Problem 3341: Find Minimum Time to Reach Last Room I

Problem Information

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

Problem Description

There is a dungeon with

n x m

rooms arranged as a grid.

You are given a 2D array

moveTime

of size

n x m

, where

moveTime[i][j]

represents the

minimum

time in seconds

after

which the room opens and can be moved to. You start from the room

(0, 0)

at time

t = 0

and can move to an

adjacent

room. Moving between adjacent rooms takes

exactly

one second.

Return the

minimum

time to reach the room

(n - 1, m - 1)

.

Two rooms are

adjacent

if they share a common wall, either

horizontally

or

vertically

.

Example 1:

Input:

moveTime = [[0,4],[4,4]]

Output:

6

Explanation:

The minimum time required is 6 seconds.

At time

t == 4

, move from room

(0, 0)

to room

(1, 0)

in one second.

At time

t == 5

, move from room

(1, 0)

to room

(1, 1)

in one second.

Example 2:

Input:

moveTime = [[0,0,0],[0,0,0]]

Output:

3

Explanation:

The minimum time required is 3 seconds.

At time

t == 0

, move from room

(0, 0)

to room

(1, 0)

in one second.

At time

t == 1

, move from room

(1, 0)

to room

(1, 1)

in one second.

At time

t == 2

, move from room

(1, 1)

to room

(1, 2)

in one second.

Example 3:

Input:

moveTime = [[0,1],[1,2]]

Output:

3

Constraints:

2 <= n == moveTime.length <= 50

2 <= m == moveTime[i].length <= 50

0 <= moveTime[i][j] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minTimeToReach(vector<vector<int>>& moveTime) {

}
};
```

Java:

```
class Solution {
public int minTimeToReach(int[][] moveTime) {

}
}
```

Python3:

```
class Solution:
def minTimeToReach(self, moveTime: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minTimeToReach(self, moveTime):
"""
:type moveTime: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} moveTime
```

```
 * @return {number}
 */
var minTimeToReach = function(moveTime) {

};
```

TypeScript:

```
function minTimeToReach(moveTime: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinTimeToReach(int[][] moveTime) {

}
}
```

C:

```
int minTimeToReach(int** moveTime, int moveTimeSize, int* moveTimeColSize) {

}
```

Go:

```
func minTimeToReach(moveTime [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minTimeToReach(moveTime: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minTimeToReach(_ moveTime: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_time_to_reach(move_time: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} move_time
# @return {Integer}
def min_time_to_reach(move_time)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $moveTime
* @return Integer
*/
function minTimeToReach($moveTime) {

}
}
```

Dart:

```
class Solution {
int minTimeToReach(List<List<int>> moveTime) {

}
}
```

Scala:

```
object Solution {
def minTimeToReach(moveTime: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_time_to_reach(move_time :: [[integer]]) :: integer
def min_time_to_reach(move_time) do

end
end
```

Erlang:

```
-spec min_time_to_reach(MoveTime :: [[integer()]]) -> integer().
min_time_to_reach(MoveTime) ->
.
```

Racket:

```
(define/contract (min-time-to-reach moveTime)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Minimum Time to Reach Last Room I
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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:
int minTimeToReach(vector<vector<int>>& moveTime) {

}
};
```

Java Solution:

```
/**
* Problem: Find Minimum Time to Reach Last Room I
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 int minTimeToReach(int[][] moveTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Minimum Time to Reach Last Room I
Difficulty: Medium
Tags: array, graph, queue, heap

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 minTimeToReach(self, moveTime: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def minTimeToReach(self, moveTime):
"""
:type moveTime: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find Minimum Time to Reach Last Room I
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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[][]} moveTime
* @return {number}
*/
var minTimeToReach = function(moveTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Minimum Time to Reach Last Room I
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 minTimeToReach(moveTime: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Find Minimum Time to Reach Last Room I
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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 int MinTimeToReach(int[][] moveTime) {

}
}
```

C Solution:

```
/*
* Problem: Find Minimum Time to Reach Last Room I
* Difficulty: Medium
* Tags: array, graph, queue, heap
*
* 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
*/

int minTimeToReach(int** moveTime, int moveTimeSize, int* moveTimeColSize) {

}
```

Go Solution:

```
// Problem: Find Minimum Time to Reach Last Room I
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// 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 minTimeToReach(moveTime [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minTimeToReach(moveTime: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minTimeToReach(_ moveTime: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Minimum Time to Reach Last Room I
// Difficulty: Medium
// Tags: array, graph, queue, heap
//
// 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 min_time_to_reach(move_time: Vec<Vec<i32>>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} move_time
# @return {Integer}
def min_time_to_reach(move_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $moveTime
* @return Integer
*/
function minTimeToReach($moveTime) {

}
}
```

Dart Solution:

```
class Solution {
int minTimeToReach(List<List<int>> moveTime) {

}
}
```

Scala Solution:

```
object Solution {
def minTimeToReach(moveTime: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_time_to_reach(move_time :: [[integer]]) :: integer
def min_time_to_reach(move_time) do

end
end
```

Erlang Solution:

```
-spec min_time_to_reach(MoveTime :: [[integer()]]) -> integer().
min_time_to_reach(MoveTime) ->
.
```

Racket Solution:

```
(define/contract (min-time-to-reach moveTime)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```