Problem 2450: Number of Distinct Binary Strings After Applying Operations

Problem Information

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

Problem Description

You are given a

binary

string

s

and a positive integer

k

.

You can apply the following operation on the string

any

number of times:

Choose any substring of size

k

from

s

and

flip

all its characters, that is, turn all

1

's into

0

's, and all

0

's into

1

's.

Return

the number of

distinct

strings you can obtain

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Note

that:

A binary string is a string that consists

only

of the characters

0

and

1

.

A substring is a contiguous part of a string.

Example 1:

Input:

s = "1001", k = 3

Output:

4

Explanation:

We can obtain the following strings: - Applying no operation on the string gives s = "1001". - Applying one operation on the substring starting at index 0 gives s = "

011

1". - Applying one operation on the substring starting at index 1 gives s = "1

110

". - Applying one operation on both the substrings starting at indices 0 and 1 gives s = "

0000

". It can be shown that we cannot obtain any other string, so the answer is 4.

Example 2:

Input:

s = "10110", k = 5

Output:

2

Explanation:

We can obtain the following strings: - Applying no operation on the string gives s = "10110". - Applying one operation on the whole string gives s = "01001". It can be shown that we cannot obtain any other string, so the answer is 2.

Constraints:

1 <= k <= s.length <= 10

5

s[i]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def countDistinctStrings(self, s, k):
"""
:type s: str
```

```
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-distinct-strings s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Distinct Binary Strings After Applying Operations
* Difficulty: Medium
* Tags: string, tree, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Number of Distinct Binary Strings After Applying Operations
* Difficulty: Medium
* Tags: string, tree, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int countDistinctStrings(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Distinct Binary Strings After Applying Operations
```

```
Difficulty: Medium
Tags: string, tree, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def countDistinctStrings(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Distinct Binary Strings After Applying Operations
* Difficulty: Medium
* Tags: string, tree, math
*
* Approach: String manipulation with hash map or two pointers
* 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} k
* @return {number}
*/
var countDistinctStrings = function(s, k) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of Distinct Binary Strings After Applying Operations
* Difficulty: Medium
* Tags: string, tree, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function countDistinctStrings(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Distinct Binary Strings After Applying Operations
* Difficulty: Medium
* Tags: string, tree, math
*
* Approach: String manipulation with hash map or two pointers
* 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 CountDistinctStrings(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Distinct Binary Strings After Applying Operations
```

```
* Difficulty: Medium
* Tags: string, tree, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int countDistinctStrings(char* s, int k) {

}
```

Go Solution:

```
// Problem: Number of Distinct Binary Strings After Applying Operations
// Difficulty: Medium
// Tags: string, tree, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countDistinctStrings(s string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Distinct Binary Strings After Applying Operations
// Difficulty: Medium
// Tags: string, tree, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-distinct-strings s k)
(-> string? exact-integer? exact-integer?)
)
```