Problem 2975: Maximum Square Area by Removing Fences From a Field

Problem Information

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

Problem Description

There is a large

(m - 1) x (n - 1)

rectangular field with corners at

(1, 1)

and

(m, n)

containing some horizontal and vertical fences given in arrays

hFences

and

vFences

respectively.

Horizontal fences are from the coordinates

(hFences[i], 1)

to

(hFences[i], n)

and vertical fences are from the coordinates

(1, vFences[i])

to

(m, vFences[i])

.

Return

the

maximum

area of a

square

field that can be formed by

removing

some fences (

possibly none

) or

-1

if it is impossible to make a square field

.

Since the answer may be large, return it

modulo

10

9

+ 7

.

Note:

The field is surrounded by two horizontal fences from the coordinates

(1, 1)

to

(1, n)

and

(m, 1)

to

(m, n)

and two vertical fences from the coordinates

(1, 1)

to

(m, 1)

and

(1, n)

to

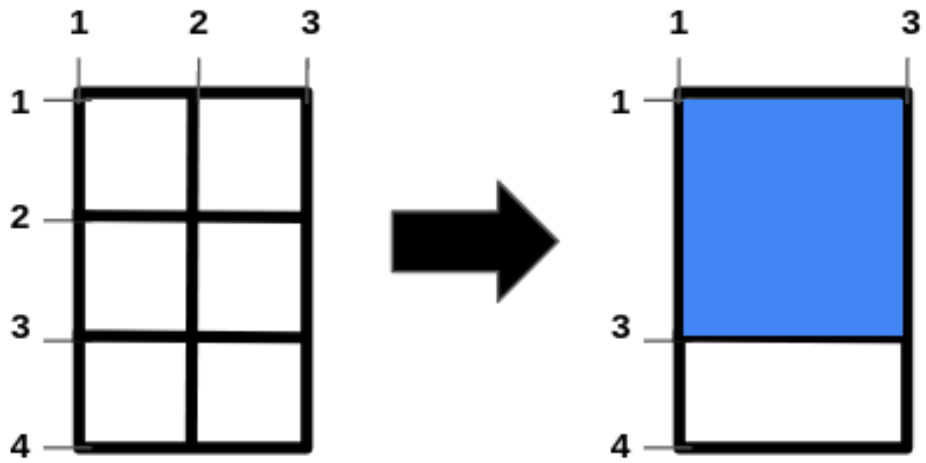
(m, n)

. These fences

cannot

be removed.

Example 1:



Input:

m = 4, n = 3, hFences = [2,3], vFences = [2]

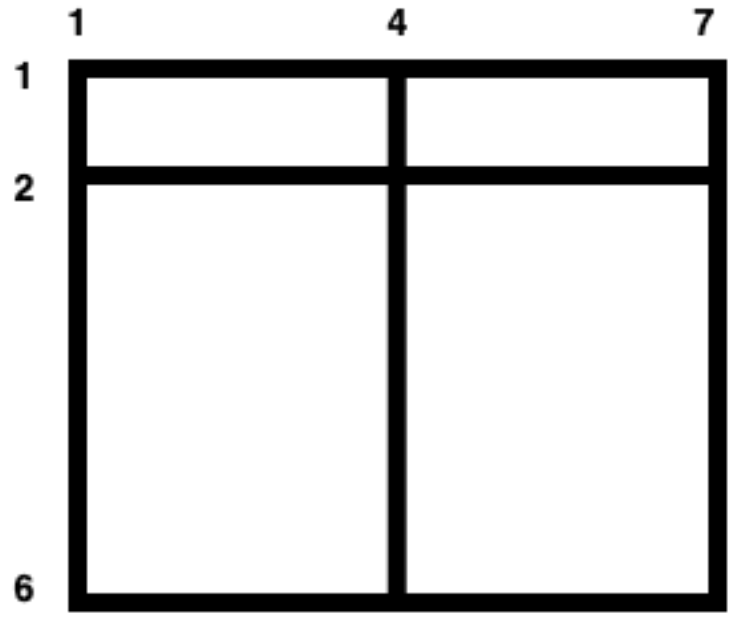
Output:

4

Explanation:

Removing the horizontal fence at 2 and the vertical fence at 2 will give a square field of area 4.

Example 2:



Input:

m = 6, n = 7, hFences = [2], vFences = [4]

Output:

-1

Explanation:

It can be proved that there is no way to create a square field by removing fences.

Constraints:

3 <= m, n <= 10

9

1 <= hF

ences

.length, vFences.length <= 600

1 < hFences[i] < m

1 < vFences[i] < n

hFences

and

vFences

are unique.

