Problem 789: Escape The Ghosts

Problem Information

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

Problem Description

You are playing a simplified PAC-MAN game on an infinite 2-D grid. You start at the point

[0, 0]

, and you are given a destination point

target = [x

target

, y

target

]

that you are trying to get to. There are several ghosts on the map with their starting positions given as a 2D array

ghosts

, where

ghosts[i] = [x

i

, y

i

]

represents the starting position of the

i

th

ghost. All inputs are

integral coordinates

.

Each turn, you and all the ghosts may independently choose to either

move 1 unit

in any of the four cardinal directions: north, east, south, or west, or

stay still

. All actions happen

simultaneously

.

You escape if and only if you can reach the target

before

any ghost reaches you. If you reach any square (including the target) at the

same time

as a ghost, it

does not

count as an escape.

Return

true

if it is possible to escape regardless of how the ghosts move, otherwise return

false

.

Example 1:

Input:

ghosts = [[1,0],[0,3]], target = [0,1]

Output:

true

Explanation:

You can reach the destination (0, 1) after 1 turn, while the ghosts located at (1, 0) and (0, 3) cannot catch up with you.

Example 2:

Input:

ghosts = [[1,0]], target = [2,0]

Output:

false

Explanation:

You need to reach the destination (2, 0), but the ghost at (1, 0) lies between you and the destination.

Example 3:

Input:

ghosts = [[2,0]], target = [1,0]

Output:

false

Explanation:

The ghost can reach the target at the same time as you.

Constraints:

1 <= ghosts.length <= 100

ghosts[i].length == 2

-10

4

<= x

i

, y

i

<= 10

4

There can be

multiple ghosts

in the same location.

target.length == 2

-10

4

<= x

target

, y

target

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
bool escapeGhosts(vector<vector<int>>& ghosts, vector<int>& target) {
```

```
}
};
```

Java:

```
class Solution {
public boolean escapeGhosts(int[][] ghosts, int[] target) {

}
}
```

Python3:

```
class Solution:
def escapeGhosts(self, ghosts: List[List[int]], target: List[int]) -> bool:
```

Python:

```
class Solution(object):
def escapeGhosts(self, ghosts, target):
"""
:type ghosts: List[List[int]]
:type target: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} ghosts
* @param {number[]} target
* @return {boolean}
*/
var escapeGhosts = function(ghosts, target) {

};
```

TypeScript:

```
function escapeGhosts(ghosts: number[][], target: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool EscapeGhosts(int[][] ghosts, int[] target) {

}
}
```

C:

```
bool escapeGhosts(int** ghosts, int ghostsSize, int* ghostsColSize, int*
target, int targetSize) {

}
```

Go:

```
func escapeGhosts(ghosts [][]int, target []int) bool {

}
```

Kotlin:

```
class Solution {
fun escapeGhosts(ghosts: Array<IntArray>, target: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func escapeGhosts(_ ghosts: [[Int]], _ target: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn escape_ghosts(ghosts: Vec<Vec<i32>>, target: Vec<i32>) -> bool {

}
```

```
}
```

Ruby:

```
# @param {Integer[][]} ghosts
# @param {Integer[]} target
# @return {Boolean}
def escape_ghosts(ghosts, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $ghosts
* @param Integer[] $target
* @return Boolean
*/
function escapeGhosts($ghosts, $target) {

}
}
```

Dart:

```
class Solution {
bool escapeGhosts(List<List<int>> ghosts, List<int> target) {

}
}
```

Scala:

```
object Solution {
def escapeGhosts(ghosts: Array[Array[Int]], target: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec escape_ghosts(ghosts :: [[integer]], target :: [integer]) :: boolean
def escape_ghosts(ghosts, target) do

end
end
```

Erlang:

```
-spec escape_ghosts(Ghosts :: [[integer()]], Target :: [integer()]) ->
boolean().
escape_ghosts(Ghosts, Target) ->
.
```

Racket:

```
(define/contract (escape-ghosts ghosts target)
(-> (listof (listof exact-integer?)) (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Escape The Ghosts
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool escapeGhosts(vector<vector<int>>& ghosts, vector<int>& target) {

}
};
```

Java Solution:

```
/**
* Problem: Escape The Ghosts
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean escapeGhosts(int[][] ghosts, int[] target) {

}
}
```

Python3 Solution:

```
"""
Problem: Escape The Ghosts
Difficulty: Medium
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def escapeGhosts(self, ghosts: List[List[int]], target: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def escapeGhosts(self, ghosts, target):
"""
:type ghosts: List[List[int]]
:type target: List[int]
:rtype: bool
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Escape The Ghosts
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} ghosts
* @param {number[]} target
* @return {boolean}
*/
var escapeGhosts = function(ghosts, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Escape The Ghosts
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function escapeGhosts(ghosts: number[][], target: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Escape The Ghosts
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool EscapeGhosts(int[][] ghosts, int[] target) {

}
}
```

C Solution:

```
/*
* Problem: Escape The Ghosts
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool escapeGhosts(int** ghosts, int ghostsSize, int* ghostsColSize, int*
target, int targetSize) {

}
```

Go Solution:

```
// Problem: Escape The Ghosts
// Difficulty: Medium
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
func escapeGhosts(ghosts [][]int, target []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun escapeGhosts(ghosts: Array<IntArray>, target: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func escapeGhosts(_ ghosts: [[Int]], _ target: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Escape The Ghosts
// Difficulty: Medium
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn escape_ghosts(ghosts: Vec<Vec<i32>>, target: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} ghosts
# @param {Integer[]} target
```

```
# @return {Boolean}
def escape_ghosts(ghosts, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $ghosts
* @param Integer[] $target
* @return Boolean
*/
function escapeGhosts($ghosts, $target) {

}
}
```

Dart Solution:

```
class Solution {
bool escapeGhosts(List<List<int>> ghosts, List<int> target) {

}
}
```

Scala Solution:

```
object Solution {
def escapeGhosts(ghosts: Array[Array[Int]], target: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec escape_ghosts(ghosts :: [[integer]], target :: [integer]) :: boolean
def escape_ghosts(ghosts, target) do
```

```
end
end
```

Erlang Solution:

```
-spec escape_ghosts(Ghosts :: [[integer()]], Target :: [integer()]) ->
boolean().
escape_ghosts(Ghosts, Target) ->
.
```

Racket Solution:

```
(define/contract (escape-ghosts ghosts target)
(-> (listof (listof exact-integer?)) (listof exact-integer?) boolean?)
)
```