Problem 3195: Find the Minimum Area to Cover All Ones I

Problem Information

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

Problem Description

You are given a 2D

binary

array

grid

. Find a rectangle with horizontal and vertical sides with the

smallest

area, such that all the 1's in

grid

lie inside this rectangle.

Return the

minimum

possible area of the rectangle.

Example 1:

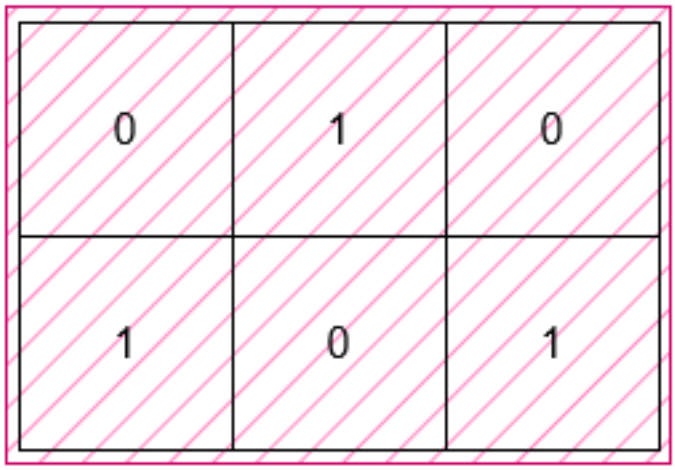
Input:

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

Output:

6

Explanation:



The smallest rectangle has a height of 2 and a width of 3, so it has an area of

2 * 3 = 6

.

Example 2:

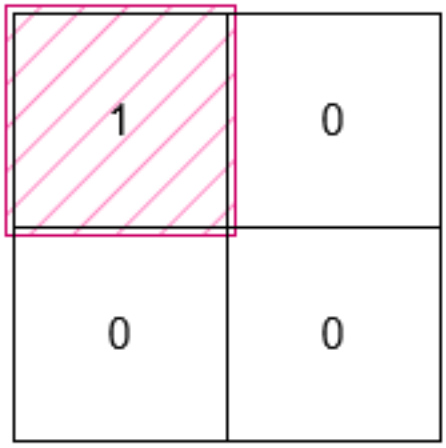
Input:

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

Output:

1

Explanation:



The smallest rectangle has both height and width 1, so its area is

1 * 1 = 1

.

Constraints:

1 <= grid.length, grid[i].length <= 1000

grid[i][j]

is either 0 or 1.

The input is generated such that there is at least one 1 in

grid

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Minimum Area to Cover All Ones I
* 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 minimumArea(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Minimum Area to Cover All Ones I
* 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 minimumArea(int[][] grid) {

}
}
```

Python3 Solution:

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Minimum Area to Cover All Ones I
* 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 minimumArea = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Minimum Area to Cover All Ones I
* 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 minimumArea(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Minimum Area to Cover All Ones I
* 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 MinimumArea(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Find the Minimum Area to Cover All Ones I
* 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 minimumArea(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Find the Minimum Area to Cover All Ones I
// 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 minimumArea(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Minimum Area to Cover All Ones I
// 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 minimum_area(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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