Problem 780: Reaching Points

Problem Information

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

Problem Description

Given four integers

sx

,

sy

,

tx

, and

ty

, return

true

if it is possible to convert the point

(sx, sy)

to the point

(tx, ty)

through some operations

, or

false

otherwise

.

The allowed operation on some point

(x, y)

is to convert it to either

(x, x + y)

or

(x + y, y)

.

Example 1:

Input:

sx = 1, sy = 1, tx = 3, ty = 5

Output:

true

Explanation:

One series of moves that transforms the starting point to the target is: (1, 1) -> (1, 2) (1, 2) -> (3, 2) (3, 2) -> (3, 5)

Example 2:

Input:

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

Output:

false

Example 3:

Input:

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

Output:

true

Constraints:

1 <= sx, sy, tx, ty <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```java
class Solution {
public boolean reachingPoints(int sx, int sy, int tx, int ty) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```typescript
function reachingPoints(sx: number, sy: number, tx: number, ty: number):
boolean {

};
```

C#:

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

}
}
```

C:

```
bool reachingPoints(int sx, int sy, int tx, int ty) {

}
```

Go:

```
func reachingPoints(sx int, sy int, tx int, ty int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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


}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reaching-points sx sy tx ty)
(-> exact-integer? exact-integer? exact-integer? exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reaching Points
* 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:
bool reachingPoints(int sx, int sy, int tx, int ty) {

}
};
```

Java Solution:

```
/**
* Problem: Reaching Points
* 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 boolean reachingPoints(int sx, int sy, int tx, int ty) {

}
}
```

Python3 Solution:

```
"""
Problem: Reaching Points
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 reachingPoints(self, sx: int, sy: int, tx: int, ty: int) -> bool:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reaching Points
* 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 {boolean}
*/
var reachingPoints = function(sx, sy, tx, ty) {

};
```

TypeScript Solution:

```
/**
* Problem: Reaching Points
* 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 reachingPoints(sx: number, sy: number, tx: number, ty: number):
boolean {

};
```

C# Solution:

```
/*
* Problem: Reaching Points
* 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 bool ReachingPoints(int sx, int sy, int tx, int ty) {

}
}
```

C Solution:

```
/*
* Problem: Reaching Points
* 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
*/

bool reachingPoints(int sx, int sy, int tx, int ty) {

}
```

Go Solution:

```
// Problem: Reaching Points
// 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 reachingPoints(sx int, sy int, tx int, ty int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reaching Points
// 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 reaching_points(sx: i32, sy: i32, tx: i32, ty: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reaching-points sx sy tx ty)
(-> exact-integer? exact-integer? exact-integer? exact-integer? boolean?)
)
```