Problem 2546: Apply Bitwise Operations to Make Strings Equal

Problem Information

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

Problem Description

You are given two

0-indexed binary

strings

s

and

target

of the same length

n

. You can do the following operation on

s

any

number of times:

Choose two

different

indices

i

and

j

where

0 <= i, j < n

.

Simultaneously, replace

s[i]

with (

s[i]

OR

s[j]

) and

s[j]

with (

s[i]

XOR

s[j]

).

For example, if

s = "0110"

, you can choose

i = 0

and

j = 2

, then simultaneously replace

s[0]

with (

s[0]

OR

s[2]

=

0

OR

1

=

1

), and

s[2]

with (

s[0]

XOR

s[2]

=

0

XOR

1

=

1

), so we will have

s = "1110"

.

Return

true

if you can make the string

s

equal to

target

, or

false

otherwise

.

Example 1:

Input:

s = "1010", target = "0110"

Output:

true

Explanation:

We can do the following operations: - Choose i = 2 and j = 0. We have now s = "

0

0

1

0". - Choose i = 2 and j = 1. We have now s = "0

11

0". Since we can make s equal to target, we return true.

Example 2:

Input:

s = "11", target = "00"

Output:

false

Explanation:

It is not possible to make s equal to target with any number of operations.

Constraints:

n == s.length == target.length

2 <= n <= 10

5

s

and

target

consist of only the digits

0

and

1

.

Code Snippets

C++:

```
class Solution {
public:
bool makeStringsEqual(string s, string target) {

}
};
```

Java:

```
class Solution {
public boolean makeStringsEqual(String s, String target) {

}
}
```

Python3:

```
class Solution:
def makeStringsEqual(self, s: str, target: str) -> bool:
```

Python:

```
class Solution(object):
def makeStringsEqual(self, s, target):
"""
:type s: str
:type target: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} target
* @return {boolean}
*/
var makeStringsEqual = function(s, target) {

};
```

}

TypeScript:

```
function makeStringsEqual(s: string, target: string): boolean {

};
```

C#:

```
public class Solution {
public bool MakeStringsEqual(string s, string target) {

}
}
```

C:

```
bool makeStringsEqual(char* s, char* target) {

}
```

Go:

```
func makeStringsEqual(s string, target string) bool {

}
```

Kotlin:

```
class Solution {
fun makeStringsEqual(s: String, target: String): Boolean {

}
}
```

Swift:

```
class Solution {
func makeStringsEqual(_ s: String, _ target: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn make_strings_equal(s: String, target: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} target
# @return {Boolean}
def make_strings_equal(s, target)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $target
* @return Boolean
*/
function makeStringsEqual($s, $target) {

}
}
```

Dart:

```
class Solution {
bool makeStringsEqual(String s, String target) {

}
}
```

Scala:

```
object Solution {
def makeStringsEqual(s: String, target: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec make_strings_equal(s :: String.t, target :: String.t) :: boolean
def make_strings_equal(s, target) do

end
end
```

Erlang:

```
-spec make_strings_equal(S :: unicode:unicode_binary(), Target ::
unicode:unicode_binary()) -> boolean().
make_strings_equal(S, Target) ->
.
```

Racket:

```
(define/contract (make-strings-equal s target)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Apply Bitwise Operations to Make Strings Equal
* Difficulty: Medium
* 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:
```

```
bool makeStringsEqual(string s, string target) {

}
};
```

Java Solution:

```
/**
* Problem: Apply Bitwise Operations to Make Strings Equal
* Difficulty: Medium
* 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 boolean makeStringsEqual(String s, String target) {


}
}
```

Python3 Solution:

```
"""
Problem: Apply Bitwise Operations to Make Strings Equal
Difficulty: Medium
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 makeStringsEqual(self, s: str, target: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def makeStringsEqual(self, s, target):
"""
:type s: str
:type target: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Apply Bitwise Operations to Make Strings Equal
* Difficulty: Medium
* 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
* @param {string} target
* @return {boolean}
*/
var makeStringsEqual = function(s, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Apply Bitwise Operations to Make Strings Equal
* Difficulty: Medium
* 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 makeStringsEqual(s: string, target: string): boolean {
};
```

C# Solution:

```
/*
* Problem: Apply Bitwise Operations to Make Strings Equal
* Difficulty: Medium
* 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 bool MakeStringsEqual(string s, string target) {

}
}
```

C Solution:

```
/*
* Problem: Apply Bitwise Operations to Make Strings Equal
* Difficulty: Medium
* 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
*/

bool makeStringsEqual(char* s, char* target) {

}
```

Go Solution:

```
// Problem: Apply Bitwise Operations to Make Strings Equal
// Difficulty: Medium
// 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 makeStringsEqual(s string, target string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun makeStringsEqual(s: String, target: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func makeStringsEqual(_ s: String, _ target: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Apply Bitwise Operations to Make Strings Equal
// Difficulty: Medium
// 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 make_strings_equal(s: String, target: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} target
# @return {Boolean}
def make_strings_equal(s, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $target
* @return Boolean
*/
function makeStringsEqual($s, $target) {

}
}
```

Dart Solution:

```
class Solution {
bool makeStringsEqual(String s, String target) {

}
}
```

Scala Solution:

```
object Solution {
def makeStringsEqual(s: String, target: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec make_strings_equal(s :: String.t, target :: String.t) :: boolean
def make_strings_equal(s, target) do

end
end
```

Erlang Solution:

```
-spec make_strings_equal(S :: unicode:unicode_binary(), Target ::
unicode:unicode_binary()) -> boolean().
make_strings_equal(S, Target) ->
.
```

Racket Solution:

```
(define/contract (make-strings-equal s target)
(-> string? string? boolean?)
)
```