Problem 2943: Maximize Area of Square Hole in Grid

Problem Information

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

Problem Description

You are given the two integers,

n

and

m

and two integer arrays,

hBars

and

vBars

. The grid has

n + 2

horizontal and

m + 2

vertical bars, creating 1 x 1 unit cells. The bars are indexed starting from

1

.

You can

remove

some of the bars in

hBars

from horizontal bars and some of the bars in

vBars

from vertical bars. Note that other bars are fixed and cannot be removed.

Return an integer denoting the

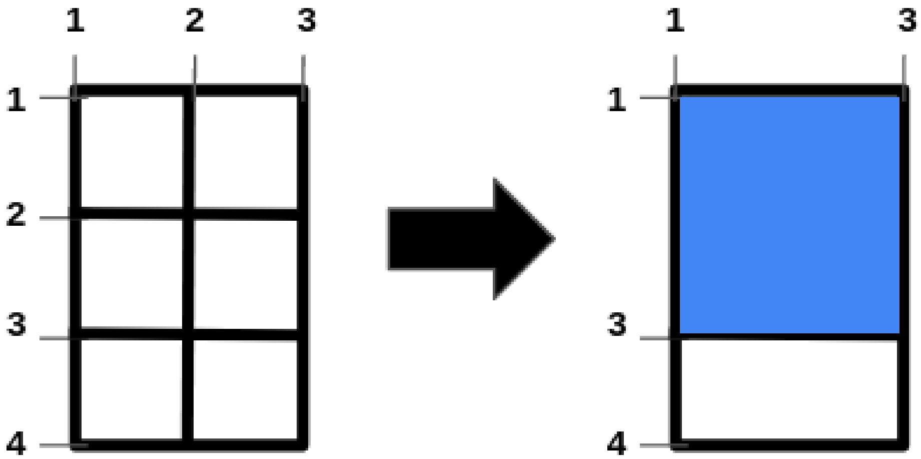
maximum area

of a

square-shaped

hole in the grid, after removing some bars (possibly none).

Example 1:



Input:

n = 2, m = 1, hBars = [2,3], vBars = [2]

Output:

4

Explanation:

The left image shows the initial grid formed by the bars. The horizontal bars are

[1,2,3,4]

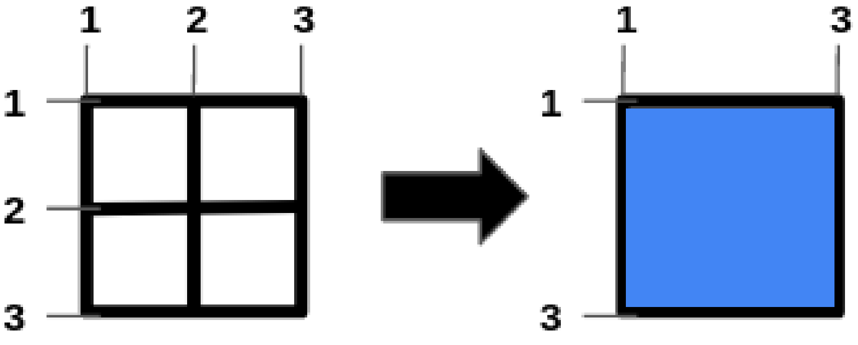
, and the vertical bars are

[1,2,3]

.

One way to get the maximum square-shaped hole is by removing horizontal bar 2 and vertical bar 2.

Example 2:



Input:

n = 1, m = 1, hBars = [2], vBars = [2]

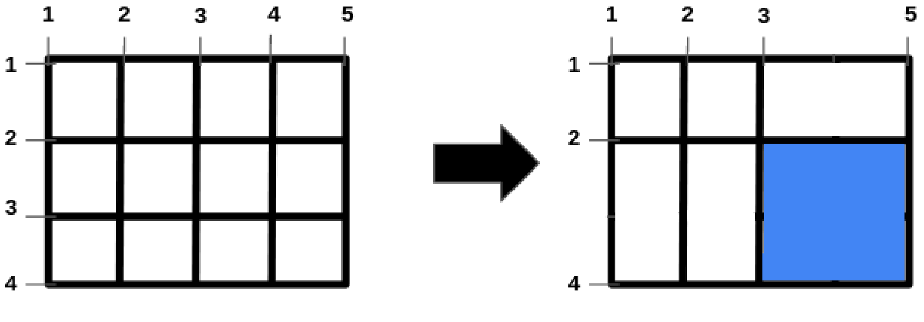
Output:

4

Explanation:

To get the maximum square-shaped hole, we remove horizontal bar 2 and vertical bar 2.

Example 3:



Input:

n = 2, m = 3, hBars = [2,3], vBars = [2,4]

Output:

4

Explanation:

One way to get the maximum square-shaped hole is by removing horizontal bar 3, and vertical bar 4.

Constraints:

1 <= n <= 10

9

1 <= m <= 10

9

1 <= hBars.length <= 100

2 <= hBars[i] <= n + 1

1 <= vBars.length <= 100

2 <= vBars[i] <= m + 1

All values in

hBars

are distinct.

All values in

vBars

are distinct.

