Problem 2617: Minimum Number of Visited Cells in a Grid

Problem Information

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

Problem Description

You are given a

0-indexed

m x n

integer matrix

grid

. Your initial position is at the

top-left

cell

(0, 0)

.

Starting from the cell

(i, j)

, you can move to one of the following cells:

Cells

(i, k)

with

j < k <= grid[i][j] + j

(rightward movement), or

Cells

(k, j)

with

i < k <= grid[i][j] + i

(downward movement).

Return

the minimum number of cells you need to visit to reach the

bottom-right

cell

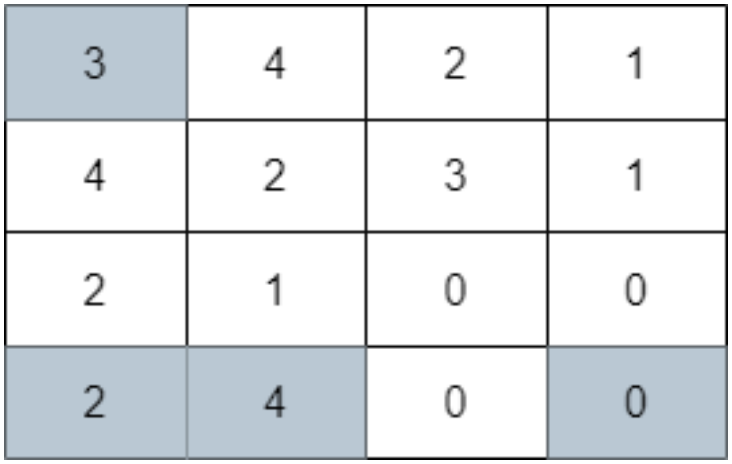
(m - 1, n - 1)

. If there is no valid path, return

-1

.

Example 1:



Input:

grid = [[3,4,2,1],[4,2,3,1],[2,1,0,0],[2,4,0,0]]

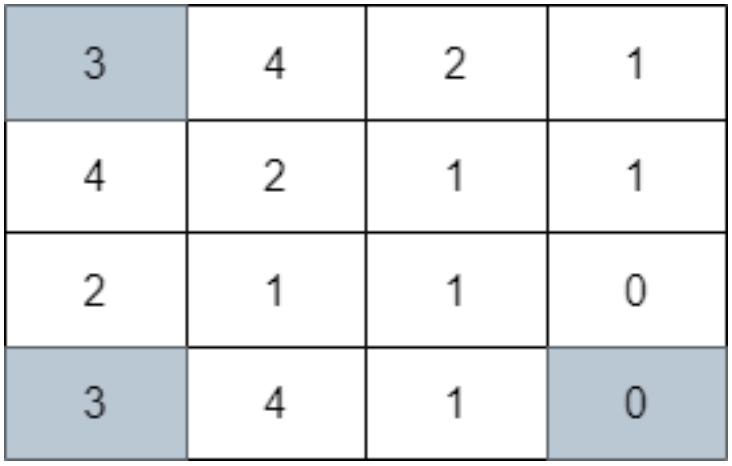
Output:

4

Explanation:

The image above shows one of the paths that visits exactly 4 cells.

Example 2:



Input:

grid = [[3,4,2,1],[4,2,1,1],[2,1,1,0],[3,4,1,0]]

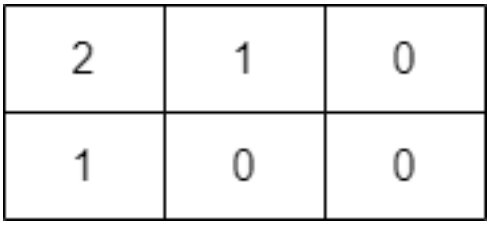
Output:

3

Explanation:

The image above shows one of the paths that visits exactly 3 cells.

Example 3:



Input:

grid = [[2,1,0],[1,0,0]]

Output:

-1

Explanation:

It can be proven that no path exists.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 10

5

1 <= m * n <= 10

5

0 <= grid[i][j] < m * n

grid[m - 1][n - 1] == 0

Code Snippets

C++:

```
class Solution {
public:
int minimumVisitedCells(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int minimumVisitedCells(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def minimumVisitedCells(self, grid: List[List[int]]) -> int:
```

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number}
*/
var minimumVisitedCells = function(grid) {

};
```

TypeScript:

```
function minimumVisitedCells(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumVisitedCells(int[][] grid) {

}
}
```

C:

```
int minimumVisitedCells(int** grid, int gridSize, int* gridColSize){

}
```

Go:

```
func minimumVisitedCells(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumVisitedCells(grid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumVisitedCells(_ grid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_visited_cells(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def minimum_visited_cells(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function minimumVisitedCells($grid) {

}
}
```

Dart:

```
class Solution {
int minimumVisitedCells(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def minimumVisitedCells(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_visited_cells(grid :: [[integer]]) :: integer
def minimum_visited_cells(grid) do

end
end
```

Erlang:

```
-spec minimum_visited_cells(Grid :: [[integer()]]) -> integer().
minimum_visited_cells(Grid) ->
.
```

Racket:

```
(define/contract (minimum-visited-cells grid)
(-> (listof (listof exact-integer?)) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Visited Cells in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public:
int minimumVisitedCells(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Visited Cells in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumVisitedCells(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Visited Cells in a Grid
Difficulty: Hard
Tags: array, graph, dp, search, stack, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumVisitedCells(self, grid: List[List[int]]) -> int:
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Visited Cells in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} grid
* @return {number}
*/
var minimumVisitedCells = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Visited Cells in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumVisitedCells(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Visited Cells in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimumVisitedCells(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Visited Cells in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, search, stack, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumVisitedCells(int** grid, int gridSize, int* gridColSize){

}
```

Go Solution:

```
// Problem: Minimum Number of Visited Cells in a Grid
// Difficulty: Hard
// Tags: array, graph, dp, search, stack, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumVisitedCells(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumVisitedCells(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumVisitedCells(_ grid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Visited Cells in a Grid
// Difficulty: Hard
// Tags: array, graph, dp, search, stack, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_visited_cells(grid: Vec<Vec<i32>>) -> i32 {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def minimum_visited_cells(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function minimumVisitedCells($grid) {

}
}
```

Dart Solution:

```
class Solution {
int minimumVisitedCells(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def minimumVisitedCells(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_visited_cells(grid :: [[integer]]) :: integer
def minimum_visited_cells(grid) do

end
end
```

Erlang Solution:

```
-spec minimum_visited_cells(Grid :: [[integer()]]) -> integer().
minimum_visited_cells(Grid) ->
.
```

Racket Solution:

```
(define/contract (minimum-visited-cells grid)
(-> (listof (listof exact-integer?)) exact-integer?)

)
```