Problem 3377: Digit Operations to Make Two Integers Equal

Problem Information

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

Problem Description

You are given two integers

n

and

m

that consist of the

same

number of digits.

You can perform the following operations

any

number of times:

Choose

any

digit from

n

that is not 9 and

increase

it by 1.

Choose

any

digit from

n

that is not 0 and

decrease

it by 1.

The integer

n

must not be a

prime

number at any point, including its original value and after each operation.

The cost of a transformation is the sum of

all

values that

n

takes throughout the operations performed.

Return the

minimum

cost to transform

n

into

m

. If it is impossible, return -1.

Example 1:

Input:

n = 10, m = 12

Output:

85

Explanation:

We perform the following operations:

Increase the first digit, now

n =

2

0

.

Increase the second digit, now

n = 2

1

.

Increase the second digit, now

n = 2

2

.

Decrease the first digit, now

n =

1

2

.

Example 2:

Input:

n = 4, m = 8

Output:

-1

Explanation:

It is impossible to make

n

equal to

m

.

Example 3:

Input:

n = 6, m = 2

Output:

-1

Explanation:

Since 2 is already a prime, we can't make

n

equal to

m

.

Constraints:

1 <= n, m < 10

n

and

m

consist of the same number of digits.

Code Snippets

C++:

```
class Solution {
public:
int minOperations(int n, int m) {

}
};
```

Java:

```
class Solution {
public int minOperations(int n, int m) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, n: int, m: int) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, n, m):
"""
:type n: int
:type m: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @return {number}
*/
var minOperations = function(n, m) {

};
```

TypeScript:

```
function minOperations(n: number, m: number): number {

};
```

C#:

```
public class Solution {
public int MinOperations(int n, int m) {

}
}
```

C:

```
int minOperations(int n, int m) {

}
```

Go:

```
func minOperations(n int, m int) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(n: Int, m: Int): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ n: Int, _ m: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(n: i32, m: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @return {Integer}
def min_operations(n, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @return Integer
*/
function minOperations($n, $m) {

}
```

```
}
```

Dart:

```
class Solution {
int minOperations(int n, int m) {

}
}
```

Scala:

```
object Solution {
def minOperations(n: Int, m: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(n :: integer, m :: integer) :: integer
def min_operations(n, m) do

end
end
```

Erlang:

```
-spec min_operations(N :: integer(), M :: integer()) -> integer().
min_operations(N, M) ->
.
```

Racket:

```
(define/contract (min-operations n m)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Digit Operations to Make Two Integers Equal
* Difficulty: Medium
* Tags: graph, math, queue, heap
*
* 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 minOperations(int n, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Digit Operations to Make Two Integers Equal
* Difficulty: Medium
* Tags: graph, math, queue, heap
*
* 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 minOperations(int n, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Digit Operations to Make Two Integers Equal
Difficulty: Medium
Tags: graph, math, queue, heap
```

```
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 minOperations(self, n: int, m: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, n, m):
"""
:type n: int
:type m: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Digit Operations to Make Two Integers Equal
* Difficulty: Medium
* Tags: graph, math, queue, heap
*
* 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} m
* @return {number}
*/
var minOperations = function(n, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Digit Operations to Make Two Integers Equal
* Difficulty: Medium
* Tags: graph, math, queue, heap
*
* 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 minOperations(n: number, m: number): number {

};
```

C# Solution:

```
/*
* Problem: Digit Operations to Make Two Integers Equal
* Difficulty: Medium
* Tags: graph, math, queue, heap
*
* 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 MinOperations(int n, int m) {

}
}
```

C Solution:

```
/*
* Problem: Digit Operations to Make Two Integers Equal
* Difficulty: Medium
* Tags: graph, math, queue, heap
*
* 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 minOperations(int n, int m) {

}
```

Go Solution:

```
// Problem: Digit Operations to Make Two Integers Equal
// Difficulty: Medium
// Tags: graph, math, queue, heap
//
// 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 minOperations(n int, m int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(n: Int, m: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ n: Int, _ m: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Digit Operations to Make Two Integers Equal
// Difficulty: Medium
// Tags: graph, math, queue, heap
//
// 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_operations(n: i32, m: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @return {Integer}
def min_operations(n, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @return Integer
*/
function minOperations($n, $m) {

}
}
```

Dart Solution:

```
class Solution {
int minOperations(int n, int m) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def minOperations(n: Int, m: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(n :: integer, m :: integer) :: integer
def min_operations(n, m) do

end
end
```

Erlang Solution:

```
-spec min_operations(N :: integer(), M :: integer()) -> integer().
min_operations(N, M) ->
.
```

Racket Solution:

```
(define/contract (min-operations n m)
(-> exact-integer? exact-integer? exact-integer?)
)
```