Problem 1758: Minimum Changes To Make Alternating Binary String

Problem Information

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

Problem Description

You are given a string

s

consisting only of the characters

'0'

and

'1'

. In one operation, you can change any

'0'

to

'1'

or vice versa.

The string is called alternating if no two adjacent characters are equal. For example, the string

"010"

is alternating, while the string

"0100"

is not.

Return

the

minimum

number of operations needed to make

s

alternating

.

Example 1:

Input:

s = "0100"

Output:

1

Explanation:

If you change the last character to '1', s will be "0101", which is alternating.

Example 2:

Input:

s = "10"

Output:

0

Explanation:

s is already alternating.

Example 3:

Input:

s = "1111"

Output:

2

Explanation:

You need two operations to reach "0101" or "1010".

Constraints:

1 <= s.length <= 10

4

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
```

```
var minOperations = function(s) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minOperations(char* s) {

}
```

Go:

```
func minOperations(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Changes To Make Alternating Binary String
* Difficulty: Easy
* Tags: string
*
* 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) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Changes To Make Alternating Binary String
* Difficulty: Easy
* Tags: string
*
* 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) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Changes To Make Alternating Binary String
Difficulty: Easy
Tags: string

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) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Changes To Make Alternating Binary String
* Difficulty: Easy
* Tags: string
*
* 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
* @return {number}
*/
var minOperations = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Changes To Make Alternating Binary String
* Difficulty: Easy
* Tags: string
*
* 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): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Changes To Make Alternating Binary String
* Difficulty: Easy
* Tags: string
*
* 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) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Changes To Make Alternating Binary String
* Difficulty: Easy
* Tags: string
*
* 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) {

}
```

Go Solution:

```
// Problem: Minimum Changes To Make Alternating Binary String
// Difficulty: Easy
```

```
// Tags: string
//
// 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) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Changes To Make Alternating Binary String
// Difficulty: Easy
// Tags: string
//
// 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) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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