Problem 3609: Minimum Moves to Reach Target in Grid

Problem Information

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

Problem Description

You are given four integers

sx

,

sy

,

tx

, and

ty

, representing two points

(sx, sy)

and

(tx, ty)

on an infinitely large 2D grid.

You start at

(sx, sy)

.

At any point

(x, y)

, define

m = max(x, y)

. You can either:

Move to

(x + m, y)

, or

Move to

(x, y + m)

.

Return the

minimum

number of moves required to reach

(tx, ty)

. If it is impossible to reach the target, return -1.

Example 1:

Input:

sx = 1, sy = 2, tx = 5, ty = 4

Output:

2

Explanation:

The optimal path is:

Move 1:

max(1, 2) = 2

. Increase the y-coordinate by 2, moving from

(1, 2)

to

(1, 2 + 2) = (1, 4)

.

Move 2:

max(1, 4) = 4

. Increase the x-coordinate by 4, moving from

(1, 4)

to

(1 + 4, 4) = (5, 4)

.

Thus, the minimum number of moves to reach

(5, 4)

is 2.

Example 2:

Input:

sx = 0, sy = 1, tx = 2, ty = 3

Output:

3

Explanation:

The optimal path is:

Move 1:

max(0, 1) = 1

. Increase the x-coordinate by 1, moving from

(0, 1)

to

(0 + 1, 1) = (1, 1)

.

Move 2:

$\max(1, 1) = 1$

. Increase the x-coordinate by 1, moving from

$(1, 1)$

to

$(1 + 1, 1) = (2, 1)$

.

Move 3:

$\max(2, 1) = 2$

. Increase the y-coordinate by 2, moving from

$(2, 1)$

to

$(2, 1 + 2) = (2, 3)$

.

Thus, the minimum number of moves to reach

$(2, 3)$

is 3.

Example 3:

Input:

sx = 1, sy = 1, tx = 2, ty = 2

Output:

-1

Explanation:

It is impossible to reach

(2, 2)

from

(1, 1)

using the allowed moves. Thus, the answer is -1.

Constraints:

0 <= sx <= tx <= 10

9

0 <= sy <= ty <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minMoves(int sx, int sy, int tx, int ty) {

}
};
```

Java:

```
class Solution {
public int minMoves(int sx, int sy, int tx, int ty) {
```

```
}
}
```

Python3:

```
class Solution:
def minMoves(self, sx: int, sy: int, tx: int, ty: int) -> int:
```

Python:

```
class Solution(object):
def minMoves(self, sx, sy, tx, ty):
"""
:type sx: int
:type sy: int
:type tx: int
:type ty: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} sx
* @param {number} sy
* @param {number} tx
* @param {number} ty
* @return {number}
*/
var minMoves = function(sx, sy, tx, ty) {

};
```

TypeScript:

```
function minMoves(sx: number, sy: number, tx: number, ty: number): number {

};
```

C#:

```
public class Solution {
public int MinMoves(int sx, int sy, int tx, int ty) {

}
}
```

C:

```
int minMoves(int sx, int sy, int tx, int ty) {

}
```

Go:

```
func minMoves(sx int, sy int, tx int, ty int) int {

}
```

Kotlin:

```
class Solution {
fun minMoves(sx: Int, sy: Int, tx: Int, ty: Int): Int {

}
}
```

Swift:

```
class Solution {
func minMoves(_ sx: Int, _ sy: Int, _ tx: Int, _ ty: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_moves(sx: i32, sy: i32, tx: i32, ty: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} sx
# @param {Integer} sy
# @param {Integer} tx
# @param {Integer} ty
# @return {Integer}
def min_moves(sx, sy, tx, ty)

end
```

PHP:

```
class Solution {

/**
* @param Integer $sx
* @param Integer $sy
* @param Integer $tx
* @param Integer $ty
* @return Integer
*/
function minMoves($sx, $sy, $tx, $ty) {

}
}
```

Dart:

```
class Solution {
int minMoves(int sx, int sy, int tx, int ty) {

}
}
```

Scala:

```
object Solution {
def minMoves(sx: Int, sy: Int, tx: Int, ty: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_moves(sx :: integer, sy :: integer, tx :: integer, ty :: integer)
:: integer
def min_moves(sx, sy, tx, ty) do

end
end
```

Erlang:

```
-spec min_moves(Sx :: integer(), Sy :: integer(), Tx :: integer(), Ty ::
integer()) -> integer().
min_moves(Sx, Sy, Tx, Ty) ->
.
```

Racket:

```
(define/contract (min-moves sx sy tx ty)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Reach Target in Grid
* Difficulty: Hard
* 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 minMoves(int sx, int sy, int tx, int ty) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Reach Target in Grid
* Difficulty: Hard
* 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 minMoves(int sx, int sy, int tx, int ty) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Reach Target in Grid
Difficulty: Hard
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 minMoves(self, sx: int, sy: int, tx: int, ty: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minMoves(self, sx, sy, tx, ty):
```

```
"""
:type sx: int
:type sy: int
:type tx: int
:type ty: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Reach Target in Grid
* Difficulty: Hard
* 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} sx
* @param {number} sy
* @param {number} tx
* @param {number} ty
* @return {number}
*/
var minMoves = function(sx, sy, tx, ty) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Reach Target in Grid
* Difficulty: Hard
* 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 minMoves(sx: number, sy: number, tx: number, ty: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Reach Target in Grid
* Difficulty: Hard
* 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 MinMoves(int sx, int sy, int tx, int ty) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Reach Target in Grid
* Difficulty: Hard
* 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 minMoves(int sx, int sy, int tx, int ty) {

}
```

Go Solution:

```
// Problem: Minimum Moves to Reach Target in Grid
// Difficulty: Hard
// 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 minMoves(sx int, sy int, tx int, ty int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMoves(sx: Int, sy: Int, tx: Int, ty: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minMoves(_ sx: Int, _ sy: Int, _ tx: Int, _ ty: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Moves to Reach Target in Grid
// Difficulty: Hard
// 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 min_moves(sx: i32, sy: i32, tx: i32, ty: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} sx
# @param {Integer} sy
# @param {Integer} tx
# @param {Integer} ty
# @return {Integer}
def min_moves(sx, sy, tx, ty)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $sx
* @param Integer $sy
* @param Integer $tx
* @param Integer $ty
* @return Integer
*/
function minMoves($sx, $sy, $tx, $ty) {

}
}
```

Dart Solution:

```
class Solution {
int minMoves(int sx, int sy, int tx, int ty) {

}
}
```

Scala Solution:

```
object Solution {
def minMoves(sx: Int, sy: Int, tx: Int, ty: Int): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_moves(sx :: integer, sy :: integer, tx :: integer, ty :: integer)
:: integer
def min_moves(sx, sy, tx, ty) do

end
end
```

Erlang Solution:

```
-spec min_moves(Sx :: integer(), Sy :: integer(), Tx :: integer(), Ty ::
integer()) -> integer().
min_moves(Sx, Sy, Tx, Ty) ->
.
```

Racket Solution:

```
(define/contract (min-moves sx sy tx ty)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```