Problem 3257: Maximum Value Sum by Placing Three Rooks II

Problem Information

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

Problem Description

You are given a

m x n

2D array

board

representing a chessboard, where

board[i][j]

represents the

value

of the cell

(i, j)

.

Rooks in the

same

row or column

attack

each other. You need to place

three

rooks on the chessboard such that the rooks

do not

attack

each other.

Return the

maximum

sum of the cell

values

on which the rooks are placed.

Example 1:

Input:

board =

[[-3,1,1,1],[-3,1,-3,1],[-3,2,1,1]]

Output:

4

Explanation:

-3	1	1	1
-3	1	-3	1
-3	2	1	1

-3	1		1
-3	1	-3	
-3		1	1

We can place the rooks in the cells

(0, 2)

,

(1, 3)

, and

(2, 1)

for a sum of

1 + 1 + 2 = 4

.

Example 2:

Input:

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

Output:

15

Explanation:

We can place the rooks in the cells

(0, 0)

,

(1, 1)

, and

(2, 2)

for a sum of

1 + 5 + 9 = 15

.

Example 3:

Input:

board = [[1,1,1],[1,1,1],[1,1,1]]

Output:

3

Explanation:

We can place the rooks in the cells

(0, 2)

,

(1, 1)

, and

(2, 0)

for a sum of

1 + 1 + 1 = 3

.

Constraints:

3 <= m == board.length <= 500

3 <= n == board[i].length <= 500

-10

9

<= board[i][j] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long maximumValueSum(vector<vector<int>>& board) {

}
};
```

Java:

```
class Solution {
public long maximumValueSum(int[][] board) {

}
}
```

Python3:

```
class Solution:
def maximumValueSum(self, board: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumValueSum(board: number[][]): number {

};
```

C#:

```
public class Solution {
public long MaximumValueSum(int[][] board) {

}
}
```

C:

```
long long maximumValueSum(int** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func maximumValueSum(board [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumValueSum(board: Array<IntArray>): Long {

}
}
```

Swift:

```
class Solution {
func maximumValueSum(_ board: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_value_sum(board: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[][]} board
# @return {Integer}
def maximum_value_sum(board)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $board
* @return Integer
*/
function maximumValueSum($board) {

}
}
```

Dart:

```
class Solution {
int maximumValueSum(List<List<int>> board) {

}
```

```
}
```

Scala:

```
object Solution {
def maximumValueSum(board: Array[Array[Int]]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_value_sum(board :: [[integer]]) :: integer
def maximum_value_sum(board) do

end
end
```

Erlang:

```
-spec maximum_value_sum(Board :: [[integer()]]) -> integer().
maximum_value_sum(Board) ->
.
```

Racket:

```
(define/contract (maximum-value-sum board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Value Sum by Placing Three Rooks II
* 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 maximumValueSum(vector<vector<int>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Value Sum by Placing Three Rooks II
* 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 maximumValueSum(int[][] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Value Sum by Placing Three Rooks II
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 maximumValueSum(self, board: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Value Sum by Placing Three Rooks II
* 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[][]} board
* @return {number}
*/
var maximumValueSum = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Value Sum by Placing Three Rooks II
* 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 maximumValueSum(board: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Value Sum by Placing Three Rooks II
* 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 MaximumValueSum(int[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Value Sum by Placing Three Rooks II
* 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 maximumValueSum(int** board, int boardSize, int* boardColSize) {
```

```
}
```

Go Solution:

```
// Problem: Maximum Value Sum by Placing Three Rooks II
// 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 maximumValueSum(board [][]int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumValueSum(board: Array<IntArray>): Long {

}
}
```

Swift Solution:

```
class Solution {
func maximumValueSum(_ board: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Value Sum by Placing Three Rooks II
// 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_value_sum(board: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} board
# @return {Integer}
def maximum_value_sum(board)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $board
* @return Integer
*/
function maximumValueSum($board) {

}
}
```

Dart Solution:

```
class Solution {
int maximumValueSum(List<List<int>> board) {

}
}
```

Scala Solution:

```
object Solution {
def maximumValueSum(board: Array[Array[Int]]): Long = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_value_sum(board :: [[integer]]) :: integer
def maximum_value_sum(board) do

end
end
```

Erlang Solution:

```
-spec maximum_value_sum(Board :: [[integer()]]) -> integer().
maximum_value_sum(Board) ->
.
```

Racket Solution:

```
(define/contract (maximum-value-sum board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```