Problem 1531: String Compression II

Problem Information

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

Problem Description

Run-length encoding

is a string compression method that works by replacing consecutive identical characters (repeated 2 or more times) with the concatenation of the character and the number marking the count of the characters (length of the run). For example, to compress the string

"aabccc"

we replace

"aa"

by

"a2"

and replace

"ccc"

by

"c3"

. Thus the compressed string becomes

"a2bc3"

.

Notice that in this problem, we are not adding

'1'

after single characters.

Given a string

s

and an integer

k

. You need to delete

at most

k

characters from

s

such that the run-length encoded version of

s

has minimum length.

Find the

minimum length of the run-length encoded version of

s

after deleting at most

k

characters

.

Example 1:

Input:

s = "aaabcccd", k = 2

Output:

4

Explanation:

Compressing s without deleting anything will give us "a3bc3d" of length 6. Deleting any of the characters 'a' or 'c' would at most decrease the length of the compressed string to 5, for instance delete 2 'a' then we will have s = "abcccd" which compressed is abc3d. Therefore, the optimal way is to delete 'b' and 'd', then the compressed version of s will be "a3c3" of length 4.

Example 2:

Input:

s = "aabbaa", k = 2

Output:

2

Explanation:

If we delete both 'b' characters, the resulting compressed string would be "a4" of length 2.

Example 3:

Input:

s = "aaaaaaaaaaa", k = 0

Output:

3

Explanation:

Since k is zero, we cannot delete anything. The compressed string is "a11" of length 3.

Constraints:

1 <= s.length <= 100

0 <= k <= s.length

s

contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int getLengthOfOptimalCompression(string s, int k) {

}
};
```

Java:

```
class Solution {
public int getLengthOfOptimalCompression(String s, int k) {
```

```
}
}
```

Python3:

```
class Solution:
def getLengthOfOptimalCompression(self, s: str, k: int) -> int:
```

Python:

```
class Solution(object):
def getLengthOfOptimalCompression(self, s, k):
"""
:type s: str
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @return {number}
*/
var getLengthOfOptimalCompression = function(s, k) {

};
```

TypeScript:

```
function getLengthOfOptimalCompression(s: string, k: number): number {

};
```

C#:

```
public class Solution {
public int GetLengthOfOptimalCompression(string s, int k) {

}
```

```
}
```

C:

```
int getLengthOfOptimalCompression(char* s, int k) {

}
```

Go:

```
func getLengthOfOptimalCompression(s string, k int) int {

}
```

Kotlin:

```
class Solution {
fun getLengthOfOptimalCompression(s: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func getLengthOfOptimalCompression(_ s: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn get_length_of_optimal_compression(s: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
```

```
# @return {Integer}
def get_length_of_optimal_compression(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function getLengthOfOptimalCompression($s, $k) {

}
}
```

Dart:

```
class Solution {
int getLengthOfOptimalCompression(String s, int k) {

}
}
```

Scala:

```
object Solution {
def getLengthOfOptimalCompression(s: String, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_length_of_optimal_compression(s :: String.t, k :: integer) ::
integer
def get_length_of_optimal_compression(s, k) do
```

```
end
end
```

Erlang:

```
-spec get_length_of_optimal_compression(S :: unicode:unicode_binary(), K ::
integer()) -> integer().
get_length_of_optimal_compression(S, K) ->
.
```

Racket:

```
(define/contract (get-length-of-optimal-compression s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: String Compression II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int getLengthOfOptimalCompression(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: String Compression II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int getLengthOfOptimalCompression(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: String Compression II
Difficulty: Hard
Tags: string, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def getLengthOfOptimalCompression(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getLengthOfOptimalCompression(self, s, k):
"""
:type s: str
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: String Compression II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @param {number} k
* @return {number}
*/
var getLengthOfOptimalCompression = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: String Compression II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function getLengthOfOptimalCompression(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: String Compression II
* Difficulty: Hard
```

```
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* 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 GetLengthOfOptimalCompression(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: String Compression II
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int getLengthOfOptimalCompression(char* s, int k) {

}
```

Go Solution:

```
// Problem: String Compression II
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func getLengthOfOptimalCompression(s string, k int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun getLengthOfOptimalCompression(s: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func getLengthOfOptimalCompression(_ s: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: String Compression II
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn get_length_of_optimal_compression(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def get_length_of_optimal_compression(s, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function getLengthOfOptimalCompression($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
int getLengthOfOptimalCompression(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def getLengthOfOptimalCompression(s: String, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_length_of_optimal_compression(s :: String.t, k :: integer) ::
integer
def get_length_of_optimal_compression(s, k) do

end
end
```

Erlang Solution:

```
-spec get_length_of_optimal_compression(S :: unicode:unicode_binary(), K ::
integer()) -> integer().
get_length_of_optimal_compression(S, K) ->
.
```

Racket Solution:

```
(define/contract (get-length-of-optimal-compression s k)
(-> string? exact-integer? exact-integer?)
)
```