Problem 3148: Maximum Difference Score in a Grid

Problem Information

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

Problem Description

You are given an

m x n

matrix

grid

consisting of

positive

integers. You can move from a cell in the matrix to

any

other cell that is either to the bottom or to the right (not necessarily adjacent). The score of a move from a cell with the value

c1

to a cell with the value

c2

is

c2 - c1

.

You can start at

any

cell, and you have to make

at least

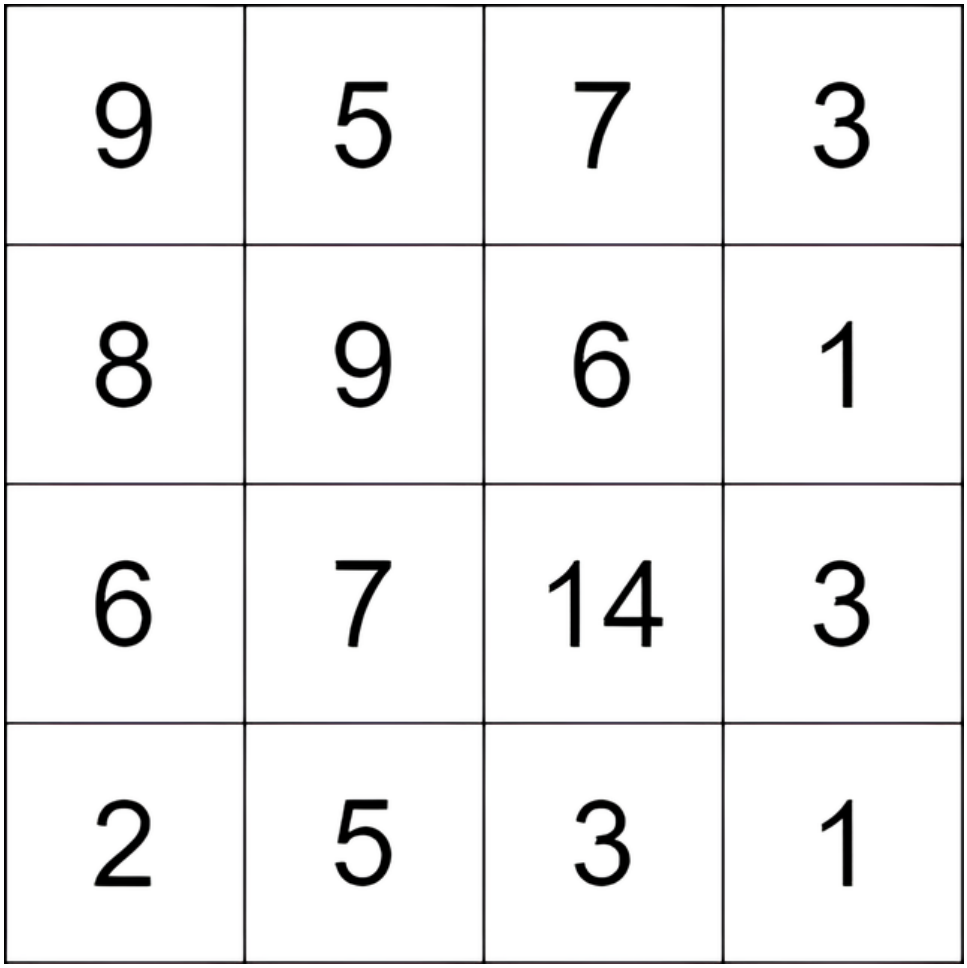
one move.

Return the

maximum

total score you can achieve.

Example 1:



Input:

grid = [[9,5,7,3],[8,9,6,1],[6,7,14,3],[2,5,3,1]]

Output:

9

Explanation:

We start at the cell

(0, 1)

, and we perform the following moves:

- Move from the cell

(0, 1)

to

(2, 1)

with a score of

7 - 5 = 2

.

- Move from the cell

(2, 1)

to

(2, 2)

with a score of

14 - 7 = 7

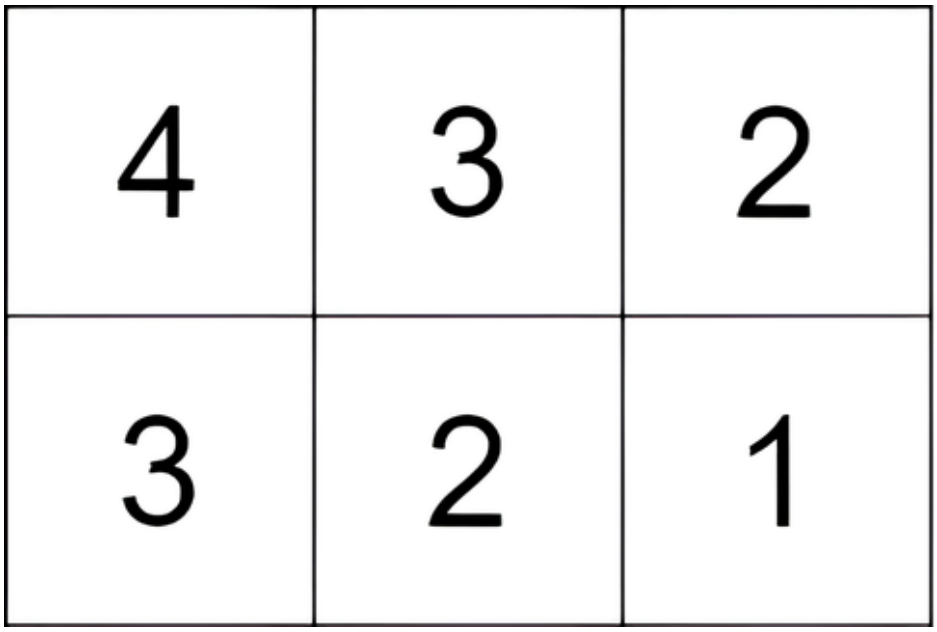
.

The total score is

2 + 7 = 9

.

Example 2:



Input:

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

Output:

-1

Explanation:

We start at the cell

(0, 0)

, and we perform one move:

(0, 0)

to

(0, 1)

. The score is

3 - 4 = -1

.

Constraints:

m == grid.length

n == grid[i].length

2 <= m, n <= 1000

4 <= m * n <= 10

5

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

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maxScore(List<List<Integer>> grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MaxScore(IList<IList<int>> grid) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun maxScore(grid: List<List<Int>>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
```

```
*/
function maxScore($grid) {

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def maxScore(grid: List[List[Int]]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Difference Score 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 maxScore(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Difference Score 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 maxScore(List<List<Integer>> grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Difference Score 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 maxScore(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Difference Score 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 maxScore = function(grid) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Difference Score 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 maxScore(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Difference Score 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 MaxScore(IList<IList<int>> grid) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Difference Score 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 maxScore(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Maximum Difference Score 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 maxScore(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxScore(grid: List<List<Int>>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Difference Score 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_score(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

```
object Solution {
def maxScore(grid: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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