Problem 1520: Maximum Number of Non-Overlapping Substrings

Problem Information

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

Problem Description

Given a string

s

of lowercase letters, you need to find the maximum number of

non-empty

substrings of

s

that meet the following conditions:

The substrings do not overlap, that is for any two substrings

s[i..j]

and

s[x..y]

, either

j < x

or

i > y

is true.

A substring that contains a certain character

c

must also contain all occurrences of

c

.

Find

the maximum number of substrings that meet the above conditions

. If there are multiple solutions with the same number of substrings,

return the one with minimum total length.

It can be shown that there exists a unique solution of minimum total length.

Notice that you can return the substrings in

any

order.

Example 1:

Input:

s = "adefaddaccc"

Output:

["e","f","ccc"]

Explanation:

The following are all the possible substrings that meet the conditions: ["adefaddaccc" "adefadda", "ef", "e", "f", "ccc",] If we choose the first string, we cannot choose anything else and we'd get only 1. If we choose "adefadda", we are left with "ccc" which is the only one that doesn't overlap, thus obtaining 2 substrings. Notice also, that it's not optimal to choose "ef" since it can be split into two. Therefore, the optimal way is to choose ["e","f","ccc"] which gives us 3 substrings. No other solution of the same number of substrings exist.

Example 2:

Input:

s = "abbaccd"

Output:

["d","bb","cc"]

Explanation:

Notice that while the set of substrings ["d","abba","cc"] also has length 3, it's considered incorrect since it has larger total length.

Constraints:

1 <= s.length <= 10

5

s

contains only lowercase English letters.

Code Snippets

C++:

```cpp
class Solution {
public:
vector<string> maxNumOfSubstrings(string s) {

}
};
```

Java:

```java
class Solution {
public List<String> maxNumOfSubstrings(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxNumOfSubstrings(s: string): string[] {


};
```

C#:

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


}
}
```

C:

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


}
```

Go:

```
func maxNumOfSubstrings(s string) []string {


}
```

Kotlin:

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


}
}
```

Swift:

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


}

}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<String> maxNumOfSubstrings(String s) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-num-of-substrings s)
(-> string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Non-Overlapping Substrings
* Difficulty: Hard
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Non-Overlapping Substrings
* Difficulty: Hard
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<String> maxNumOfSubstrings(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Non-Overlapping Substrings
Difficulty: Hard
Tags: string, tree, greedy

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Non-Overlapping Substrings
* Difficulty: Hard
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Non-Overlapping Substrings
* Difficulty: Hard
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

function maxNumOfSubstrings(s: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Non-Overlapping Substrings
* Difficulty: Hard
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<string> MaxNumOfSubstrings(string s) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Non-Overlapping Substrings
* Difficulty: Hard
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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


}
```

Go Solution:

```
// Problem: Maximum Number of Non-Overlapping Substrings
// Difficulty: Hard
// Tags: string, tree, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxNumOfSubstrings(s string) []string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Non-Overlapping Substrings
// Difficulty: Hard
// Tags: string, tree, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height
```

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<String> maxNumOfSubstrings(String s) {

}
}
```

Scala Solution:

```
object Solution {
def maxNumOfSubstrings(s: String): List[String] = {
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-num-of-substrings s)
(-> string? (listof string?))
)
```