Problem 926: Flip String to Monotone Increasing

Problem Information

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

Problem Description

A binary string is monotone increasing if it consists of some number of

0

's (possibly none), followed by some number of

1

's (also possibly none).

You are given a binary string

s

. You can flip

s[i]

changing it from

0

to

1

or from

1

to

0

.

Return

the minimum number of flips to make

s

monotone increasing

.

Example 1:

Input:

s = "00110"

Output:

1

Explanation:

We flip the last digit to get 00111.

Example 2:

Input:

s = "010110"

Output:

2

Explanation:

We flip to get 011111, or alternatively 000111.

Example 3:

Input:

s = "00011000"

Output:

2

Explanation:

We flip to get 00000000.

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var minFlipsMonoIncr = function(s) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minFlipsMonoIncr(char* s) {

}
```

Go:

```
func minFlipsMonoIncr(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-flips-mono-incr s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Flip String to Monotone Increasing
* Difficulty: Medium
* 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 minFlipsMonoIncr(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Flip String to Monotone Increasing
* Difficulty: Medium
* 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 minFlipsMonoIncr(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Flip String to Monotone Increasing
Difficulty: Medium
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 minFlipsMonoIncr(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Flip String to Monotone Increasing
* Difficulty: Medium
* 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
* @return {number}
*/
var minFlipsMonoIncr = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Flip String to Monotone Increasing
* Difficulty: Medium
* 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 minFlipsMonoIncr(s: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Flip String to Monotone Increasing
* Difficulty: Medium
* 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 MinFlipsMonoIncr(string s) {

}
}
```

C Solution:

```
/*
* Problem: Flip String to Monotone Increasing
* Difficulty: Medium
* 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 minFlipsMonoIncr(char* s) {

}
```

Go Solution:

```
// Problem: Flip String to Monotone Increasing
// Difficulty: Medium
```

```
// 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 minFlipsMonoIncr(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Flip String to Monotone Increasing
// Difficulty: Medium
// 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 min_flips_mono_incr(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-flips-mono-incr s)
(-> string? exact-integer?)
)
```