Problem 1896: Minimum Cost to Change the Final Value of Expression

Problem Information

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

Problem Description

You are given a

valid

boolean expression as a string

expression

consisting of the characters

'1'

,

'0'

,

'&'

(bitwise

AND

operator),

'|'

(bitwise

OR

operator),

'('

, and

')'

.

For example,

"()1|1"

and

"(1)&()"

are

not valid

while

"1"

,

"(((1))|(0))"

, and

"1|(0&(1))"

are

valid

expressions.

Return

the

minimum cost

to change the final value of the expression

.

For example, if

expression = "1|1|(0&0)&1"

, its

value

is

1|1|(0&0)&1 = 1|1|0&1 = 1|0&1 = 1&1 = 1

. We want to apply operations so that the

new

expression evaluates to

0

.

The

cost

of changing the final value of an expression is the

number of operations

performed on the expression. The types of

operations

are described as follows:

Turn a

'1'

into a

'0'

.

Turn a

'0'

into a

'1'

.

Turn a

'&'

into a

'|'

.

Turn a

'|'

into a

'&'

.

Note:

'&'

does

not

take precedence over

'|'

in the

order of calculation

. Evaluate parentheses

first

, then in

left-to-right

order.

Example 1:

Input:

expression = "1&(0|1)"

Output:

1

Explanation:

We can turn "1&(0

|

1)" into "1&(0

&

1)" by changing the '|' to a '&' using 1 operation. The new expression evaluates to 0.

Example 2:

Input:

expression = "(0&0)&(0&0&0)"

Output:

3

Explanation:

We can turn "(0

&0

)

&

(0&0&0)" into "(0

|1

)

|

(0&0&0)" using 3 operations. The new expression evaluates to 1.

Example 3:

Input:

expression = "(0|(1|0&1))"

Output:

1

Explanation:

We can turn "(0|(

1

|0&1))" into "(0|(

0

|0&1))" using 1 operation. The new expression evaluates to 0.

Constraints:

1 <= expression.length <= 10

5

expression

only contains

'1'

,

'0'

,

'&'

,

'|'

,

'('

, and

')'

All parentheses are properly matched.

There will be no empty parentheses (i.e:

"()"

is not a substring of

expression

).

Code Snippets

C++:

```
class Solution {
public:
int minOperationsToFlip(string expression) {

}
};
```

Java:

```
class Solution {
public int minOperationsToFlip(String expression) {

}
}
```

Python3:

```
class Solution:
def minOperationsToFlip(self, expression: str) -> int:
```

Python:

```
class Solution(object):
def minOperationsToFlip(self, expression):
"""
:type expression: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} expression
* @return {number}
*/
var minOperationsToFlip = function(expression) {

};
```

TypeScript:

```
function minOperationsToFlip(expression: string): number {

};
```

C#:

```
public class Solution {
public int MinOperationsToFlip(string expression) {

}
}
```

C:

```
int minOperationsToFlip(char* expression) {

}
```

Go:

```
func minOperationsToFlip(expression string) int {

}
```

Kotlin:

```
class Solution {
fun minOperationsToFlip(expression: String): Int {

}
}
```

Swift:

```
class Solution {
func minOperationsToFlip(_ expression: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations_to_flip(expression: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} expression
# @return {Integer}
def min_operations_to_flip(expression)

end
```

PHP:

```
class Solution {

/**
* @param String $expression
* @return Integer
*/
function minOperationsToFlip($expression) {

}
}
```

Dart:

```
class Solution {
int minOperationsToFlip(String expression) {

}
}
```

Scala:

```
object Solution {
def minOperationsToFlip(expression: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations_to_flip(expression :: String.t) :: integer
def min_operations_to_flip(expression) do

end
end
```

Erlang:

```
-spec min_operations_to_flip(Expression :: unicode:unicode_binary()) ->
integer().
min_operations_to_flip(Expression) ->
.
```

Racket:

```
(define/contract (min-operations-to-flip expression)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Change the Final Value of Expression
* Difficulty: Hard
* Tags: string, tree, dp, math, stack
*
* 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 minOperationsToFlip(string expression) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Change the Final Value of Expression
* Difficulty: Hard
* Tags: string, tree, dp, math, stack
*
* 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 minOperationsToFlip(String expression) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Change the Final Value of Expression
Difficulty: Hard
Tags: string, tree, dp, math, stack

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 minOperationsToFlip(self, expression: str) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperationsToFlip(self, expression):
"""
:type expression: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Change the Final Value of Expression
* Difficulty: Hard
* Tags: string, tree, dp, math, stack
*
* 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} expression
* @return {number}
*/
var minOperationsToFlip = function(expression) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Change the Final Value of Expression
* Difficulty: Hard
* Tags: string, tree, dp, math, stack
*
* 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 minOperationsToFlip(expression: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Change the Final Value of Expression
* Difficulty: Hard
* Tags: string, tree, dp, math, stack
*
* 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 MinOperationsToFlip(string expression) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Change the Final Value of Expression
* Difficulty: Hard
* Tags: string, tree, dp, math, stack
*
* 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 minOperationsToFlip(char* expression) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Change the Final Value of Expression
// Difficulty: Hard
// Tags: string, tree, dp, math, stack
//
// 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 minOperationsToFlip(expression string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperationsToFlip(expression: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperationsToFlip(_ expression: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Change the Final Value of Expression
// Difficulty: Hard
// Tags: string, tree, dp, math, stack
//
// 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_operations_to_flip(expression: String) -> i32 {


}
}
```

Ruby Solution:

```
# @param {String} expression
# @return {Integer}
def min_operations_to_flip(expression)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $expression
* @return Integer
*/
function minOperationsToFlip($expression) {

}
}
```

Dart Solution:

```
class Solution {
int minOperationsToFlip(String expression) {

}
}
```

Scala Solution:

```
object Solution {
def minOperationsToFlip(expression: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations_to_flip(expression :: String.t) :: integer
def min_operations_to_flip(expression) do

end
end
```

Erlang Solution:

```
-spec min_operations_to_flip(Expression :: unicode:unicode_binary()) ->
integer().
min_operations_to_flip(Expression) ->
.
```

Racket Solution:

```
(define/contract (min-operations-to-flip expression)
(-> string? exact-integer?)
)
```