Problem 2998: Minimum Number of Operations to Make X and Y Equal

Problem Information

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

Problem Description

You are given two positive integers

x

and

y

.

In one operation, you can do one of the four following operations:

Divide

x

by

11

if

x

is a multiple of

11

.

Divide

x

by

5

if

x

is a multiple of

5

.

Decrement

x

by

1

.

Increment

x

by

1

.

Return

the

minimum

number of operations required to make

x

and

y

equal.

Example 1:

Input:

x = 26, y = 1

Output:

3

Explanation:

We can make 26 equal to 1 by applying the following operations: 1. Decrement x by 1 2. Divide x by 5 3. Divide x by 5 It can be shown that 3 is the minimum number of operations required to make 26 equal to 1.

Example 2:

Input:

x = 54, y = 2

Output:

4

Explanation:

We can make 54 equal to 2 by applying the following operations: 1. Increment x by 1 2. Divide x by 11 3. Divide x by 5 4. Increment x by 1 It can be shown that 4 is the minimum number of operations required to make 54 equal to 2.

Example 3:

Input:

x = 25, y = 30

Output:

5

Explanation:

We can make 25 equal to 30 by applying the following operations: 1. Increment x by 1 2. Increment x by 1 3. Increment x by 1 4. Increment x by 1 5. Increment x by 1 It can be shown that 5 is the minimum number of operations required to make 25 equal to 30.

Constraints:

1 <= x, y <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int minimumOperationsToMakeEqual(int x, int y) {

}
};
```

Java:

```
class Solution {
public int minimumOperationsToMakeEqual(int x, int y) {

}
}
```

Python3:

```
class Solution:
def minimumOperationsToMakeEqual(self, x: int, y: int) -> int:
```

Python:

```
class Solution(object):
def minimumOperationsToMakeEqual(self, x, y):
"""
:type x: int
:type y: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} y
* @return {number}
*/
var minimumOperationsToMakeEqual = function(x, y) {

};
```

TypeScript:

```
function minimumOperationsToMakeEqual(x: number, y: number): number {

};
```

C#:

```
public class Solution {
public int MinimumOperationsToMakeEqual(int x, int y) {

}
}
```

C:

```
int minimumOperationsToMakeEqual(int x, int y) {

}
```

Go:

```
func minimumOperationsToMakeEqual(x int, y int) int {

}
```

Kotlin:

```
class Solution {
fun minimumOperationsToMakeEqual(x: Int, y: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumOperationsToMakeEqual(_ x: Int, _ y: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_operations_to_make_equal(x: i32, y: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} x
# @param {Integer} y
# @return {Integer}
def minimum_operations_to_make_equal(x, y)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @return Integer
*/
function minimumOperationsToMakeEqual($x, $y) {

}
}
```

Dart:

```
class Solution {
int minimumOperationsToMakeEqual(int x, int y) {

}
}
```

Scala:

```
object Solution {
def minimumOperationsToMakeEqual(x: Int, y: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec minimum_operations_to_make_equal(x :: integer, y :: integer) :: integer
def minimum_operations_to_make_equal(x, y) do

end
end
```

Erlang:

```
-spec minimum_operations_to_make_equal(X :: integer(), Y :: integer()) ->
integer().
minimum_operations_to_make_equal(X, Y) ->
.
```

Racket:

```
(define/contract (minimum-operations-to-make-equal x y)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Make X and Y Equal
* Difficulty: Medium
* Tags: dp, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
```

```
int minimumOperationsToMakeEqual(int x, int y) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Make X and Y Equal
* Difficulty: Medium
* Tags: dp, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumOperationsToMakeEqual(int x, int y) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Make X and Y Equal
Difficulty: Medium
Tags: dp, search

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumOperationsToMakeEqual(self, x: int, y: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumOperationsToMakeEqual(self, x, y):
"""
:type x: int
:type y: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make X and Y Equal
* Difficulty: Medium
* Tags: dp, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} x
* @param {number} y
* @return {number}
*/
var minimumOperationsToMakeEqual = function(x, y) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make X and Y Equal
* Difficulty: Medium
* Tags: dp, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumOperationsToMakeEqual(x: number, y: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Make X and Y Equal
* Difficulty: Medium
* Tags: dp, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimumOperationsToMakeEqual(int x, int y) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Make X and Y Equal
* Difficulty: Medium
* Tags: dp, search
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumOperationsToMakeEqual(int x, int y) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Make X and Y Equal
// Difficulty: Medium
```

```
// Tags: dp, search
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func minimumOperationsToMakeEqual(x int, y int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumOperationsToMakeEqual(x: Int, y: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumOperationsToMakeEqual(_ x: Int, _ y: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Make X and Y Equal
// Difficulty: Medium
// Tags: dp, search
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_operations_to_make_equal(x: i32, y: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @param {Integer} y
# @return {Integer}
def minimum_operations_to_make_equal(x, y)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @param Integer $y
* @return Integer
*/
function minimumOperationsToMakeEqual($x, $y) {

}
}
```

Dart Solution:

```
class Solution {
int minimumOperationsToMakeEqual(int x, int y) {

}
}
```

Scala Solution:

```
object Solution {
def minimumOperationsToMakeEqual(x: Int, y: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_operations_to_make_equal(x :: integer, y :: integer) :: integer
def minimum_operations_to_make_equal(x, y) do

end
end
```

Erlang Solution:

```
-spec minimum_operations_to_make_equal(X :: integer(), Y :: integer()) ->
integer().
minimum_operations_to_make_equal(X, Y) ->
.
```

Racket Solution:

```
(define/contract (minimum-operations-to-make-equal x y)
(-> exact-integer? exact-integer? exact-integer?)
)
```