Problem 2017: Grid Game

Problem Information

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

Problem Description

You are given a

0-indexed

2D array

grid

of size

2 x n

, where

grid[r][c]

represents the number of points at position

(r, c)

on the matrix. Two robots are playing a game on this matrix.

Both robots initially start at

(0, 0)

and want to reach

(1, n-1)

. Each robot may only move to the

right

(

(r, c)

to

(r, c + 1)

) or

down

(

(r, c)

to

(r + 1, c)

).

At the start of the game, the

first

robot moves from

(0, 0)

to

(1, n-1)

, collecting all the points from the cells on its path. For all cells

(r, c)

traversed on the path,

grid[r][c]

is set to

0

. Then, the

second

robot moves from

(0, 0)

to

(1, n-1)

, collecting the points on its path. Note that their paths may intersect with one another.

The

first

robot wants to

minimize

the number of points collected by the

second

robot. In contrast, the

second

robot wants to

maximize

the number of points it collects. If both robots play

optimally

, return

the

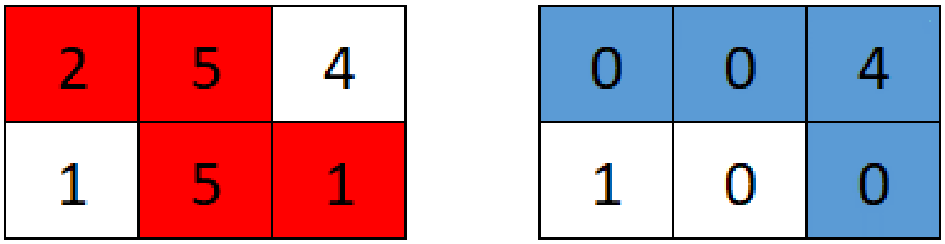
number of points

collected by the

second

robot.

Example 1:



Input:

grid = [[2,5,4],[1,5,1]]

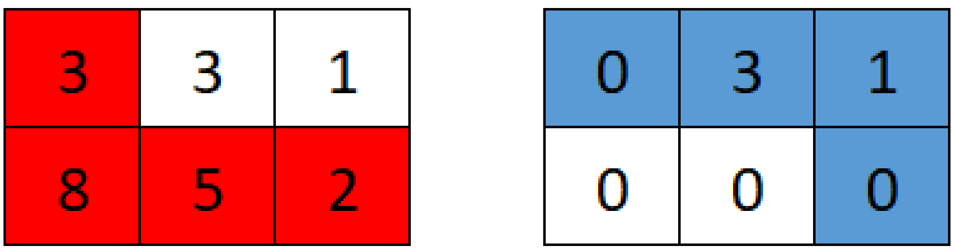
Output:

4

Explanation:

The optimal path taken by the first robot is shown in red, and the optimal path taken by the second robot is shown in blue. The cells visited by the first robot are set to 0. The second robot will collect 0 + 0 + 4 + 0 = 4 points.

Example 2:



Input:

grid = [[3,3,1],[8,5,2]]

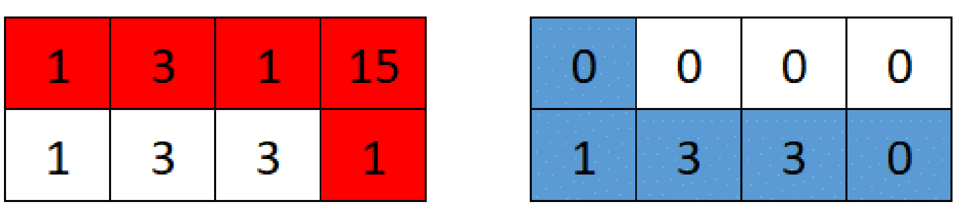
Output:

4

Explanation:

The optimal path taken by the first robot is shown in red, and the optimal path taken by the second robot is shown in blue. The cells visited by the first robot are set to 0. The second robot will collect 0 + 3 + 1 + 0 = 4 points.

Example 3:



Input:

grid = [[1,3,1,15],[1,3,3,1]]

Output:

7

Explanation:

The optimal path taken by the first robot is shown in red, and the optimal path taken by the second robot is shown in blue. The cells visited by the first robot are set to 0. The second robot will collect 0 + 1 + 3 + 3 + 0 = 7 points.

Constraints:

grid.length == 2

n == grid[r].length

1 <= n <= 5 * 10

4

1 <= grid[r][c] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long gridGame(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long GridGame(int[][] grid) {

}
}
```

C:

```
long long gridGame(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func gridGame(grid [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun gridGame(grid: Array<IntArray>): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn grid_game(grid: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Grid Game
* Difficulty: Medium
* Tags: array
*
* 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:
long long gridGame(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Grid Game
* Difficulty: Medium
* Tags: array
*
* 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 long gridGame(int[][] grid) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Grid Game
Difficulty: Medium
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Grid Game
* Difficulty: Medium
* Tags: array
*
* 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[][]} grid
* @return {number}
*/
var gridGame = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Grid Game
* Difficulty: Medium
* Tags: array
*
* 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 gridGame(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Grid Game
* Difficulty: Medium
* Tags: array
*
* 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 long GridGame(int[][] grid) {

}
```

```
}
```

C Solution:

```c
/*
* Problem: Grid Game
* Difficulty: Medium
* Tags: array
*
* 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
*/

long long gridGame(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```go
// Problem: Grid Game
// Difficulty: Medium
// Tags: array
//
// 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 gridGame(grid [][]int) int64 {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun gridGame(grid: Array<IntArray>): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Grid Game
// Difficulty: Medium
// Tags: array
//
// 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 grid_game(grid: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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