Problem 2896: Apply Operations to Make Two Strings Equal

Problem Information

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

Problem Description

You are given two

0-indexed

binary strings

s1

and

s2

, both of length

n

, and a positive integer

x

.

You can perform any of the following operations on the string

s1

any

number of times:

Choose two indices

i

and

j

, and flip both

s1[i]

and

s1[j]

. The cost of this operation is

x

.

Choose an index

i

such that

i < n - 1

and flip both

s1[i]

and

s1[i + 1]

. The cost of this operation is

1

.

Return

the

minimum

cost needed to make the strings

s1

and

s2

equal, or return

-1

if it is impossible.

Note

that flipping a character means changing it from

0

to

1

or vice-versa.

Example 1:

Input:

s1 = "1100011000", s2 = "0101001010", x = 2

Output:

4

Explanation:

We can do the following operations: - Choose i = 3 and apply the second operation. The resulting string is s1 = "110

11

11000". - Choose i = 4 and apply the second operation. The resulting string is s1 = "1101

00

1000". - Choose i = 0 and j = 8 and apply the first operation. The resulting string is s1 = "

0

1010010

1

0" = s2. The total cost is 1 + 1 + 2 = 4. It can be shown that it is the minimum cost possible.

Example 2:

Input:

s1 = "10110", s2 = "00011", x = 4

Output:

-1

Explanation:

It is not possible to make the two strings equal.

Constraints:

n == s1.length == s2.length

1 <= n, x <= 500

s1

and

s2

consist only of the characters

'0'

and

'1'

.

Code Snippets

C++:

```
class Solution {
public:
int minOperations(string s1, string s2, int x) {
}
};
```

Java:

```
class Solution {
public int minOperations(String s1, String s2, int x) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, s1: str, s2: str, x: int) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, s1, s2, x):
"""
:type s1: str
:type s2: str
:type x: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s1
* @param {string} s2
* @param {number} x
* @return {number}
*/
var minOperations = function(s1, s2, x) {

};
```

TypeScript:

```
function minOperations(s1: string, s2: string, x: number): number {

};
```

C#:

```
public class Solution {
public int MinOperations(string s1, string s2, int x) {

}
}
```

C:

```
int minOperations(char* s1, char* s2, int x) {

}
```

Go:

```
func minOperations(s1 string, s2 string, x int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(s1: String, s2: String, x: Int): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ s1: String, _ s2: String, _ x: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(s1: String, s2: String, x: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s1
# @param {String} s2
# @param {Integer} x
# @return {Integer}
def min_operations(s1, s2, x)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @param Integer $x
* @return Integer
*/
function minOperations($s1, $s2, $x) {

}
}
```

Dart:

```
class Solution {
int minOperations(String s1, String s2, int x) {

}
}
```

Scala:

```
object Solution {
def minOperations(s1: String, s2: String, x: Int): Int = {
}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(s1 :: String.t, s2 :: String.t, x :: integer) :: integer
def min_operations(s1, s2, x) do

end
end
```

Erlang:

```
-spec min_operations(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), X :: integer()) -> integer().
min_operations(S1, S2, X) ->
.
```

Racket:

```
(define/contract (min-operations s1 s2 x)
(-> string? string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Apply Operations to Make Two Strings Equal
* 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 minOperations(string s1, string s2, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Apply Operations to Make Two Strings Equal
* 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 minOperations(String s1, String s2, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Apply Operations to Make Two Strings Equal
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 minOperations(self, s1: str, s2: str, x: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, s1, s2, x):
"""
:type s1: str
:type s2: str
:type x: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Apply Operations to Make Two Strings Equal
* 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} s1
* @param {string} s2
* @param {number} x
* @return {number}
*/
var minOperations = function(s1, s2, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Operations to Make Two Strings Equal
* 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 minOperations(s1: string, s2: string, x: number): number {

};
```

C# Solution:

```
/*
* Problem: Apply Operations to Make Two Strings Equal
* 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 MinOperations(string s1, string s2, int x) {

}
}
```

C Solution:

```
/*
* Problem: Apply Operations to Make Two Strings Equal
* 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 minOperations(char* s1, char* s2, int x) {

}
```

Go Solution:

```
// Problem: Apply Operations to Make Two Strings Equal
// 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 minOperations(s1 string, s2 string, x int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(s1: String, s2: String, x: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ s1: String, _ s2: String, _ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Apply Operations to Make Two Strings Equal
// 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_operations(s1: String, s2: String, x: i32) -> i32 {
}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {String} s2
# @param {Integer} x
# @return {Integer}
def min_operations(s1, s2, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @param Integer $x
* @return Integer
*/
function minOperations($s1, $s2, $x) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(String s1, String s2, int x) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(s1: String, s2: String, x: Int): Int = {



}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(s1 :: String.t, s2 :: String.t, x :: integer) :: integer
def min_operations(s1, s2, x) do

end
end
```

Erlang Solution:

```
-spec min_operations(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), X :: integer()) -> integer().
min_operations(S1, S2, X) ->
.
```

Racket Solution:

```
(define/contract (min-operations s1 s2 x)
(-> string? string? exact-integer? exact-integer?)
)
```