Problem 2182: Construct String With Repeat Limit

Problem Information

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

Problem Description

You are given a string

s

and an integer

repeatLimit

. Construct a new string

repeatLimitedString

using the characters of

s

such that no letter appears

more than

repeatLimit

times

in a row

. You do

not

have to use all characters from

s

.

Return

the

lexicographically largest

repeatLimitedString

possible

.

A string

a

is

lexicographically larger

than a string

b

if in the first position where

a

and

b

differ, string

a

has a letter that appears later in the alphabet than the corresponding letter in

b

. If the first

min(a.length, b.length)

characters do not differ, then the longer string is the lexicographically larger one.

Example 1:

Input:

s = "cczazcc", repeatLimit = 3

Output:

"zzcccac"

Explanation:

We use all of the characters from s to construct the repeatLimitedString "zzcccac". The letter 'a' appears at most 1 time in a row. The letter 'c' appears at most 3 times in a row. The letter 'z' appears at most 2 times in a row. Hence, no letter appears more than repeatLimit times in a row and the string is a valid repeatLimitedString. The string is the lexicographically largest repeatLimitedString possible so we return "zzcccac". Note that the string "zzcccca" is lexicographically larger but the letter 'c' appears more than 3 times in a row, so it is not a valid repeatLimitedString.

Example 2:

Input:

s = "aababab", repeatLimit = 2

Output:

"bbabaa"

Explanation:

We use only some of the characters from s to construct the repeatLimitedString "bbabaa". The letter 'a' appears at most 2 times in a row. The letter 'b' appears at most 2 times in a row. Hence, no letter appears more than repeatLimit times in a row and the string is a valid repeatLimitedString. The string is the lexicographically largest repeatLimitedString possible so we return "bbabaa". Note that the string "bbabaaa" is lexicographically larger but the letter 'a' appears more than 2 times in a row, so it is not a valid repeatLimitedString.

Constraints:

1 <= repeatLimit <= s.length <= 10

5

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
string repeatLimitedString(string s, int repeatLimit) {

}
};
```

Java:

```
class Solution {
public String repeatLimitedString(String s, int repeatLimit) {

}
}
```

Python3:

```
class Solution:
def repeatLimitedString(self, s: str, repeatLimit: int) -> str:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function repeatLimitedString(s: string, repeatLimit: number): string {

};
```

C#:

```
public class Solution {
public string RepeatLimitedString(string s, int repeatLimit) {
}
}
```

C:

```
char* repeatLimitedString(char* s, int repeatLimit) {

}
```

Go:

```
func repeatLimitedString(s string, repeatLimit int) string {

}
```

Kotlin:

```
class Solution {
fun repeatLimitedString(s: String, repeatLimit: Int): String {

}
}
```

Swift:

```
class Solution {
func repeatLimitedString(_ s: String, _ repeatLimit: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn repeat_limited_string(s: String, repeat_limit: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} repeat_limit
# @return {String}
def repeat_limited_string(s, repeat_limit)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $repeatLimit
* @return String
*/
function repeatLimitedString($s, $repeatLimit) {

}
}
```

Dart:

```
class Solution {
String repeatLimitedString(String s, int repeatLimit) {

}
}
```

Scala:

```
object Solution {
def repeatLimitedString(s: String, repeatLimit: Int): String = {

}
}
```

Elixir:

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

```
end
end
```

Erlang:

```
-spec repeat_limited_string(S :: unicode:unicode_binary(), RepeatLimit ::
integer()) -> unicode:unicode_binary().
repeat_limited_string(S, RepeatLimit) ->
.
```

Racket:

```
(define/contract (repeat-limited-string s repeatLimit)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Construct String With Repeat Limit
* Difficulty: Medium
* Tags: string, graph, greedy, hash, queue, heap
*
* 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 repeatLimitedString(string s, int repeatLimit) {

}
};
```

Java Solution:

```
/**
* Problem: Construct String With Repeat Limit
* Difficulty: Medium
* Tags: string, graph, greedy, hash, queue, heap
*
* 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 repeatLimitedString(String s, int repeatLimit) {

}
}
```

Python3 Solution:

```
"""
Problem: Construct String With Repeat Limit
Difficulty: Medium
Tags: string, graph, greedy, hash, queue, heap

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 repeatLimitedString(self, s: str, repeatLimit: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Construct String With Repeat Limit
* Difficulty: Medium
* Tags: string, graph, greedy, hash, queue, heap
*
* 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} s
* @param {number} repeatLimit
* @return {string}
*/
var repeatLimitedString = function(s, repeatLimit) {

};
```

TypeScript Solution:

```
/**
* Problem: Construct String With Repeat Limit
* Difficulty: Medium
* Tags: string, graph, greedy, hash, queue, heap
*
* 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 repeatLimitedString(s: string, repeatLimit: number): string {

};
```

C# Solution:

```
/*
* Problem: Construct String With Repeat Limit
* Difficulty: Medium
* Tags: string, graph, greedy, hash, queue, heap
*
* 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 RepeatLimitedString(string s, int repeatLimit) {

}
}
```

C Solution:

```
/*
* Problem: Construct String With Repeat Limit
* Difficulty: Medium
* Tags: string, graph, greedy, hash, queue, heap
*
* 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* repeatLimitedString(char* s, int repeatLimit) {

}
```

Go Solution:

```
// Problem: Construct String With Repeat Limit
// Difficulty: Medium
// Tags: string, graph, greedy, hash, queue, heap
//
// 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 repeatLimitedString(s string, repeatLimit int) string {
}
```

Kotlin Solution:

```
class Solution {
fun repeatLimitedString(s: String, repeatLimit: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func repeatLimitedString(_ s: String, _ repeatLimit: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Construct String With Repeat Limit
// Difficulty: Medium
// Tags: string, graph, greedy, hash, queue, heap
//
// 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 repeat_limited_string(s: String, repeat_limit: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} repeat_limit
# @return {String}
def repeat_limited_string(s, repeat_limit)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $repeatLimit
* @return String
*/
function repeatLimitedString($s, $repeatLimit) {

}
}
```

Dart Solution:

```
class Solution {
String repeatLimitedString(String s, int repeatLimit) {

}
}
```

Scala Solution:

```
object Solution {
def repeatLimitedString(s: String, repeatLimit: Int): String = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec repeat_limited_string(S :: unicode:unicode_binary(), RepeatLimit ::
integer()) -> unicode:unicode_binary().
repeat_limited_string(S, RepeatLimit) ->
.
```

Racket Solution:

```
(define/contract (repeat-limited-string s repeatLimit)
(-> string? exact-integer? string?)
)
```