Problem 3197: Find the Minimum Area to Cover All Ones II

Problem Information

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

Problem Description

You are given a 2D

binary

array

grid

. You need to find 3

non-overlapping

rectangles having

non-zero

areas with horizontal and vertical sides such that all the 1's in

grid

lie inside these rectangles.

Return the

minimum

possible sum of the area of these rectangles.

Note

that the rectangles are allowed to touch.

Example 1:

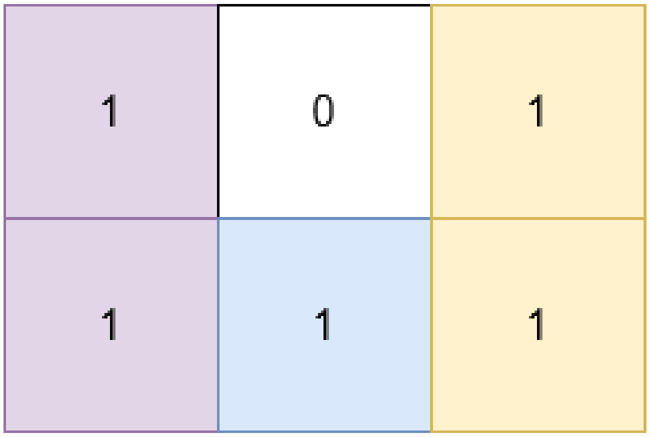
Input:

grid = [[1,0,1],[1,1,1]]

Output:

5

Explanation:



1	0	1
1	1	1

The 1's at

(0, 0)

and

(1, 0)

are covered by a rectangle of area 2.

The 1's at

(0, 2)

and

(1, 2)

are covered by a rectangle of area 2.

The 1 at

(1, 1)

is covered by a rectangle of area 1.

Example 2:

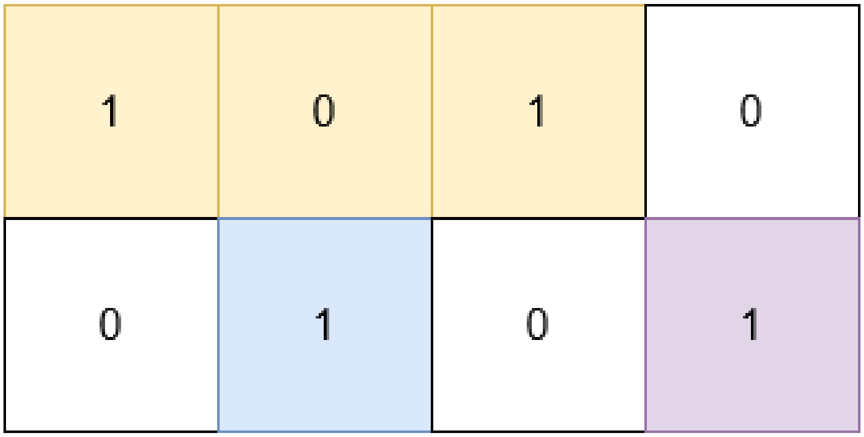
Input:

grid = [[1,0,1,0],[0,1,0,1]]

Output:

5

Explanation:



1	0	1	0
0	1	0	1

The 1's at

(0, 0)

and

(0, 2)

are covered by a rectangle of area 3.

The 1 at

(1, 1)

is covered by a rectangle of area 1.

The 1 at

(1, 3)

is covered by a rectangle of area 1.

Constraints:

1 <= grid.length, grid[i].length <= 30

grid[i][j]

is either 0 or 1.

The input is generated such that there are at least three 1's in

grid

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}

}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Minimum Area to Cover All Ones II
* Difficulty: Hard
* 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 minimumSum(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Minimum Area to Cover All Ones II
* Difficulty: Hard
* 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 minimumSum(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Minimum Area to Cover All Ones II
Difficulty: Hard
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 minimumSum(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Minimum Area to Cover All Ones II
* Difficulty: Hard
* 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 minimumSum = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Minimum Area to Cover All Ones II
* Difficulty: Hard
* 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 minimumSum(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Minimum Area to Cover All Ones II
* Difficulty: Hard
* 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 MinimumSum(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Find the Minimum Area to Cover All Ones II
* Difficulty: Hard
* 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 minimumSum(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Find the Minimum Area to Cover All Ones II
// Difficulty: Hard
// 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 minimumSum(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Minimum Area to Cover All Ones II
// Difficulty: Hard
// 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 minimum_sum(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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