Problem 2914: Minimum Number of Changes to Make Binary String Beautiful

Problem Information

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

Problem Description

You are given a

0-indexed

binary string

s

having an even length.

A string is

beautiful

if it's possible to partition it into one or more substrings such that:

Each substring has an

even length

.

Each substring contains

only

1

's or

only

0

's.

You can change any character in

s

to

0

or

1

.

Return

the

minimum

number of changes required to make the string

s

beautiful

.

Example 1:

Input:

s = "1001"

Output:

2

Explanation:

We change s[1] to 1 and s[3] to 0 to get string "1100". It can be seen that the string "1100" is beautiful because we can partition it into "11|00". It can be proven that 2 is the minimum number of changes needed to make the string beautiful.

Example 2:

Input:

s = "10"

Output:

1

Explanation:

We change s[1] to 1 to get string "11". It can be seen that the string "11" is beautiful because we can partition it into "11". It can be proven that 1 is the minimum number of changes needed to make the string beautiful.

Example 3:

Input:

s = "0000"

Output:

0

Explanation:

We don't need to make any changes as the string "0000" is beautiful already.

Constraints:

2 <= s.length <= 10

5

s

has an even length.

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minChanges(char* s) {

}
```

Go:

```
func minChanges(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Changes to Make Binary String Beautiful
* Difficulty: Medium
* Tags: string, tree
*
* 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 minChanges(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Changes to Make Binary String Beautiful
* Difficulty: Medium
* Tags: string, tree
*
* 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 minChanges(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Changes to Make Binary String Beautiful
Difficulty: Medium
Tags: string, tree

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 minChanges(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Changes to Make Binary String Beautiful
* Difficulty: Medium
* Tags: string, tree
*
* 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
* @return {number}
*/
var minChanges = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Changes to Make Binary String Beautiful
* Difficulty: Medium
* Tags: string, tree
*
* 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 minChanges(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Changes to Make Binary String Beautiful
* Difficulty: Medium
* Tags: string, tree
*
* 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 MinChanges(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Changes to Make Binary String Beautiful
* Difficulty: Medium
* Tags: string, tree
*
* 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 minChanges(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Number of Changes to Make Binary String Beautiful
// Difficulty: Medium
// Tags: string, tree
//
// 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 minChanges(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Changes to Make Binary String Beautiful
// Difficulty: Medium
// Tags: string, tree
//
// 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 min_changes(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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