Problem 2468: Split Message Based on Limit

Problem Information

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

Problem Description

You are given a string,

message

, and a positive integer,

limit

.

You must

split

message

into one or more

parts

based on

limit

. Each resulting part should have the suffix

"<a/b>"

, where

"b"

is to be

replaced

with the total number of parts and

"a"

is to be

replaced

with the index of the part, starting from

1

and going up to

b

. Additionally, the length of each resulting part (including its suffix) should be

equal

to

limit

, except for the last part whose length can be

at most

limit

.

The resulting parts should be formed such that when their suffixes are removed and they are all concatenated

in order

, they should be equal to

message

. Also, the result should contain as few parts as possible.

Return

the parts

message

would be split into as an array of strings

. If it is impossible to split

message

as required, return

an empty array

.

Example 1:

Input:

message = "this is really a very awesome message", limit = 9

Output:

' '

["thi<1/14>","s i<2/14>","s r<3/14>","eal<4/14>","ly <5/14>","a v<6/14>","ery<7/14>"," aw<8/14>","eso<9/14>","me<10/14>"," m<11/14>","es<12/14>","sa<13/14>","ge<14/14>"]

Explanation:

The first 9 parts take 3 characters each from the beginning of message. The next 5 parts take 2 characters each to finish splitting message. In this example, each part, including the last, has length 9. It can be shown it is not possible to split message into less than 14 parts.

Example 2:

Input:

message = "short message", limit = 15

Output:

["short mess<1/2>","age<2/2>"]

Explanation:

Under the given constraints, the string can be split into two parts: - The first part comprises of the first 10 characters, and has a length 15. - The next part comprises of the last 3 characters, and has a length 8.

Constraints:

1 <= message.length <= 10

4

message

consists only of lowercase English letters and

' '

.

1 <= limit <= 10

4

Code Snippets

C++:

```
class Solution {
public:
vector<string> splitMessage(string message, int limit) {

}
};
```

Java:

```
class Solution {
public String[] splitMessage(String message, int limit) {

}
}
```

Python3:

```
class Solution:
def splitMessage(self, message: str, limit: int) -> List[str]:
```

Python:

```
class Solution(object):
def splitMessage(self, message, limit):
"""
:type message: str
:type limit: int
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} message
* @param {number} limit
* @return {string[]}
*/
var splitMessage = function(message, limit) {

};
```

TypeScript:

```
function splitMessage(message: string, limit: number): string[] {

};
```

C#:

```
public class Solution {
public string[] SplitMessage(string message, int limit) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** splitMessage(char* message, int limit, int* returnSize) {

}
```

Go:

```
func splitMessage(message string, limit int) []string {

}
```

Kotlin:

```
class Solution {
fun splitMessage(message: String, limit: Int): Array<String> {


}
}
```

Swift:

```
class Solution {
func splitMessage(_ message: String, _ limit: Int) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn split_message(message: String, limit: i32) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} message
# @param {Integer} limit
# @return {String[]}
def split_message(message, limit)

end
```

PHP:

```
class Solution {

/**
* @param String $message
* @param Integer $limit
* @return String[]
*/
function splitMessage($message, $limit) {

}
}
```

Dart:

```
class Solution {
List<String> splitMessage(String message, int limit) {

}
}
```

Scala:

```
object Solution {
def splitMessage(message: String, limit: Int): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec split_message(message :: String.t, limit :: integer) :: [String.t]
def split_message(message, limit) do

end
end
```

Erlang:

```
-spec split_message(Message :: unicode:unicode_binary(), Limit :: integer())
-> [unicode:unicode_binary()].
split_message(Message, Limit) ->
.
```

Racket:

```
(define/contract (split-message message limit)
(-> string? exact-integer? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split Message Based on Limit
* Difficulty: Hard
* Tags: array, string, search
*
* 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:
vector<string> splitMessage(string message, int limit) {

}
};
```

Java Solution:

```
/**
* Problem: Split Message Based on Limit
* Difficulty: Hard
* Tags: array, string, search
*
* 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 String[] splitMessage(String message, int limit) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Message Based on Limit
Difficulty: Hard
Tags: array, string, search
```

```
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 splitMessage(self, message: str, limit: int) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def splitMessage(self, message, limit):
"""
:type message: str
:type limit: int
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Split Message Based on Limit
* Difficulty: Hard
* Tags: array, string, search
*
* 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 {string} message
* @param {number} limit
* @return {string[]}
*/
var splitMessage = function(message, limit) {

};
```

TypeScript Solution:

```
/**
* Problem: Split Message Based on Limit
* Difficulty: Hard
* Tags: array, string, search
*
* 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 splitMessage(message: string, limit: number): string[] {

};
```

C# Solution:

```
/*
* Problem: Split Message Based on Limit
* Difficulty: Hard
* Tags: array, string, search
*
* 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 string[] SplitMessage(string message, int limit) {

}
}
```

C Solution:

```
/*
* Problem: Split Message Based on Limit
* Difficulty: Hard
* Tags: array, string, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** splitMessage(char* message, int limit, int* returnSize) {

}
```

Go Solution:

```
// Problem: Split Message Based on Limit
// Difficulty: Hard
// Tags: array, string, search
//
// 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 splitMessage(message string, limit int) []string {

}
```

Kotlin Solution:

```
class Solution {
fun splitMessage(message: String, limit: Int): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func splitMessage(_ message: String, _ limit: Int) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Split Message Based on Limit
// Difficulty: Hard
// Tags: array, string, search
//
// 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 split_message(message: String, limit: i32) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} message
# @param {Integer} limit
# @return {String[]}
def split_message(message, limit)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $message
* @param Integer $limit
* @return String[]
*/
function splitMessage($message, $limit) {

}
}
```

Dart Solution:

```
class Solution {
List<String> splitMessage(String message, int limit) {


}
}
```

Scala Solution:

```
object Solution {
def splitMessage(message: String, limit: Int): Array[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec split_message(message :: String.t, limit :: integer) :: [String.t]
def split_message(message, limit) do

end
end
```

Erlang Solution:

```
-spec split_message(Message :: unicode:unicode_binary(), Limit :: integer())
-> [unicode:unicode_binary()].
split_message(Message, Limit) ->
.
```

Racket Solution:

```
(define/contract (split-message message limit)
(-> string? exact-integer? (listof string?))
)
```