Problem 2220: Minimum Bit Flips to Convert Number

Problem Information

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

Problem Description

A

bit flip

of a number

x

is choosing a bit in the binary representation of

x

and

flipping

it from either

0

to

1

or

1

to

0

.

For example, for

x = 7

, the binary representation is

111

and we may choose any bit (including any leading zeros not shown) and flip it. We can flip the first bit from the right to get

110

, flip the second bit from the right to get

101

, flip the fifth bit from the right (a leading zero) to get

10111

, etc.

Given two integers

start

and

goal

, return

the

minimum

number of

bit flips

to convert

start

to

goal

.

Example 1:

Input:

start = 10, goal = 7

Output:

3

Explanation:

The binary representation of 10 and 7 are 1010 and 0111 respectively. We can convert 10 to 7 in 3 steps: - Flip the first bit from the right: 101

0

-> 101

1

. - Flip the third bit from the right: 1

0

11 -> 1

1

11. - Flip the fourth bit from the right:

1

111 ->

0

111. It can be shown we cannot convert 10 to 7 in less than 3 steps. Hence, we return 3.

Example 2:

Input:

start = 3, goal = 4

Output:

3

Explanation:

The binary representation of 3 and 4 are 011 and 100 respectively. We can convert 3 to 4 in 3 steps: - Flip the first bit from the right: 01

1

-> 01

0

. - Flip the second bit from the right: 0

1

0 -> 0

0

0. - Flip the third bit from the right:

0

00 ->

1

00. It can be shown we cannot convert 3 to 4 in less than 3 steps. Hence, we return 3.

Constraints:

0 <= start, goal <= 10

9

Note:

This question is the same as

461: Hamming Distance.

Code Snippets

C++:

```
class Solution {
public:
```

```
int minBitFlips(int start, int goal) {

}
};
```

Java:

```
class Solution {
public int minBitFlips(int start, int goal) {

}
}
```

Python3:

```
class Solution:
def minBitFlips(self, start: int, goal: int) -> int:
```

Python:

```
class Solution(object):
def minBitFlips(self, start, goal):
"""
:type start: int
:type goal: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} start
* @param {number} goal
* @return {number}
*/
var minBitFlips = function(start, goal) {

};
```

TypeScript:

```
function minBitFlips(start: number, goal: number): number {

};
```

C#:

```
public class Solution {
public int MinBitFlips(int start, int goal) {

}
}
```

C:

```
int minBitFlips(int start, int goal) {

}
```

Go:

```
func minBitFlips(start int, goal int) int {

}
```

Kotlin:

```
class Solution {
fun minBitFlips(start: Int, goal: Int): Int {

}
}
```

Swift:

```
class Solution {
func minBitFlips(_ start: Int, _ goal: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_bit_flips(start: i32, goal: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} start
# @param {Integer} goal
# @return {Integer}
def min_bit_flips(start, goal)

end
```

PHP:

```
class Solution {

/**
* @param Integer $start
* @param Integer $goal
* @return Integer
*/
function minBitFlips($start, $goal) {

}
}
```

Dart:

```
class Solution {
int minBitFlips(int start, int goal) {

}
}
```

Scala:

```
object Solution {
def minBitFlips(start: Int, goal: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_bit_flips(start :: integer, goal :: integer) :: integer
def min_bit_flips(start, goal) do

end
end
```

Erlang:

```
-spec min_bit_flips(Start :: integer(), Goal :: integer()) -> integer().
min_bit_flips(Start, Goal) ->
.
```

Racket:

```
(define/contract (min-bit-flips start goal)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Bit Flips to Convert Number
* 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 minBitFlips(int start, int goal) {
}
};
```

Java Solution:

```
/**
* Problem: Minimum Bit Flips to Convert Number
* 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 minBitFlips(int start, int goal) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Bit Flips to Convert Number
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 minBitFlips(self, start: int, goal: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minBitFlips(self, start, goal):
"""
:type start: int
:type goal: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Bit Flips to Convert Number
* 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} start
* @param {number} goal
* @return {number}
*/
var minBitFlips = function(start, goal) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Bit Flips to Convert Number
* 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 minBitFlips(start: number, goal: number): number {
};
```

C# Solution:

```
/*
* Problem: Minimum Bit Flips to Convert Number
* 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 MinBitFlips(int start, int goal) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Bit Flips to Convert Number
* 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 minBitFlips(int start, int goal) {

}
```

Go Solution:

```
// Problem: Minimum Bit Flips to Convert Number
// 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 minBitFlips(start int, goal int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minBitFlips(start: Int, goal: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minBitFlips(_ start: Int, _ goal: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Bit Flips to Convert Number
// 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_bit_flips(start: i32, goal: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} start
# @param {Integer} goal
# @return {Integer}
def min_bit_flips(start, goal)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $start
* @param Integer $goal
* @return Integer
*/
function minBitFlips($start, $goal) {

}
}
```

Dart Solution:

```
class Solution {
int minBitFlips(int start, int goal) {

}
}
```

Scala Solution:

```
object Solution {
def minBitFlips(start: Int, goal: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_bit_flips(start :: integer, goal :: integer) :: integer
def min_bit_flips(start, goal) do

end
end
```

Erlang Solution:

```
-spec min_bit_flips(Start :: integer(), Goal :: integer()) -> integer().
min_bit_flips(Start, Goal) ->
.
```

Racket Solution:

```
(define/contract (min-bit-flips start goal)
(-> exact-integer? exact-integer? exact-integer?)
)
```