Problem 3666: Minimum Operations to Equalize Binary String

Problem Information

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

Problem Description

You are given a binary string

s

, and an integer

k

.

In one operation, you must choose

exactly

k

different

indices and

flip

each

'0'

to

'1'

and each

'1'

to

'0'

.

Return the

minimum

number of operations required to make all characters in the string equal to

'1'

. If it is not possible, return -1.

Example 1:

Input:

s = "110", k = 1

Output:

1

Explanation:

There is one

'0'

in

s

.

Since

k = 1

, we can flip it directly in one operation.

Example 2:

Input:

s = "0101", k = 3

Output:

2

Explanation:

One optimal set of operations choosing

k = 3

indices in each operation is:

Operation 1

: Flip indices

[0, 1, 3]

.

s

changes from

"0101"

to

"1000"

.

Operation 2

: Flip indices

[1, 2, 3]

.

s

changes from

"1000"

to

"1111"

.

Thus, the minimum number of operations is 2.

Example 3:

Input:

s = "101", k = 2

Output:

-1

Explanation:

Since

k = 2

and

s

has only one

'0'

, it is impossible to flip exactly

k

indices to make all

'1'

. Hence, the answer is -1.

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'0'

or

'1'

.

1 <= k <= s.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-operations s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Equalize Binary String
* Difficulty: Hard
* Tags: string, graph, math, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Equalize Binary String
* Difficulty: Hard
* Tags: string, graph, math, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Equalize Binary String
```

```
Difficulty: Hard
Tags: string, graph, math, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Equalize Binary String
* Difficulty: Hard
* Tags: string, graph, math, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Equalize Binary String
* Difficulty: Hard
* Tags: string, graph, math, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Equalize Binary String
* Difficulty: Hard
* Tags: string, graph, math, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinOperations(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Equalize Binary String
```

```
* Difficulty: Hard
* Tags: string, graph, math, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int minOperations(char* s, int k) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Equalize Binary String
// Difficulty: Hard
// Tags: string, graph, math, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func minOperations(s string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Equalize Binary String
// Difficulty: Hard
// Tags: string, graph, math, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-operations s k)
(-> string? exact-integer? exact-integer?)
)
```