Problem 2428: Maximum Sum of an Hourglass

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

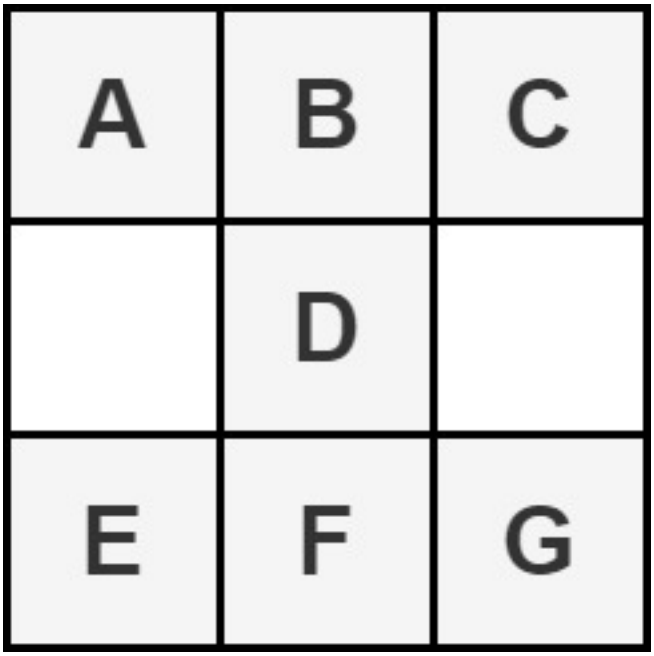
grid

.

We define an

hourglass

as a part of the matrix with the following form:



Return

the

maximum

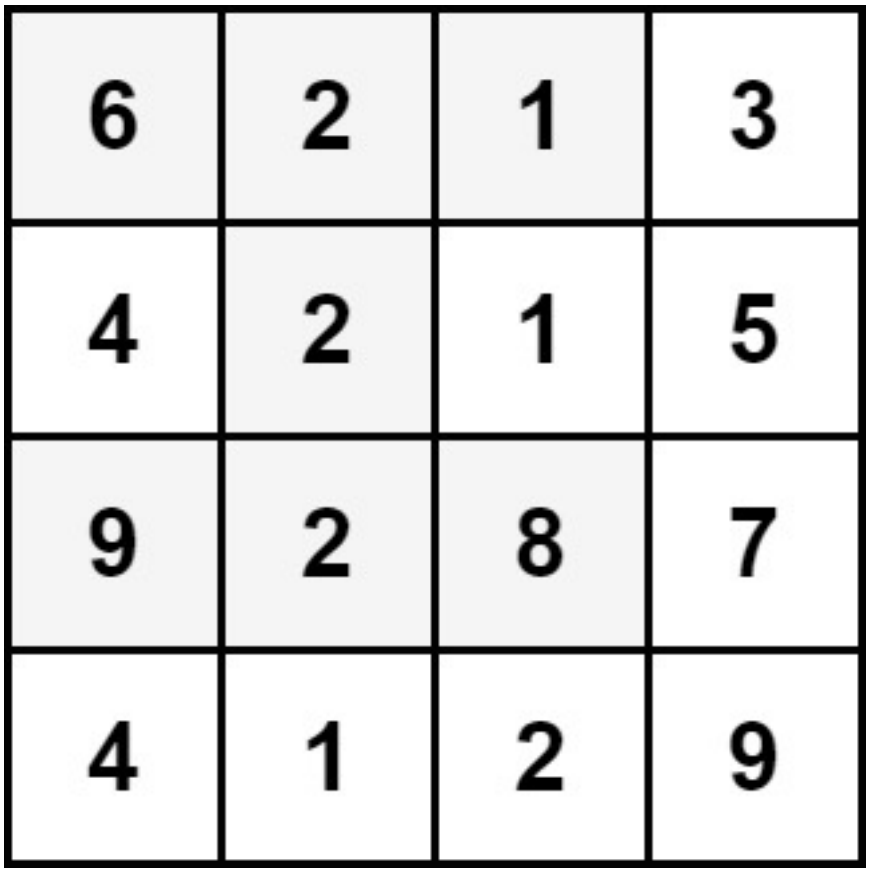
sum of the elements of an hourglass

.

Note

that an hourglass cannot be rotated and must be entirely contained within the matrix.

Example 1:



Input:

grid = [[6,2,1,3],[4,2,1,5],[9,2,8,7],[4,1,2,9]]

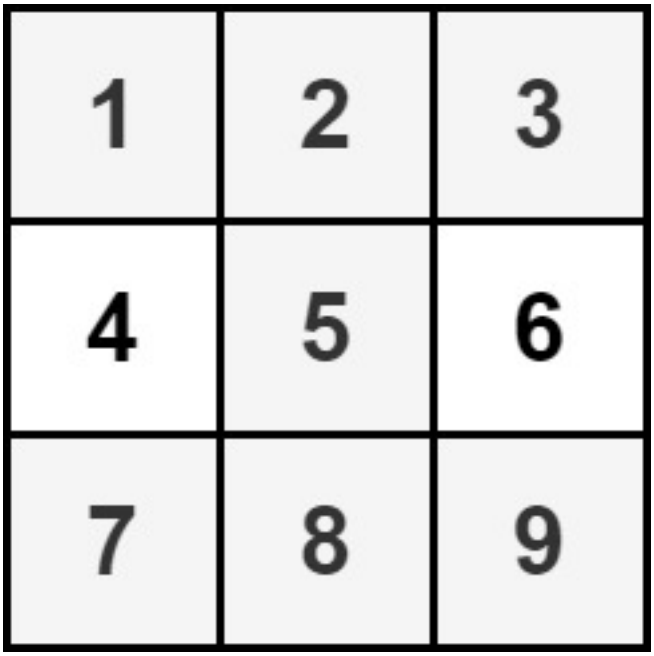
Output:

30

Explanation:

The cells shown above represent the hourglass with the maximum sum: 6 + 2 + 1 + 2 + 9 + 2 + 8 = 30.

Example 2:



Input:

grid = [[1,2,3],[4,5,6],[7,8,9]]

Output:

35

Explanation:

There is only one hourglass in the matrix, with the sum: 1 + 2 + 3 + 5 + 7 + 8 + 9 = 35.

Constraints:

m == grid.length

n == grid[i].length

3 <= m, n <= 150

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

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum of an Hourglass
* 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:
int maxSum(vector<vector<int>>& grid) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum of an Hourglass
* 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 int maxSum(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum of an Hourglass
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 maxSum(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum of an Hourglass
* 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 maxSum = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum of an Hourglass
* 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 maxSum(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum of an Hourglass
* 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 int MaxSum(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum of an Hourglass
* 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
*/

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

}
```

Go Solution:

```
// Problem: Maximum Sum of an Hourglass
// 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 maxSum(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum of an Hourglass
// 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 max_sum(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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