Problem 3458: Select K Disjoint Special Substrings

Problem Information

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

Problem Description

Given a string

s

of length

n

and an integer

k

, determine whether it is possible to select

k

disjoint

special substrings

.

A

special substring

is a

substring

where:

Any character present inside the substring should not appear outside it in the string.

The substring is not the entire string

s

.

Note

that all

k

substrings must be disjoint, meaning they cannot overlap.

Return

true

if it is possible to select

k

such disjoint special substrings; otherwise, return

false

.

Example 1:

Input:

s = "abcdbaefab", k = 2

Output:

true

Explanation:

We can select two disjoint special substrings:

"cd"

and

"ef"

.

"cd"

contains the characters

'c'

and

'd'

, which do not appear elsewhere in

s

.

"ef"

contains the characters

'e'

and

'f'

, which do not appear elsewhere in

s

.

Example 2:

Input:

s = "cdefdc", k = 3

Output:

false

Explanation:

There can be at most 2 disjoint special substrings:

"e"

and

"f"

. Since

k = 3

, the output is

false

.

Example 3:

Input:

s = "abeabe", k = 0

Output:

true

Constraints:

2 <= n == s.length <= 5 * 10

4

0 <= k <= 26

s

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool maxSubstringLength(string s, int k) {

}
};
```

Java:

```
class Solution {
public boolean maxSubstringLength(String s, int k) {

}
}
```

Python3:

```
class Solution:
def maxSubstringLength(self, s: str, k: int) -> bool:
```

Python:

```
class Solution(object):
def maxSubstringLength(self, s, k):
"""
:type s: str
:type k: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @return {boolean}
*/
var maxSubstringLength = function(s, k) {

};
```

TypeScript:

```
function maxSubstringLength(s: string, k: number): boolean {

};
```

C#:

```
public class Solution {
public bool MaxSubstringLength(string s, int k) {
```

```
}
}
```

C:

```
bool maxSubstringLength(char* s, int k) {

}
```

Go:

```
func maxSubstringLength(s string, k int) bool {

}
```

Kotlin:

```
class Solution {
fun maxSubstringLength(s: String, k: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func maxSubstringLength(_ s: String, _ k: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn max_substring_length(s: String, k: i32) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {Boolean}
def max_substring_length(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Boolean
*/
function maxSubstringLength($s, $k) {

}
}
```

Dart:

```
class Solution {
bool maxSubstringLength(String s, int k) {

}
}
```

Scala:

```
object Solution {
def maxSubstringLength(s: String, k: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_substring_length(s :: String.t, k :: integer) :: boolean
def max_substring_length(s, k) do
```

```
end
end
```

Erlang:

```
-spec max_substring_length(S :: unicode:unicode_binary(), K :: integer()) ->
boolean().
max_substring_length(S, K) ->
.
```

Racket:

```
(define/contract (max-substring-length s k)
(-> string? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Select K Disjoint Special Substrings
* Difficulty: Medium
* Tags: string, tree, dp, greedy, hash, sort
*
* 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:
bool maxSubstringLength(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Select K Disjoint Special Substrings
* Difficulty: Medium
* Tags: string, tree, dp, greedy, hash, sort
*
* 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 boolean maxSubstringLength(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Select K Disjoint Special Substrings
Difficulty: Medium
Tags: string, tree, dp, greedy, hash, sort

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 maxSubstringLength(self, s: str, k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxSubstringLength(self, s, k):
"""
:type s: str
:type k: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Select K Disjoint Special Substrings
* Difficulty: Medium
* Tags: string, tree, dp, greedy, hash, sort
*
* 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
* @param {number} k
* @return {boolean}
*/
var maxSubstringLength = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Select K Disjoint Special Substrings
* Difficulty: Medium
* Tags: string, tree, dp, greedy, hash, sort
*
* 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 maxSubstringLength(s: string, k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Select K Disjoint Special Substrings
* Difficulty: Medium
```

```
* Tags: string, tree, dp, greedy, hash, sort
*
* 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 bool MaxSubstringLength(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Select K Disjoint Special Substrings
* Difficulty: Medium
* Tags: string, tree, dp, greedy, hash, sort
*
* 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
*/

bool maxSubstringLength(char* s, int k) {

}
```

Go Solution:

```
// Problem: Select K Disjoint Special Substrings
// Difficulty: Medium
// Tags: string, tree, dp, greedy, hash, sort
//
// 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 maxSubstringLength(s string, k int) bool {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun maxSubstringLength(s: String, k: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func maxSubstringLength(_ s: String, _ k: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Select K Disjoint Special Substrings
// Difficulty: Medium
// Tags: string, tree, dp, greedy, hash, sort
//
// 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 max_substring_length(s: String, k: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {Boolean}
def max_substring_length(s, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Boolean
*/
function maxSubstringLength($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
bool maxSubstringLength(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxSubstringLength(s: String, k: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_substring_length(s :: String.t, k :: integer) :: boolean
def max_substring_length(s, k) do

end
end
```

Erlang Solution:

```
-spec max_substring_length(S :: unicode:unicode_binary(), K :: integer()) ->
boolean().
max_substring_length(S, K) ->
.
```

Racket Solution:

```
(define/contract (max-substring-length s k)
(-> string? exact-integer? boolean?)
)
```