Code Snippets

C++:

```
class Solution {
public:
int maximizeSquareHoleArea(int n, int m, vector<int>& hBars, vector<int>&
vBars) {

}
};
```

Java:

```
class Solution {
public int maximizeSquareHoleArea(int n, int m, int[] hBars, int[] vBars) {

}
}
```

Python3:

```
class Solution:
def maximizeSquareHoleArea(self, n: int, m: int, hBars: List[int], vBars:
List[int]) -> int:
```

Python:

```
class Solution(object):
def maximizeSquareHoleArea(self, n, m, hBars, vBars):
"""
:type n: int
:type m: int
:type hBars: List[int]
:type vBars: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @param {number[]} hBars
* @param {number[]} vBars
* @return {number}
*/
```

```
var maximizeSquareHoleArea = function(n, m, hBars, vBars) {

};
```

TypeScript:

```
function maximizeSquareHoleArea(n: number, m: number, hBars: number[], vBars:
number[]): number {

};
```

C#:

```
public class Solution {
public int MaximizeSquareHoleArea(int n, int m, int[] hBars, int[] vBars) {

}
}
```

C:

```
int maximizeSquareHoleArea(int n, int m, int* hBars, int hBarsSize, int*
vBars, int vBarsSize) {

}
```

Go:

```
func maximizeSquareHoleArea(n int, m int, hBars []int, vBars []int) int {

}
```

Kotlin:

```
class Solution {
fun maximizeSquareHoleArea(n: Int, m: Int, hBars: IntArray, vBars: IntArray):
Int {

}
}
```

Swift:

```
class Solution {
func maximizeSquareHoleArea(_ n: Int, _ m: Int, _ hBars: [Int], _ vBars:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximize_square_hole_area(n: i32, m: i32, h_bars: Vec<i32>, v_bars:
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer[]} h_bars
# @param {Integer[]} v_bars
# @return {Integer}
def maximize_square_hole_area(n, m, h_bars, v_bars)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer[] $hBars
* @param Integer[] $vBars
* @return Integer
*/
function maximizeSquareHoleArea($n, $m, $hBars, $vBars) {

}
}
```

Dart:

```
class Solution {
int maximizeSquareHoleArea(int n, int m, List<int> hBars, List<int> vBars) {

}
}
```

Scala:

```
object Solution {
def maximizeSquareHoleArea(n: Int, m: Int, hBars: Array[Int], vBars:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximize_square_hole_area(n :: integer, m :: integer, h_bars ::
[integer], v_bars :: [integer]) :: integer
def maximize_square_hole_area(n, m, h_bars, v_bars) do

end
end
```

Erlang:

```
-spec maximize_square_hole_area(N :: integer(), M :: integer(), HBars ::
[integer()], VBars :: [integer()]) -> integer().
maximize_square_hole_area(N, M, HBars, VBars) ->
.
```

Racket:

```
(define/contract (maximize-square-hole-area n m hBars vBars)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Area of Square Hole in Grid
* Difficulty: Medium
* Tags: array, sort
*
* 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 maximizeSquareHoleArea(int n, int m, vector<int>& hBars, vector<int>& vBars) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Area of Square Hole in Grid
* Difficulty: Medium
* Tags: array, sort
*
* 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 maximizeSquareHoleArea(int n, int m, int[] hBars, int[] vBars) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Area of Square Hole in Grid
Difficulty: Medium
Tags: array, sort

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 maximizeSquareHoleArea(self, n: int, m: int, hBars: List[int], vBars:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximizeSquareHoleArea(self, n, m, hBars, vBars):
"""
:type n: int
:type m: int
:type hBars: List[int]
:type vBars: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Area of Square Hole in Grid
* Difficulty: Medium
* Tags: array, sort
*
* 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} n
* @param {number} m
* @param {number[]} hBars
* @param {number[]} vBars
* @return {number}
*/
var maximizeSquareHoleArea = function(n, m, hBars, vBars) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Area of Square Hole in Grid
* Difficulty: Medium
* Tags: array, sort
*
* 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 maximizeSquareHoleArea(n: number, m: number, hBars: number[], vBars:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Area of Square Hole in Grid
* Difficulty: Medium
* Tags: array, sort
*
* 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 MaximizeSquareHoleArea(int n, int m, int[] hBars, int[] vBars) {
}
}
```

C Solution:

```c
/*
* Problem: Maximize Area of Square Hole in Grid
* Difficulty: Medium
* Tags: array, sort
*
* 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 maximizeSquareHoleArea(int n, int m, int* hBars, int hBarsSize, int*
vBars, int vBarsSize) {

}
```

Go Solution:

```go
// Problem: Maximize Area of Square Hole in Grid
// Difficulty: Medium
// Tags: array, sort
//
// 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 maximizeSquareHoleArea(n int, m int, hBars []int, vBars []int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maximizeSquareHoleArea(n: Int, m: Int, hBars: IntArray, vBars: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func maximizeSquareHoleArea(_ n: Int, _ m: Int, _ hBars: [Int], _ vBars:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Area of Square Hole in Grid
// Difficulty: Medium
// Tags: array, sort
//
// 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 maximize_square_hole_area(n: i32, m: i32, h_bars: Vec<i32>, v_bars:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer[]} h_bars
# @param {Integer[]} v_bars
# @return {Integer}
def maximize_square_hole_area(n, m, h_bars, v_bars)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer[] $hBars
* @param Integer[] $vBars
* @return Integer
*/
function maximizeSquareHoleArea($n, $m, $hBars, $vBars) {

}
}
```

Dart Solution:

```
class Solution {
int maximizeSquareHoleArea(int n, int m, List<int> hBars, List<int> vBars) {

}
}
```

Scala Solution:

```
object Solution {
def maximizeSquareHoleArea(n: Int, m: Int, hBars: Array[Int], vBars:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_square_hole_area(n :: integer, m :: integer, h_bars ::
[integer], v_bars :: [integer]) :: integer
def maximize_square_hole_area(n, m, h_bars, v_bars) do

end
end
```

Erlang Solution:

```
-spec maximize_square_hole_area(N :: integer(), M :: integer(), HBars ::
[integer()], VBars :: [integer()]) -> integer().
maximize_square_hole_area(N, M, HBars, VBars) ->
.
```

Racket Solution:

```
(define/contract (maximize-square-hole-area n m hBars vBars)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```