Problem 830: Positions of Large Groups

Problem Information

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

Problem Description

In a string

s

of lowercase letters, these letters form consecutive groups of the same character.

For example, a string like

s = "abbxxxxzyy"

has the groups

"a"

,

"bb"

,

"xxxx"

,

"z"

, and

"yy"

.

A group is identified by an interval

[start, end]

, where

start

and

end

denote the start and end indices (inclusive) of the group. In the above example,

"xxxx"

has the interval

[3,6]

.

A group is considered

large

if it has 3 or more characters.

Return

the intervals of every

large

group sorted in

increasing order by start index

.

Example 1:

Input:

s = "abbxxxxzzy"

Output:

[[3,6]]

Explanation:

"xxxx" is the only

large group with start index 3 and end index 6.

Example 2:

Input:

s = "abc"

Output:

[]

Explanation:

We have groups "a", "b", and "c", none of which are large groups.

Example 3:

Input:

s = "abcdddeeeeaabbbcd"

Output:

[[3,5],[6,9],[12,14]]

Explanation:

The large groups are "ddd", "eeee", and "bbb".

Constraints:

1 <= s.length <= 1000

s

contains lowercase English letters only.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** largeGroupPositions(char* s, int* returnSize, int** returnColumnSizes)
{

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer[][]}
```

```
def large_group_positions(s)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (large-group-positions s)
(-> string? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Positions of Large Groups
* Difficulty: Easy
* Tags: string, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Positions of Large Groups
* Difficulty: Easy
* Tags: string, sort
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Positions of Large Groups
Difficulty: Easy
Tags: string, sort

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Positions of Large Groups
* Difficulty: Easy
```

```
* Tags: string, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Positions of Large Groups
* Difficulty: Easy
* Tags: string, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

C# Solution:

```
/*
* Problem: Positions of Large Groups
* Difficulty: Easy
* Tags: string, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

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

}
}
```

C Solution:

```
/*
* Problem: Positions of Large Groups
* Difficulty: Easy
* Tags: string, sort
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** largeGroupPositions(char* s, int* returnSize, int** returnColumnSizes)
{

}
```

Go Solution:

```
// Problem: Positions of Large Groups
// Difficulty: Easy
// Tags: string, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Positions of Large Groups
// Difficulty: Easy
// Tags: string, sort
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (large-group-positions s)
(-> string? (listof (listof exact-integer?)))
)
```