```