Problem 3226: Number of Bit Changes to Make Two Integers Equal

Problem Information

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

Problem Description

You are given two positive integers

n

and

k

.

You can choose

any

bit in the

binary representation

of

n

that is equal to 1 and change it to 0.

Return the

number of changes

needed to make

n

equal to

k

. If it is impossible, return -1.

Example 1:

Input:

n = 13, k = 4

Output:

2

Explanation:

Initially, the binary representations of

n

and

k

are

n = (1101)

2

and

k = (0100)

2

.

We can change the first and fourth bits of

n

. The resulting integer is

n = (

0

10

0

)

2

= k

.

Example 2:

Input:

n = 21, k = 21

Output:

0

Explanation:

n

and

k

are already equal, so no changes are needed.

Example 3:

Input:

n = 14, k = 13

Output:

-1

Explanation:

It is not possible to make

n

equal to

k

.

Constraints:

1 <= n, k <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int minChanges(int n, int k) {

}
};
```

Java:

```
class Solution {
public int minChanges(int n, int k) {

}
}
```

Python3:

```
class Solution:
def minChanges(self, n: int, k: int) -> int:
```

Python:

```
class Solution(object):
def minChanges(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var minChanges = function(n, k) {
};
```

TypeScript:

```
function minChanges(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int MinChanges(int n, int k) {

}
}
```

C:

```
int minChanges(int n, int k) {

}
```

Go:

```
func minChanges(n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minChanges(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minChanges(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_changes(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def min_changes(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function minChanges($n, $k) {

}
}
```

Dart:

```
class Solution {
int minChanges(int n, int k) {

}
}
```

Scala:

```
object Solution {
def minChanges(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_changes(n :: integer, k :: integer) :: integer
def min_changes(n, k) do

end
end
```

Erlang:

```
-spec min_changes(N :: integer(), K :: integer()) -> integer().
min_changes(N, K) ->
.
```

Racket:

```
(define/contract (min-changes n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Bit Changes to Make Two Integers Equal
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int minChanges(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Bit Changes to Make Two Integers Equal
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int minChanges(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Bit Changes to Make Two Integers Equal
Difficulty: Easy
Tags: general

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def minChanges(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def minChanges(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Bit Changes to Make Two Integers Equal
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var minChanges = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Bit Changes to Make Two Integers Equal
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function minChanges(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Bit Changes to Make Two Integers Equal
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MinChanges(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Bit Changes to Make Two Integers Equal
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int minChanges(int n, int k) {

}
```

Go Solution:

```
// Problem: Number of Bit Changes to Make Two Integers Equal
// Difficulty: Easy
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func minChanges(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minChanges(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minChanges(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Bit Changes to Make Two Integers Equal
// Difficulty: Easy
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn min_changes(n: i32, k: i32) -> i32 {
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def min_changes(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function minChanges($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minChanges(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minChanges(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_changes(n :: integer, k :: integer) :: integer
def min_changes(n, k) do

end
end
```

Erlang Solution:

```
-spec min_changes(N :: integer(), K :: integer()) -> integer().
min_changes(N, K) ->
.
```

Racket Solution:

```
(define/contract (min-changes n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```