Problem 3639: Minimum Time to Activate String

Problem Information

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

Problem Description

You are given a string

s

of length

n

and an integer array

order

, where

order

is a

permutation

of the numbers in the range

[0, n - 1]

.

Starting from time

t = 0

, replace the character at index

order[t]

in

s

with

'*'

at each time step.

A

substring

is

valid

if it contains

at least

one

'*'

.

A string is

active

if the total number of

valid

substrings is greater than or equal to

k

.

Return the

minimum

time

t

at which the string

s

becomes

active

. If it is impossible, return -1.

Example 1:

Input:

s = "abc", order = [1,0,2], k = 2

Output:

0

Explanation:

t

order[t]

Modified

s

Valid Substrings

Count

Active

(Count >= k)

0

1

"a*c"

"*"

,

"a*"

,

"*c"

,

"a*c"

4

Yes

The string

s

becomes active at

t = 0

. Thus, the answer is 0.

Example 2:

Input:

s = "cat", order = [0,2,1], k = 6

Output:

2

Explanation:

t

order[t]

Modified

s

Valid Substrings

Count

Active

(Count >= k)

0

0

"*at"

"*"

,

"*a"

,

"*at"

3

No

1

2

"*a*"

"*"

,

"*a"

,

"

a"

,

"

a*"

,

"*"

5

No

2

1

"***"

All substrings (contain

'*'

)

6

Yes

The string

s

becomes active at

t = 2

. Thus, the answer is 2.

Example 3:

Input:

s = "xy", order = [0,1], k = 4

Output:

-1

Explanation:

Even after all replacements, it is impossible to obtain

k = 4

valid substrings. Thus, the answer is -1.

Constraints:

1 <= n == s.length <= 10

5

order.length == n

0 <= order[i] <= n - 1

s

consists of lowercase English letters.

order

is a permutation of integers from 0 to

n - 1

.

1 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minTime(string s, vector<int>& order, int k) {

}
};
```

Java:

```
class Solution {
public int minTime(String s, int[] order, int k) {

}
}
```

Python3:

```
class Solution:
def minTime(self, s: str, order: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def minTime(self, s, order, k):
"""
:type s: str
:type order: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[]} order
* @param {number} k
* @return {number}
*/
var minTime = function(s, order, k) {

};
```

TypeScript:

```
function minTime(s: string, order: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinTime(string s, int[] order, int k) {

}
}
```

C:

```
int minTime(char* s, int* order, int orderSize, int k) {

}
```

Go:

```
func minTime(s string, order []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minTime(s: String, order: IntArray, k: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func minTime(_ s: String, _ order: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_time(s: String, order: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[]} order
# @param {Integer} k
# @return {Integer}
def min_time(s, order, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[] $order
* @param Integer $k
* @return Integer
*/
function minTime($s, $order, $k) {

}
```

```
}
```

Dart:

```
class Solution {
int minTime(String s, List<int> order, int k) {

}
}
```

Scala:

```
object Solution {
def minTime(s: String, order: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_time(s :: String.t, order :: [integer], k :: integer) :: integer
def min_time(s, order, k) do

end
end
```

Erlang:

```
-spec min_time(S :: unicode:unicode_binary(), Order :: [integer()], K ::
integer()) -> integer().
min_time(S, Order, K) ->
.
```

Racket:

```
(define/contract (min-time s order k)
(-> string? (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time to Activate String
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minTime(string s, vector<int>& order, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time to Activate String
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minTime(String s, int[] order, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time to Activate String
```

```
Difficulty: Medium
Tags: array, string, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

```
class Solution(object):
def minTime(self, s, order, k):
"""
:type s: str
:type order: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Time to Activate String
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} s
* @param {number[]} order
* @param {number} k
* @return {number}
```

```
*/
var minTime = function(s, order, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time to Activate String
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minTime(s: string, order: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time to Activate String
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinTime(string s, int[] order, int k) {

}
}
```

C Solution:

```c
/*
* Problem: Minimum Time to Activate String
* Difficulty: Medium
* Tags: array, string, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minTime(char* s, int* order, int orderSize, int k) {

}
```

Go Solution:

```go
// Problem: Minimum Time to Activate String
// Difficulty: Medium
// Tags: array, string, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minTime(s string, order []int, k int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun minTime(s: String, order: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func minTime(_ s: String, _ order: [Int], _ k: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Minimum Time to Activate String
// Difficulty: Medium
// Tags: array, string, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn min_time(s: String, order: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[]} order
# @param {Integer} k
# @return {Integer}
def min_time(s, order, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[] $order
* @param Integer $k
* @return Integer
*/
function minTime($s, $order, $k) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int minTime(String s, List<int> order, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minTime(s: String, order: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_time(s :: String.t, order :: [integer], k :: integer) :: integer
def min_time(s, order, k) do

end
end
```

Erlang Solution:

```
-spec min_time(S :: unicode:unicode_binary(), Order :: [integer()], K ::
integer()) -> integer().
min_time(S, Order, K) ->
.
```

Racket Solution:

```
(define/contract (min-time s order k)
(-> string? (listof exact-integer?) exact-integer? exact-integer?)
)
```