Problem 3163: String Compression III

Problem Information

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

Problem Description

Given a string

word

, compress it using the following algorithm:

Begin with an empty string

comp

. While

word

is

not

empty, use the following operation:

Remove a maximum length prefix of

word

made of a

single character

c

repeating

at most

9 times.

Append the length of the prefix followed by

c

to

comp

.

Return the string

comp

.

Example 1:

Input:

word = "abcde"

Output:

"1a1b1c1d1e"

Explanation:

Initially,

comp = ""

. Apply the operation 5 times, choosing

"a"

,

"b"

,

"c"

,

"d"

, and

"e"

as the prefix in each operation.

For each prefix, append

"1"

followed by the character to

comp

.

Example 2:

Input:

word = "aaaaaaaaaaaaaabb"

Output:

"9a5a2b"

Explanation:

Initially,

comp = ""

. Apply the operation 3 times, choosing

"aaaaaaaaa"

,

"aaaaa"

, and

"bb"

as the prefix in each operation.

For prefix

"aaaaaaaaa"

, append

"9"

followed by

"a"

to

comp

.

For prefix

"aaaaa"

, append

"5"

followed by

"a"

to

comp

.

For prefix

"bb"

, append

"2"

followed by

"b"

to

comp

.

Constraints:

1 <= word.length <= 2 * 10

5

word

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string compressedString(string word) {

}
};
```

Java:

```
class Solution {
public String compressedString(String word) {

}
}
```

Python3:

```
class Solution:
def compressedString(self, word: str) -> str:
```

Python:

```
class Solution(object):
def compressedString(self, word):
```

```
"""
:type word: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} word
* @return {string}
*/
var compressedString = function(word) {

};
```

TypeScript:

```
function compressedString(word: string): string {

};
```

C#:

```
public class Solution {
public string CompressedString(string word) {

}
}
```

C:

```
char* compressedString(char* word) {

}
```

Go:

```
func compressedString(word string) string {

}
```

Kotlin:

```
class Solution {
fun compressedString(word: String): String {

}
}
```

Swift:

```
class Solution {
func compressedString(_ word: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn compressed_string(word: String) -> String {

}
}
```

Ruby:

```
# @param {String} word
# @return {String}
def compressed_string(word)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @return String
*/
function compressedString($word) {

}
}
```

Dart:

```
class Solution {
String compressedString(String word) {

}
}
```

Scala:

```
object Solution {
def compressedString(word: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec compressed_string(word :: String.t) :: String.t
def compressed_string(word) do

end
end
```

Erlang:

```
-spec compressed_string(Word :: unicode:unicode_binary()) ->
unicode:unicode_binary().
compressed_string(Word) ->
.
```

Racket:

```
(define/contract (compressed-string word)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: String Compression III
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
string compressedString(string word) {

}
};
```

Java Solution:

```
/**
* Problem: String Compression III
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String compressedString(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: String Compression III
Difficulty: Medium
Tags: string
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def compressedString(self, word: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def compressedString(self, word):
"""
:type word: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: String Compression III
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} word
* @return {string}
*/
var compressedString = function(word) {

};
```

TypeScript Solution:

```
/**
* Problem: String Compression III
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function compressedString(word: string): string {

};
```

C# Solution:

```
/*
* Problem: String Compression III
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string CompressedString(string word) {

}
}
```

C Solution:

```
/*
* Problem: String Compression III
* Difficulty: Medium
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

char* compressedString(char* word) {

}
```

Go Solution:

```
// Problem: String Compression III
// Difficulty: Medium
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func compressedString(word string) string {

}
```

Kotlin Solution:

```
class Solution {
fun compressedString(word: String): String {

}
}
```

Swift Solution:

```
class Solution {
func compressedString(_ word: String) -> String {

}
}
```

Rust Solution:

```
// Problem: String Compression III
// Difficulty: Medium
// Tags: string
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn compressed_string(word: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} word
# @return {String}
def compressed_string(word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return String
*/
function compressedString($word) {

}
}
```

Dart Solution:

```
class Solution {
String compressedString(String word) {

}
}
```

Scala Solution:

```
object Solution {
def compressedString(word: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec compressed_string(word :: String.t) :: String.t
def compressed_string(word) do

end
end
```

Erlang Solution:

```
-spec compressed_string(Word :: unicode:unicode_binary()) ->
unicode:unicode_binary().
compressed_string(Word) ->
.
```

Racket Solution:

```
(define/contract (compressed-string word)
(-> string? string?)
)
```