Problem 3597: Partition String

Problem Information

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

Problem Description

Given a string

s

, partition it into

unique segments

according to the following procedure:

Start building a segment beginning at index 0.

Continue extending the current segment character by character until the current segment has not been seen before.

Once the segment is unique, add it to your list of segments, mark it as seen, and begin a new segment from the next index.

Repeat until you reach the end of

s

.

Return an array of strings

segments

, where

segments[i]

is the

i

th

segment created.

Example 1:

Input:

s = "abbccccd"

Output:

["a","b","bc","c","cc","d"]

Explanation:

Index

Segment After Adding

Seen Segments

Current Segment Seen Before?

New Segment

Updated Seen Segments

0

"a"

[]

No

""

["a"]

1

"b"

["a"]

No

""

["a", "b"]

2

"b"

["a", "b"]

Yes

"b"

["a", "b"]

3

"bc"

["a", "b"]

No

""

["a", "b", "bc"]

4

"c"

["a", "b", "bc"]

No

""

["a", "b", "bc", "c"]

5

"c"

["a", "b", "bc", "c"]

Yes

"c"

["a", "b", "bc", "c"]

6

"cc"

["a", "b", "bc", "c"]

No

""

["a", "b", "bc", "c", "cc"]

7

"d"

["a", "b", "bc", "c", "cc"]

No

""

["a", "b", "bc", "c", "cc", "d"]

Hence, the final output is

["a", "b", "bc", "c", "cc", "d"]

.

Example 2:

Input:

s = "aaaa"

Output:

["a","aa"]

Explanation:

Index

Segment After Adding

Seen Segments

Current Segment Seen Before?

New Segment

Updated Seen Segments

0

"a"

[]

No

""

["a"]

1

"a"

["a"]

Yes

"a"

["a"]

2

"aa"

["a"]

No

""

["a", "aa"]

3

"a"

["a", "aa"]

Yes

"a"

["a", "aa"]

Hence, the final output is

["a", "aa"]

.

Constraints:

1 <= s.length <= 10

5

s

contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<string> partitionString(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Partition String
* Difficulty: Medium
* Tags: array, string, 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<string> partitionString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Partition String
* Difficulty: Medium
* Tags: array, string, 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<String> partitionString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition String
Difficulty: Medium
Tags: array, string, 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 partitionString(self, s: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition String
* Difficulty: Medium
* Tags: array, string, 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 {string[]}
*/
var partitionString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition String
* Difficulty: Medium
* Tags: array, string, 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 partitionString(s: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Partition String
* Difficulty: Medium
* Tags: array, string, 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<string> PartitionString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Partition String
* Difficulty: Medium
* Tags: array, string, 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().
*/
char** partitionString(char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: Partition String
// Difficulty: Medium
// Tags: array, string, 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 partitionString(s string) []string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition String
// Difficulty: Medium
// Tags: array, string, 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_string(s: String) -> Vec<String> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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