Problem 443: String Compression

Problem Information

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

Problem Description

Given an array of characters

chars

, compress it using the following algorithm:

Begin with an empty string

s

. For each group of

consecutive repeating characters

in

chars

:

If the group's length is

1

, append the character to

s

.

Otherwise, append the character followed by the group's length.

The compressed string

s

should not be returned separately

, but instead, be stored

in the input character array

chars

. Note that group lengths that are

10

or longer will be split into multiple characters in

chars

.

After you are done

modifying the input array,

return

the new length of the array

.

You must write an algorithm that uses only constant extra space.

Note:

The characters in the array beyond the returned length do not matter and should be ignored.

Example 1:

Input:

chars = ["a","a","b","b","c","c","c"]

Output:

Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

Explanation:

The groups are "aa", "bb", and "ccc". This compresses to "a2b2c3".

Example 2:

Input:

chars = ["a"]

Output:

Return 1, and the first character of the input array should be: ["a"]

Explanation:

The only group is "a", which remains uncompressed since it's a single character.

Example 3:

Input:

chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"]

Output:

Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"].

Explanation:

The groups are "a" and "bbbbbbbbbbbb". This compresses to "ab12".

Constraints:

1 <= chars.length <= 2000

chars[i]

is a lowercase English letter, uppercase English letter, digit, or symbol.

Code Snippets

C++:

```
class Solution {
public:
int compress(vector<char>& chars) {

}
};
```

Java:

```
class Solution {
public int compress(char[] chars) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {character[]} chars
* @return {number}
*/
var compress = function(chars) {

};
```

TypeScript:

```
function compress(chars: string[]): number {

};
```

C#:

```
public class Solution {
public int Compress(char[] chars) {

}
}
```

C:

```
int compress(char* chars, int charsSize) {

}
```

Go:

```
func compress(chars []byte) int {
```

```
}
```

Kotlin:

```
class Solution {
fun compress(chars: CharArray): Int {

}
}
```

Swift:

```
class Solution {
func compress(_ chars: inout [Character]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn compress(chars: &mut Vec<char>) -> i32 {

}
}
```

Ruby:

```
# @param {Character[]} chars
# @return {Integer}
def compress(chars)

end
```

PHP:

```
class Solution {

/**
* @param String[] $chars
* @return Integer
*/
```

```
function compress(&$chars) {

}
}
```

Dart:

```
class Solution {
int compress(List<String> chars) {

}
}
```

Scala:

```
object Solution {
def compress(chars: Array[Char]): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: String Compression
* 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 compress(vector<char>& chars) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: String Compression
* 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 compress(char[] chars) {

}
}
```

Python3 Solution:

```
"""
Problem: String Compression
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 compress(self, chars: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def compress(self, chars):
```

```
"""
:type chars: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: String Compression
* 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 {character[]} chars
* @return {number}
*/
var compress = function(chars) {

};
```

TypeScript Solution:

```
/**
* Problem: String Compression
* 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 compress(chars: string[]): number {

};
```

C# Solution:

```
/*
* Problem: String Compression
* 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 Compress(char[] chars) {

}
}
```

C Solution:

```
/*
* Problem: String Compression
* 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
*/

int compress(char* chars, int charsSize) {

}
```

Go Solution:

```
// Problem: String Compression
// 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 compress(chars []byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun compress(chars: CharArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func compress(_ chars: inout [Character]) -> Int {

}
}
```

Rust Solution:

```
// Problem: String Compression
// 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 compress(chars: &mut Vec<char>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Character[]} chars
# @return {Integer}
```

```
def compress(chars)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $chars
* @return Integer
*/
function compress(&$chars) {

}
}
```

Dart Solution:

```
class Solution {
int compress(List<String> chars) {

}
}
```

Scala Solution:

```
object Solution {
def compress(chars: Array[Char]): Int = {

}
}
```