Problem 131: Palindrome Partitioning

Problem Information

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

Problem Description

Given a string

s

, partition

s

such that every

substring

of the partition is a

palindrome

. Return

all possible palindrome partitioning of

s

.

Example 1:

Input:

s = "aab"

Output:

[["a","a","b"],["aa","b"]]

Example 2:

Input:

s = "a"

Output:

[["a"]]

Constraints:

1 <= s.length <= 16

s

contains only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<string>> Partition(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().
*/
char*** partition(char* s, int* returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Palindrome Partitioning
* Difficulty: Medium
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Palindrome Partitioning
* Difficulty: Medium
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Palindrome Partitioning
Difficulty: Medium
Tags: string, tree, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Palindrome Partitioning
* Difficulty: Medium
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Palindrome Partitioning
* Difficulty: Medium
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Palindrome Partitioning
* Difficulty: Medium
* Tags: string, tree, dp
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Palindrome Partitioning
* Difficulty: Medium
* Tags: string, tree, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* 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().
*/
char*** partition(char* s, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Palindrome Partitioning
// Difficulty: Medium
// Tags: string, tree, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(n) or O(n * m) for DP table

func partition(s string) [][]string {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Palindrome Partitioning
// Difficulty: Medium
// Tags: string, tree, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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