Problem 763: Partition Labels

Problem Information

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

Problem Description

You are given a string

s

. We want to partition the string into as many parts as possible so that each letter appears in at most one part. For example, the string

"ababcc"

can be partitioned into

["abab", "cc"]

, but partitions such as

["aba", "bcc"]

or

["ab", "ab", "cc"]

are invalid.

Note that the partition is done so that after concatenating all the parts in order, the resultant string should be

s

.

Return

a list of integers representing the size of these parts

.

Example 1:

Input:

s = "ababcbacadefegdehijhklij"

Output:

[9,7,8]

Explanation:

The partition is "ababcbaca", "defegde", "hijhklij". This is a partition so that each letter appears in at most one part. A partition like "ababcbacadefegde", "hijhklij" is incorrect, because it splits s into less parts.

Example 2:

Input:

s = "eccbbbbdec"

Output:

[10]

Constraints:

1 <= s.length <= 500

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> partitionLabels(string s) {

}
};
```

Java:

```
class Solution {
public List<Integer> partitionLabels(String s) {

}
}
```

Python3:

```
class Solution:
def partitionLabels(self, s: str) -> List[int]:
```

Python:

```
class Solution(object):
def partitionLabels(self, s):
"""
:type s: str
:rtype: List[int]
"""
```

JavaScript:

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

```
* @return {number[]}
*/
var partitionLabels = function(s) {

};
```

TypeScript:

```
function partitionLabels(s: string): number[] {

};
```

C#:

```
public class Solution {
public IList<int> PartitionLabels(string s) {

}
}
```

C:

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

}
```

Go:

```
func partitionLabels(s string) []int {

}
```

Kotlin:

```
class Solution {
fun partitionLabels(s: String): List<Int> {

}
}
```

Swift:

```
class Solution {
func partitionLabels(_ s: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn partition_labels(s: String) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer[]}
def partition_labels(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer[]
*/
function partitionLabels($s) {

}
}
```

Dart:

```
class Solution {
List<int> partitionLabels(String s) {

}
```

```
}
```

Scala:

```
object Solution {
def partitionLabels(s: String): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec partition_labels(s :: String.t) :: [integer]
def partition_labels(s) do

end
end
```

Erlang:

```
-spec partition_labels(S :: unicode:unicode_binary()) -> [integer()].
partition_labels(S) ->
.
```

Racket:

```
(define/contract (partition-labels s)
(-> string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partition Labels
* Difficulty: Medium
* Tags: array, string, greedy, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> partitionLabels(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Labels
* Difficulty: Medium
* Tags: array, string, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<Integer> partitionLabels(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition Labels
Difficulty: Medium
Tags: array, string, greedy, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def partitionLabels(self, s: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def partitionLabels(self, s):
"""
:type s: str
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Partition Labels
* Difficulty: Medium
* Tags: array, string, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @return {number[]}
*/
var partitionLabels = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition Labels
* Difficulty: Medium
* Tags: array, string, greedy, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function partitionLabels(s: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Partition Labels
* Difficulty: Medium
* Tags: array, string, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public IList<int> PartitionLabels(string s) {

}
}
```

C Solution:

```
/*
* Problem: Partition Labels
* Difficulty: Medium
* Tags: array, string, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
```

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

}
```

Go Solution:

```
// Problem: Partition Labels
// Difficulty: Medium
// Tags: array, string, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func partitionLabels(s string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun partitionLabels(s: String): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func partitionLabels(_ s: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Partition Labels
// Difficulty: Medium
// Tags: array, string, greedy, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn partition_labels(s: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer[]}
def partition_labels(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer[]
*/
function partitionLabels($s) {

}
}
```

Dart Solution:

```
class Solution {
List<int> partitionLabels(String s) {

}
}
```

Scala Solution:

```
object Solution {
def partitionLabels(s: String): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec partition_labels(s :: String.t) :: [integer]
def partition_labels(s) do

end
end
```

Erlang Solution:

```
-spec partition_labels(S :: unicode:unicode_binary()) -> [integer()].
partition_labels(S) ->
.
```

Racket Solution:

```
(define/contract (partition-labels s)
(-> string? (listof exact-integer?))
)
```