# Problem 2849: Determine if a Cell Is Reachable at a Given Time

## Problem Information

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

## Problem Description

You are given four integers

sx

,

sy

,

fx

,

fy

, and a

non-negative

integer

t

.

In an infinite 2D grid, you start at the cell

(sx, sy)

. Each second, you

must

move to any of its adjacent cells.

Return

true

if you can reach cell

(fx, fy)

after

exactly

t

seconds

,

or

false

otherwise

.

A cell's

adjacent cells

are the 8 cells around it that share at least one corner with it. You can visit the same cell several times.

Example 1:

Input:

sx = 2, sy = 4, fx = 7, fy = 7, t = 6

Output:

true

Explanation:

Starting at cell (2, 4), we can reach cell (7, 7) in exactly 6 seconds by going through the cells depicted in the picture above.

Example 2:

Input:

sx = 3, sy = 1, fx = 7, fy = 3, t = 3

Output:

false

Explanation:

Starting at cell (3, 1), it takes at least 4 seconds to reach cell (7, 3) by going through the cells depicted in the picture above. Hence, we cannot reach cell (7, 3) at the third second.

Constraints:

1 <= sx, sy, fx, fy <= 10

9

0 <= t <= 10

9

## Code Snippets

**C++:**

```
class Solution {
public:
bool isReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
};
```

**Java:**

```
class Solution {
public boolean isReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
}
```

**Python3:**

```
class Solution:
def isReachableAtTime(self, sx: int, sy: int, fx: int, fy: int, t: int) ->
bool:
```

**Python:**

```
class Solution(object):
def isReachableAtTime(self, sx, sy, fx, fy, t):
"""
:type sx: int
:type sy: int
:type fx: int
:type fy: int
:type t: int
```

```
:rtype: bool
"""
```

**JavaScript:**

```
/**
* @param {number} sx
* @param {number} sy
* @param {number} fx
* @param {number} fy
* @param {number} t
* @return {boolean}
*/
var isReachableAtTime = function(sx, sy, fx, fy, t) {

};
```

**TypeScript:**

```
function isReachableAtTime(sx: number, sy: number, fx: number, fy: number, t:
number): boolean {

};
```

**C#:**

```
public class Solution {
public bool IsReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
}
```

**C:**

```
bool isReachableAtTime(int sx, int sy, int fx, int fy, int t){

}
```

**Go:**

```
func isReachableAtTime(sx int, sy int, fx int, fy int, t int) bool {
```

```
}
```

**Kotlin:**

```
class Solution {
fun isReachableAtTime(sx: Int, sy: Int, fx: Int, fy: Int, t: Int): Boolean {

}
}
```

**Swift:**

```
class Solution {
func isReachableAtTime(_ sx: Int, _ sy: Int, _ fx: Int, _ fy: Int, _ t: Int)
-> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn is_reachable_at_time(sx: i32, sy: i32, fx: i32, fy: i32, t: i32) ->
bool {

}
}
```

**Ruby:**

```
# @param {Integer} sx
# @param {Integer} sy
# @param {Integer} fx
# @param {Integer} fy
# @param {Integer} t
# @return {Boolean}
def is_reachable_at_time(sx, sy, fx, fy, t)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $sx
* @param Integer $sy
* @param Integer $fx
* @param Integer $fy
* @param Integer $t
* @return Boolean
*/
function isReachableAtTime($sx, $sy, $fx, $fy, $t) {

}
}
```

**Dart:**

```
class Solution {
bool isReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
}
```

**Scala:**

```
object Solution {
def isReachableAtTime(sx: Int, sy: Int, fx: Int, fy: Int, t: Int): Boolean =
{

}
}
```

**Elixir:**

```
defmodule Solution do
@spec is_reachable_at_time(sx :: integer, sy :: integer, fx :: integer, fy ::
integer, t :: integer) :: boolean
def is_reachable_at_time(sx, sy, fx, fy, t) do

end
end
```