Code Snippets

C++:

```
class Solution {
public:
int maximizeSquareArea(int m, int n, vector<int>& hFences, vector<int>&
vFences) {

}
};
```

Java:

```
class Solution {
public int maximizeSquareArea(int m, int n, int[] hFences, int[] vFences) {

}
}
```

Python3:

```
class Solution:
def maximizeSquareArea(self, m: int, n: int, hFences: List[int], vFences:
List[int]) -> int:
```

Python:

```
class Solution(object):
def maximizeSquareArea(self, m, n, hFences, vFences):
"""
:type m: int
:type n: int
:type hFences: List[int]
:type vFences: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number[]} hFences
* @param {number[]} vFences
* @return {number}
*/
var maximizeSquareArea = function(m, n, hFences, vFences) {

};
```

TypeScript:

```
function maximizeSquareArea(m: number, n: number, hFences: number[], vFences:
number[]): number {

};
```

C#:

```
public class Solution {
public int MaximizeSquareArea(int m, int n, int[] hFences, int[] vFences) {

}
}
```

C:

```
int maximizeSquareArea(int m, int n, int* hFences, int hFencesSize, int*
vFences, int vFencesSize) {

}
```

Go:

```
func maximizeSquareArea(m int, n int, hFences []int, vFences []int) int {

}
```

Kotlin:

```
class Solution {
fun maximizeSquareArea(m: Int, n: Int, hFences: IntArray, vFences: IntArray):
Int {

}
}
```

Swift:

```
class Solution {
func maximizeSquareArea(_ m: Int, _ n: Int, _ hFences: [Int], _ vFences:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximize_square_area(m: i32, n: i32, h_fences: Vec<i32>, v_fences:
Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer[]} h_fences
# @param {Integer[]} v_fences
# @return {Integer}
def maximize_square_area(m, n, h_fences, v_fences)

end
```

PHP:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[] $hFences
* @param Integer[] $vFences
* @return Integer
*/
function maximizeSquareArea($m, $n, $hFences, $vFences) {

}
}
```

Dart:

```
class Solution {
int maximizeSquareArea(int m, int n, List<int> hFences, List<int> vFences) {

}
}
```

Scala:

```
object Solution {
def maximizeSquareArea(m: Int, n: Int, hFences: Array[Int], vFences:
Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximize_square_area(m :: integer, n :: integer, h_fences :: [integer],
v_fences :: [integer]) :: integer
def maximize_square_area(m, n, h_fences, v_fences) do

end
end
```

Erlang:

```
-spec maximize_square_area(M :: integer(), N :: integer(), HFences ::
[integer()], VFences :: [integer()]) -> integer().
maximize_square_area(M, N, HFences, VFences) ->
.
```

Racket:

```
(define/contract (maximize-square-area m n hFences vFences)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Square Area by Removing Fences From a Field
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int maximizeSquareArea(int m, int n, vector<int>& hFences, vector<int>&
vFences) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Square Area by Removing Fences From a Field
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int maximizeSquareArea(int m, int n, int[] hFences, int[] vFences) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Square Area by Removing Fences From a Field
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def maximizeSquareArea(self, m: int, n: int, hFences: List[int], vFences:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximizeSquareArea(self, m, n, hFences, vFences):
"""
:type m: int
:type n: int
:type hFences: List[int]
:type vFences: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Square Area by Removing Fences From a Field
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} m
* @param {number} n
* @param {number[]} hFences
* @param {number[]} vFences
* @return {number}
*/
var maximizeSquareArea = function(m, n, hFences, vFences) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Square Area by Removing Fences From a Field
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) for hash map
*/

function maximizeSquareArea(m: number, n: number, hFences: number[], vFences:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Square Area by Removing Fences From a Field
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MaximizeSquareArea(int m, int n, int[] hFences, int[] vFences) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Square Area by Removing Fences From a Field
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int maximizeSquareArea(int m, int n, int* hFences, int hFencesSize, int*
vFences, int vFencesSize) {
```

```
}
```

Go Solution:

```
// Problem: Maximum Square Area by Removing Fences From a Field
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maximizeSquareArea(m int, n int, hFences []int, vFences []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximizeSquareArea(m: Int, n: Int, hFences: IntArray, vFences: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func maximizeSquareArea(_ m: Int, _ n: Int, _ hFences: [Int], _ vFences:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Square Area by Removing Fences From a Field
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
```

```
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn maximize_square_area(m: i32, n: i32, h_fences: Vec<i32>, v_fences:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer[]} h_fences
# @param {Integer[]} v_fences
# @return {Integer}
def maximize_square_area(m, n, h_fences, v_fences)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[] $hFences
* @param Integer[] $vFences
* @return Integer
*/
function maximizeSquareArea($m, $n, $hFences, $vFences) {

}
}
```

Dart Solution:

```
class Solution {
int maximizeSquareArea(int m, int n, List<int> hFences, List<int> vFences) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maximizeSquareArea(m: Int, n: Int, hFences: Array[Int], vFences:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximize_square_area(m :: integer, n :: integer, h_fences :: [integer],
v_fences :: [integer]) :: integer
def maximize_square_area(m, n, h_fences, v_fences) do

end
end
```

Erlang Solution:

```
-spec maximize_square_area(M :: integer(), N :: integer(), HFences ::
[integer()], VFences :: [integer()]) -> integer().
maximize_square_area(M, N, HFences, VFences) ->
.
```

Racket Solution:

```
(define/contract (maximize-square-area m n hFences vFences)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```