Problem 3225: Maximum Score From Grid Operations

Problem Information

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

Problem Description

You are given a 2D matrix

grid

of size

n x n

. Initially, all cells of the grid are colored white. In one operation, you can select any cell of indices

(i, j)

, and color black all the cells of the

j

th

column starting from the top row down to the

i

th

row.

The grid score is the sum of all

grid[i][j]

such that cell

(i, j)

is white and it has a horizontally adjacent black cell.

Return the

maximum

score that can be achieved after some number of operations.

Example 1:

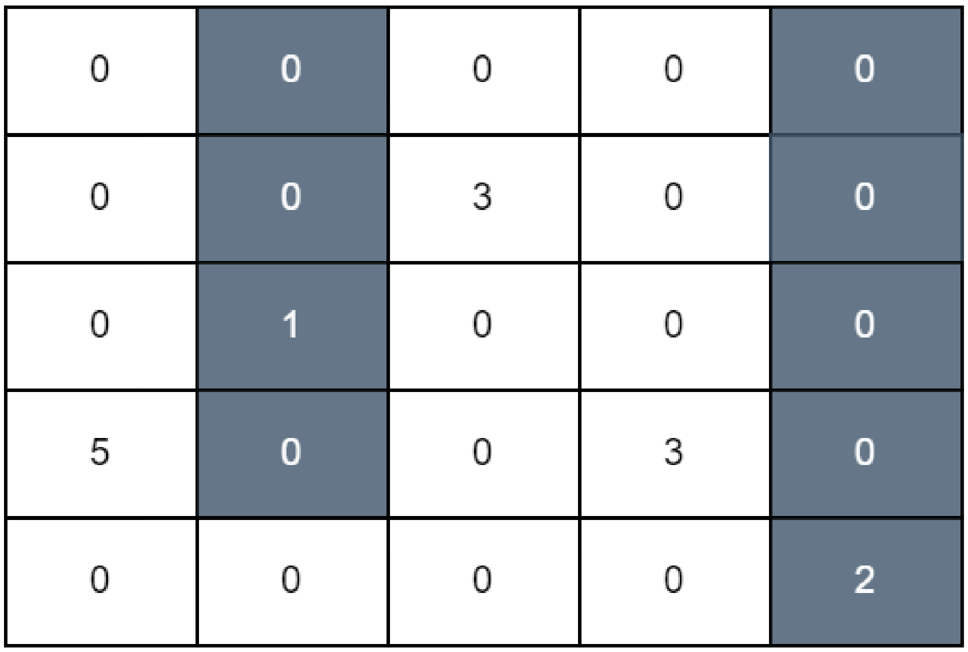
Input:

grid = [[0,0,0,0,0],[0,0,3,0,0],[0,1,0,0,0],[5,0,0,3,0],[0,0,0,0,2]]

Output:

11

Explanation:



0	0	0	0	0
0	0	3	0	0
0	1	0	0	0
5	0	0	3	0
0	0	0	0	2

In the first operation, we color all cells in column 1 down to row 3, and in the second operation, we color all cells in column 4 down to the last row. The score of the resulting grid is

grid[3][0] + grid[1][2] + grid[3][3]

which is equal to 11.

Example 2:

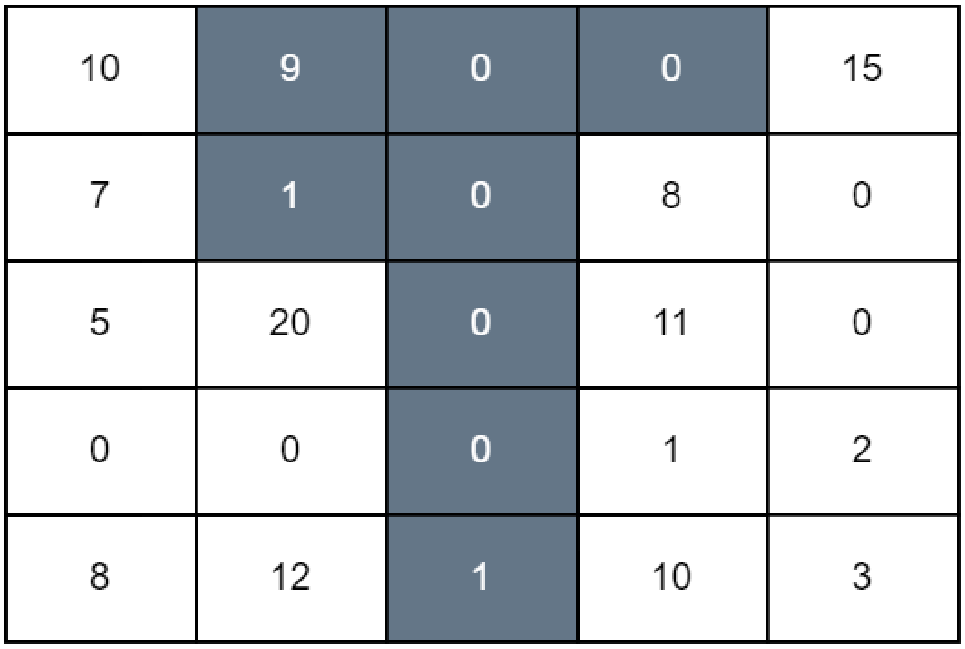
Input:

grid = [[10,9,0,0,15],[7,1,0,8,0],[5,20,0,11,0],[0,0,0,1,2],[8,12,1,10,3]]

Output:

94

Explanation:



10	9	0	0	15
7	1	0	8	0
5	20	0	11	0
0	0	0	1	2
8	12	1	10	3

We perform operations on 1, 2, and 3 down to rows 1, 4, and 0, respectively. The score of the resulting grid is

grid[0][0] + grid[1][0] + grid[2][1] + grid[4][1] + grid[1][3] + grid[2][3] + grid[3][3] + grid[4][3] + grid[0][4]

which is equal to 94.

Constraints:

1 <= n == grid.length <= 100

n == grid[i].length

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

9

Code Snippets

C++:

```
class Solution {
public:
```

```
long long maximumScore(vector<vector<int>>& grid) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Score From Grid Operations
* Difficulty: Hard
* 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:
long long maximumScore(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Score From Grid Operations
* Difficulty: Hard
* 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 long maximumScore(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Score From Grid Operations
Difficulty: Hard
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 maximumScore(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Score From Grid Operations
* Difficulty: Hard
* 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 maximumScore = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Score From Grid Operations
* Difficulty: Hard
* 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 maximumScore(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Score From Grid Operations
* Difficulty: Hard
* 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 long MaximumScore(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Score From Grid Operations
* Difficulty: Hard
* 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
*/

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

}
```

Go Solution:

```
// Problem: Maximum Score From Grid Operations
// Difficulty: Hard
// 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 maximumScore(grid [][]int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Score From Grid Operations
// Difficulty: Hard
// 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 maximum_score(grid: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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