Problem 3167: Better Compression of String

Problem Information

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

Problem Description

You are given a string

compressed

representing a compressed version of a string. The format is a character followed by its frequency. For example,

"a3b1a1c2"

is a compressed version of the string

"aaabacc"

.

We seek a

better compression

with the following conditions:

Each character should appear

only once

in the compressed version.

The characters should be in

alphabetical order

.

Return the

better compression

of

compressed

.

Note:

In the better version of compression, the order of letters may change, which is acceptable.

Example 1:

Input:

compressed = "a3c9b2c1"

Output:

"a3b2c10"

Explanation:

Characters "a" and "b" appear only once in the input, but "c" appears twice, once with a size of 9 and once with a size of 1.

Hence, in the resulting string, it should have a size of 10.

Example 2:

Input:

compressed = "c2b3a1"

Output:

"a1b3c2"

Example 3:

Input:

compressed = "a2b4c1"

Output:

"a2b4c1"

Constraints:

1 <= compressed.length <= 6 * 10

4

compressed

consists only of lowercase English letters and digits.

compressed

is a valid compression, i.e., each character is followed by its frequency.

Frequencies are in the range

[1, 10

4

]

and have no leading zeroes.

Code Snippets

C++:

```
class Solution {
public:
string betterCompression(string compressed) {

}
};
```

Java:

```
class Solution {
public String betterCompression(String compressed) {

}
}
```

Python3:

```
class Solution:
def betterCompression(self, compressed: str) -> str:
```

Python:

```
class Solution(object):
def betterCompression(self, compressed):
"""
:type compressed: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} compressed
```

```
 * @return {string}
 */
var betterCompression = function(compressed) {

};
```

TypeScript:

```
function betterCompression(compressed: string): string {

};
```

C#:

```
public class Solution {
public string BetterCompression(string compressed) {

}
}
```

C:

```
char* betterCompression(char* compressed) {

}
```

Go:

```
func betterCompression(compressed string) string {

}
```

Kotlin:

```
class Solution {
fun betterCompression(compressed: String): String {

}
}
```

Swift:

```
class Solution {
func betterCompression(_ compressed: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn better_compression(compressed: String) -> String {

}
}
```

Ruby:

```
# @param {String} compressed
# @return {String}
def better_compression(compressed)

end
```

PHP:

```
class Solution {

/**
* @param String $compressed
* @return String
*/
function betterCompression($compressed) {

}
}
```

Dart:

```
class Solution {
String betterCompression(String compressed) {

}
}
```

Scala:

```
object Solution {
def betterCompression(compressed: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec better_compression(compressed :: String.t) :: String.t
def better_compression(compressed) do

end
end
```

Erlang:

```
-spec better_compression(Compressed :: unicode:unicode_binary()) ->
unicode:unicode_binary().
better_compression(Compressed) ->
.
```

Racket:

```
(define/contract (better-compression compressed)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Better Compression of String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public:
string betterCompression(string compressed) {

}
};
```

Java Solution:

```
/**
* Problem: Better Compression of String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String betterCompression(String compressed) {

}
}
```

Python3 Solution:

```
"""
Problem: Better Compression of String
Difficulty: Medium
Tags: string, hash, sort

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

class Solution:
def betterCompression(self, compressed: str) -> str:
```

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

Python Solution:

```
class Solution(object):
def betterCompression(self, compressed):
"""
:type compressed: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Better Compression of String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} compressed
* @return {string}
*/
var betterCompression = function(compressed) {

};
```

TypeScript Solution:

```
/**
* Problem: Better Compression of String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

function betterCompression(compressed: string): string {

};
```

C# Solution:

```
/*
* Problem: Better Compression of String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string BetterCompression(string compressed) {

}
}
```

C Solution:

```
/*
* Problem: Better Compression of String
* Difficulty: Medium
* Tags: string, hash, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* betterCompression(char* compressed) {

}
```

Go Solution:

```
// Problem: Better Compression of String
// Difficulty: Medium
// Tags: string, hash, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func betterCompression(compressed string) string {

}
```

Kotlin Solution:

```
class Solution {
fun betterCompression(compressed: String): String {

}
}
```

Swift Solution:

```
class Solution {
func betterCompression(_ compressed: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Better Compression of String
// Difficulty: Medium
// Tags: string, hash, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn better_compression(compressed: String) -> String {
```

```
}
}
```

Ruby Solution:

```
# @param {String} compressed
# @return {String}
def better_compression(compressed)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $compressed
* @return String
*/
function betterCompression($compressed) {

}
}
```

Dart Solution:

```
class Solution {
String betterCompression(String compressed) {

}
}
```

Scala Solution:

```
object Solution {
def betterCompression(compressed: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec better_compression(compressed :: String.t) :: String.t
def better_compression(compressed) do

end
end
```

Erlang Solution:

```
-spec better_compression(Compressed :: unicode:unicode_binary()) ->
unicode:unicode_binary().
better_compression(Compressed) ->
.
```

Racket Solution:

```
(define/contract (better-compression compressed)
(-> string? string?)
)
```