Problem 858: Mirror Reflection

Problem Information

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

Problem Description

There is a special square room with mirrors on each of the four walls. Except for the southwest corner, there are receptors on each of the remaining corners, numbered

0

,

1

, and

2

.

The square room has walls of length

p

and a laser ray from the southwest corner first meets the east wall at a distance

q

from the

0

th

receptor.

Given the two integers

p

and

q

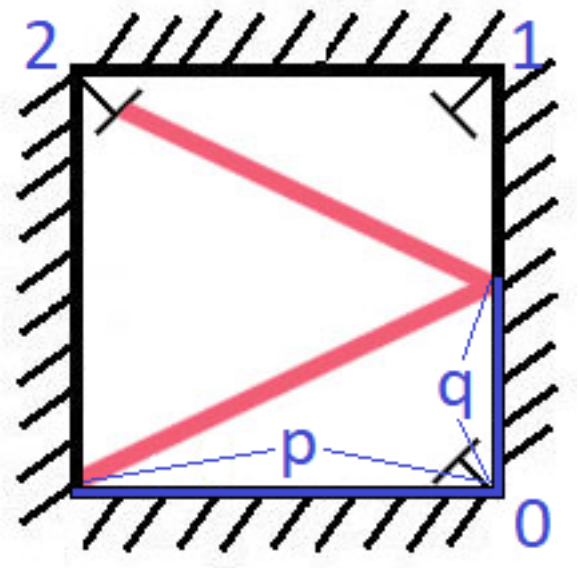
, return

the number of the receptor that the ray meets first

.

The test cases are guaranteed so that the ray will meet a receptor eventually.

Example 1:



Input:

p = 2, q = 1

Output:

2

Explanation:

The ray meets receptor 2 the first time it gets reflected back to the left wall.

Example 2:

Input:

p = 3, q = 1

Output:

1

Constraints:

1 <= q <= p <= 1000

Code Snippets

C++:

```
class Solution {
public:
int mirrorReflection(int p, int q) {

}
};
```

Java:

```
class Solution {
public int mirrorReflection(int p, int q) {

}
```

```
}
```

Python3:

```
class Solution:
def mirrorReflection(self, p: int, q: int) -> int:
```

Python:

```
class Solution(object):
def mirrorReflection(self, p, q):
"""
:type p: int
:type q: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} p
* @param {number} q
* @return {number}
*/
var mirrorReflection = function(p, q) {

};
```

TypeScript:

```
function mirrorReflection(p: number, q: number): number {

};
```

C#:

```
public class Solution {
public int MirrorReflection(int p, int q) {

}
}
```

C:

```
int mirrorReflection(int p, int q) {

}
```

Go:

```
func mirrorReflection(p int, q int) int {

}
```

Kotlin:

```
class Solution {
fun mirrorReflection(p: Int, q: Int): Int {

}
}
```

Swift:

```
class Solution {
func mirrorReflection(_ p: Int, _ q: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn mirror_reflection(p: i32, q: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} p
# @param {Integer} q
# @return {Integer}
def mirror_reflection(p, q)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $p
* @param Integer $q
* @return Integer
*/
function mirrorReflection($p, $q) {

}
}
```

Dart:

```
class Solution {
int mirrorReflection(int p, int q) {

}
}
```

Scala:

```
object Solution {
def mirrorReflection(p: Int, q: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec mirror_reflection(p :: integer, q :: integer) :: integer
def mirror_reflection(p, q) do

end
end
```

Erlang:

```
-spec mirror_reflection(P :: integer(), Q :: integer()) -> integer().
mirror_reflection(P, Q) ->
.
```

Racket:

```
(define/contract (mirror-reflection p q)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Mirror Reflection
* Difficulty: Medium
* Tags: math
*
* 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 mirrorReflection(int p, int q) {

}
};
```

Java Solution:

```
/**
* Problem: Mirror Reflection
* Difficulty: Medium
* Tags: math
*
* 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 mirrorReflection(int p, int q) {

}
}
```

Python3 Solution:

```
"""
Problem: Mirror Reflection
Difficulty: Medium
Tags: math

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 mirrorReflection(self, p: int, q: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def mirrorReflection(self, p, q):
"""
:type p: int
:type q: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Mirror Reflection
* Difficulty: Medium
* Tags: math
```

```
*
* 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} p
* @param {number} q
* @return {number}
*/
var mirrorReflection = function(p, q) {

};
```

TypeScript Solution:

```
/**
* Problem: Mirror Reflection
* Difficulty: Medium
* Tags: math
*
* 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 mirrorReflection(p: number, q: number): number {

};
```

C# Solution:

```
/*
* Problem: Mirror Reflection
* Difficulty: Medium
* Tags: math
*
* 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 MirrorReflection(int p, int q) {

}
}
```

C Solution:

```
/*
* Problem: Mirror Reflection
* Difficulty: Medium
* Tags: math
*
* 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 mirrorReflection(int p, int q) {

}
```

Go Solution:

```
// Problem: Mirror Reflection
// Difficulty: Medium
// Tags: math
//
// 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 mirrorReflection(p int, q int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mirrorReflection(p: Int, q: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func mirrorReflection(_ p: Int, _ q: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Mirror Reflection
// Difficulty: Medium
// Tags: math
//
// 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 mirror_reflection(p: i32, q: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} p
# @param {Integer} q
# @return {Integer}
def mirror_reflection(p, q)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $p
* @param Integer $q
* @return Integer
*/
function mirrorReflection($p, $q) {

}
}
```

Dart Solution:

```
class Solution {
int mirrorReflection(int p, int q) {

}
}
```

Scala Solution:

```
object Solution {
def mirrorReflection(p: Int, q: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec mirror_reflection(p :: integer, q :: integer) :: integer
def mirror_reflection(p, q) do

end
end
```

Erlang Solution:

```
-spec mirror_reflection(P :: integer(), Q :: integer()) -> integer().
mirror_reflection(P, Q) ->
.
```

Racket Solution:

```
(define/contract (mirror-reflection p q)
(-> exact-integer? exact-integer? exact-integer?)
)
```