**Erlang:**

```
-spec is_reachable_at_time(Sx :: integer(), Sy :: integer(), Fx :: integer(),
Fy :: integer(), T :: integer()) -> boolean().
is_reachable_at_time(Sx, Sy, Fx, Fy, T) ->
.
```

**Racket:**

```
(define/contract (is-reachable-at-time sx sy fx fy t)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? boolean?)

)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Determine if a Cell Is Reachable at a Given Time
* 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:
bool isReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
};
```

### Java Solution:

```
/**
* Problem: Determine if a Cell Is Reachable at a Given Time
* 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 boolean isReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
}
```

### Python3 Solution:

```
"""
Problem: Determine if a Cell Is Reachable at a Given Time
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 isReachableAtTime(self, sx: int, sy: int, fx: int, fy: int, t: int) ->
bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def isReachableAtTime(self, sx, sy, fx, fy, t):
"""
:type sx: int
:type sy: int
:type fx: int
:type fy: int
:type t: int
:rtype: bool
"""
```

### JavaScript Solution:

```
/**
* Problem: Determine if a Cell Is Reachable at a Given Time
* 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} sx
* @param {number} sy
* @param {number} fx
* @param {number} fy
* @param {number} t
* @return {boolean}
*/
var isReachableAtTime = function(sx, sy, fx, fy, t) {

};
```

### TypeScript Solution:

```
/**
* Problem: Determine if a Cell Is Reachable at a Given Time
* 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 isReachableAtTime(sx: number, sy: number, fx: number, fy: number, t:
number): boolean {

};
```

### C# Solution:

```
/*
* Problem: Determine if a Cell Is Reachable at a Given Time
* 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 bool IsReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
}
```

### C Solution:

```
/*
* Problem: Determine if a Cell Is Reachable at a Given Time
* 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
*/

bool isReachableAtTime(int sx, int sy, int fx, int fy, int t){

}
```

### Go Solution:

```
// Problem: Determine if a Cell Is Reachable at a Given Time
// 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 isReachableAtTime(sx int, sy int, fx int, fy int, t int) bool {

}
```

**Kotlin Solution:**

```
class Solution {
fun isReachableAtTime(sx: Int, sy: Int, fx: Int, fy: Int, t: Int): Boolean {

}
}
```

**Swift Solution:**

```
class Solution {
func isReachableAtTime(_ sx: Int, _ sy: Int, _ fx: Int, _ fy: Int, _ t: Int)
-> Bool {

}
}
```

**Rust Solution:**

```
// Problem: Determine if a Cell Is Reachable at a Given Time
// 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 is_reachable_at_time(sx: i32, sy: i32, fx: i32, fy: i32, t: i32) ->
bool {

}
}
```

**Ruby Solution:**

```
# @param {Integer} sx
# @param {Integer} sy
# @param {Integer} fx
# @param {Integer} fy
# @param {Integer} t
# @return {Boolean}
def is_reachable_at_time(sx, sy, fx, fy, t)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer $sx
* @param Integer $sy
* @param Integer $fx
* @param Integer $fy
* @param Integer $t
* @return Boolean
*/
function isReachableAtTime($sx, $sy, $fx, $fy, $t) {

}
}
```

**Dart Solution:**

```
class Solution {
bool isReachableAtTime(int sx, int sy, int fx, int fy, int t) {

}
}
```

**Scala Solution:**

```
object Solution {
def isReachableAtTime(sx: Int, sy: Int, fx: Int, fy: Int, t: Int): Boolean =
{

}
```

```
}
```

### Elixir Solution:

```
defmodule Solution do
@spec is_reachable_at_time(sx :: integer, sy :: integer, fx :: integer, fy ::
integer, t :: integer) :: boolean
def is_reachable_at_time(sx, sy, fx, fy, t) do

end
end
```

### Erlang Solution:

```
-spec is_reachable_at_time(Sx :: integer(), Sy :: integer(), Fx :: integer(),
Fy :: integer(), T :: integer()) -> boolean().
is_reachable_at_time(Sx, Sy, Fx, Fy, T) ->
.
```

### Racket Solution:

```
(define/contract (is-reachable-at-time sx sy fx fy t)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? boolean?)

)
```