Problem 2684: Maximum Number of Moves in a Grid

Problem Information

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

Problem Description

You are given a

0-indexed

m x n

matrix

grid

consisting of

positive

integers.

You can start at

any

cell in the first column of the matrix, and traverse the grid in the following way:

From a cell

(row, col)

, you can move to any of the cells:

(row - 1, col + 1)

,

(row, col + 1)

and

(row + 1, col + 1)

such that the value of the cell you move to, should be

strictly

bigger than the value of the current cell.

Return

the

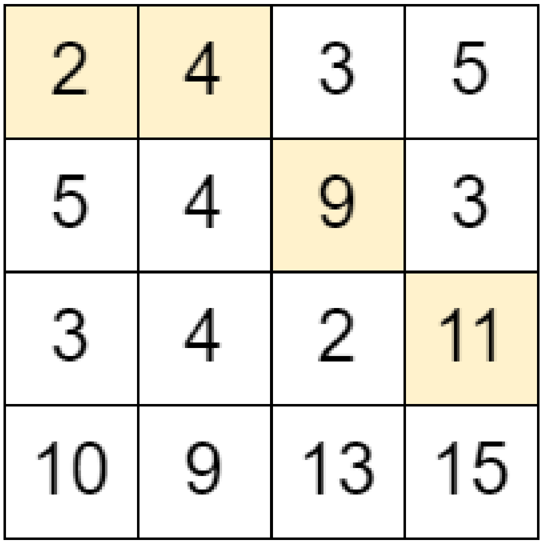
maximum

number of

moves

that you can perform.

Example 1:



2	4	3	5
5	4	9	3
3	4	2	11
10	9	13	15

Input:

grid = [[2,4,3,5],[5,4,9,3],[3,4,2,11],[10,9,13,15]]

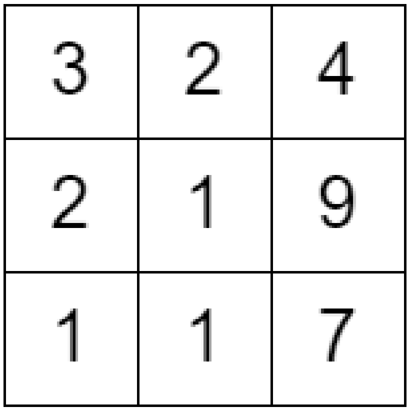
Output:

3

Explanation:

We can start at the cell (0, 0) and make the following moves: - (0, 0) -> (0, 1). - (0, 1) -> (1, 2). - (1, 2) -> (2, 3). It can be shown that it is the maximum number of moves that can be made.

Example 2:



3	2	4
2	1	9
1	1	7

Input:

grid = [[3,2,4],[2,1,9],[1,1,7]]

Output:

0

Explanation:

Starting from any cell in the first column we cannot perform any moves.

Constraints:

m == grid.length

n == grid[i].length

2 <= m, n <= 1000

4 <= m * n <= 10

5

1 <= grid[i][j] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-moves grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Moves in a Grid
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxMoves(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Moves in a Grid
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxMoves(int[][] grid) {
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Moves in a Grid
Difficulty: Medium
Tags: array, dp

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 maxMoves(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Moves in a Grid
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxMoves = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Moves in a Grid
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxMoves(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Moves in a Grid
* Difficulty: Medium
* Tags: array, dp
*
* 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 MaxMoves(int[][] grid) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Number of Moves in a Grid
* Difficulty: Medium
* Tags: array, dp
*
* 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 maxMoves(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Moves in a Grid
// Difficulty: Medium
// Tags: array, dp
//
// 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 maxMoves(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Moves in a Grid
// Difficulty: Medium
// Tags: array, dp
//
// 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 max_moves(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-moves grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```