Problem 3003: Maximize the Number of Partitions After Operations

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a string

s

and an integer

k

.

First, you are allowed to change

at most

one

index in

s

to another lowercase English letter.

After that, do the following partitioning operation until

s

is

empty

:

Choose the

longest

prefix

of

s

containing at most

k

distinct

characters.

Delete

the prefix from

s

and increase the number of partitions by one. The remaining characters (if any) in

s

maintain their initial order.

Return an integer denoting the

maximum

number of resulting partitions after the operations by optimally choosing at most one index to change.

Example 1:

Input:

s = "accca", k = 2

Output:

3

Explanation:

The optimal way is to change

s[2]

to something other than a and c, for example, b. then it becomes

"acbca"

.

Then we perform the operations:

The longest prefix containing at most 2 distinct characters is

"ac"

, we remove it and

s

becomes

"bca"

.

Now The longest prefix containing at most 2 distinct characters is

"bc"

, so we remove it and

s

becomes

"a"

.

Finally, we remove

"a"

and

s

becomes empty, so the procedure ends.

Doing the operations, the string is divided into 3 partitions, so the answer is 3.

Example 2:

Input:

s = "aabaab", k = 3

Output:

1

Explanation:

Initially

s

contains 2 distinct characters, so whichever character we change, it will contain at most 3 distinct characters, so the longest prefix with at most 3 distinct characters would always be all of it, therefore the answer is 1.

Example 3:

Input:

s = "xxyz", k = 1

Output:

4

Explanation:

The optimal way is to change

s[0]

or

s[1]

to something other than characters in

s

, for example, to change

s[0]

to

w

.

Then

s

becomes

"wxyz"

, which consists of 4 distinct characters, so as

k

is 1, it will divide into 4 partitions.

Constraints:

1 <= s.length <= 10

4

s

consists only of lowercase English letters.

1 <= k <= 26

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxPartitionsAfterOperations(s: string, k: number): number {

};
```

C#:

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

}
}
```

C:

```
int maxPartitionsAfterOperations(char* s, int k) {

}
```

Go:

```
func maxPartitionsAfterOperations(s string, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int maxPartitionsAfterOperations(String s, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
def max_partitions_after_operations(s, k) do

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-partitions-after-operations s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Number of Partitions After Operations
* Difficulty: Hard
* Tags: string, 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:
int maxPartitionsAfterOperations(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Number of Partitions After Operations
* Difficulty: Hard
* Tags: string, 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 int maxPartitionsAfterOperations(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Number of Partitions After Operations
Difficulty: Hard
Tags: string, 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 maxPartitionsAfterOperations(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize the Number of Partitions After Operations
* Difficulty: Hard
* Tags: string, 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
* @param {number} k
* @return {number}
*/
var maxPartitionsAfterOperations = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Number of Partitions After Operations
* Difficulty: Hard
* Tags: string, 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 maxPartitionsAfterOperations(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Number of Partitions After Operations
* Difficulty: Hard
* Tags: string, 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 int MaxPartitionsAfterOperations(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Number of Partitions After Operations
* Difficulty: Hard
* Tags: string, 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
*/

int maxPartitionsAfterOperations(char* s, int k) {

}
```

Go Solution:

```
// Problem: Maximize the Number of Partitions After Operations
// Difficulty: Hard
// Tags: string, 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 maxPartitionsAfterOperations(s string, k int) int {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize the Number of Partitions After Operations
// Difficulty: Hard
// Tags: string, 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 max_partitions_after_operations(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int maxPartitionsAfterOperations(String s, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-partitions-after-operations s k)
(-> string? exact-integer? exact-integer?)
)
```