Problem 3000: Maximum Area of Longest Diagonal Rectangle

Problem Information

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

Problem Description

You are given a 2D

0-indexed

integer array

dimensions

.

For all indices

i

,

0 <= i < dimensions.length

,

dimensions[i][0]

represents the length and

dimensions[i][1]

represents the width of the rectangle

i

.

Return

the

area

of the rectangle having the

longest

diagonal. If there are multiple rectangles with the longest diagonal, return the area of the rectangle having the

maximum

area.

Example 1:

Input:

dimensions = [[9,3],[8,6]]

Output:

48

Explanation:

For index = 0, length = 9 and width = 3. Diagonal length = sqrt(9 * 9 + 3 * 3) = sqrt(90) ≈

9.487. For index = 1, length = 8 and width = 6. Diagonal length = sqrt(8 * 8 + 6 * 6) = sqrt(100) = 10. So, the rectangle at index 1 has a greater diagonal length therefore we return area = 8 *

6 = 48.

Example 2:

Input:

dimensions = [[3,4],[4,3]]

Output:

12

Explanation:

Length of diagonal is the same for both which is 5, so maximum area = 12.

Constraints:

1 <= dimensions.length <= 100

dimensions[i].length == 2

1 <= dimensions[i][0], dimensions[i][1] <= 100

Code Snippets

C++:

```
class Solution {
public:
int areaOfMaxDiagonal(vector<vector<int>>& dimensions) {

}
};
```

Java:

```
class Solution {
public int areaOfMaxDiagonal(int[][] dimensions) {
}
}
```

Python3:

```
class Solution:
def areaOfMaxDiagonal(self, dimensions: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function areaOfMaxDiagonal(dimensions: number[][]): number {

};
```

C#:

```
public class Solution {
public int AreaOfMaxDiagonal(int[][] dimensions) {

}
}
```

C:

```
int areaOfMaxDiagonal(int** dimensions, int dimensionsSize, int*
dimensionsColSize) {

}
```

Go:

```
func areaOfMaxDiagonal(dimensions [][]int) int {

}
```

Kotlin:

```
class Solution {
fun areaOfMaxDiagonal(dimensions: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func areaOfMaxDiagonal(_ dimensions: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn area_of_max_diagonal(dimensions: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} dimensions
# @return {Integer}
def area_of_max_diagonal(dimensions)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $dimensions
* @return Integer
*/
function areaOfMaxDiagonal($dimensions) {

}
}
```

Dart:

```
class Solution {
int areaOfMaxDiagonal(List<List<int>> dimensions) {

}
}
```

Scala:

```
object Solution {
def areaOfMaxDiagonal(dimensions: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec area_of_max_diagonal(dimensions :: [[integer]]) :: integer
def area_of_max_diagonal(dimensions) do

end
end
```

Erlang:

```
-spec area_of_max_diagonal(Dimensions :: [[integer()]]) -> integer().
area_of_max_diagonal(Dimensions) ->
.
```

Racket:

```
(define/contract (area-of-max-diagonal dimensions)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Area of Longest Diagonal Rectangle
* Difficulty: Easy
* 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 areaOfMaxDiagonal(vector<vector<int>>& dimensions) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Area of Longest Diagonal Rectangle
* Difficulty: Easy
* 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 areaOfMaxDiagonal(int[][] dimensions) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Area of Longest Diagonal Rectangle
Difficulty: Easy
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 areaOfMaxDiagonal(self, dimensions: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Area of Longest Diagonal Rectangle
* Difficulty: Easy
* 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[][]} dimensions
* @return {number}
*/
var areaOfMaxDiagonal = function(dimensions) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Area of Longest Diagonal Rectangle
* Difficulty: Easy
* 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 areaOfMaxDiagonal(dimensions: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Area of Longest Diagonal Rectangle
* Difficulty: Easy
* 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 AreaOfMaxDiagonal(int[][] dimensions) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Area of Longest Diagonal Rectangle
* Difficulty: Easy
* 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 areaOfMaxDiagonal(int** dimensions, int dimensionsSize, int*
dimensionsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Area of Longest Diagonal Rectangle
// Difficulty: Easy
// 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 areaOfMaxDiagonal(dimensions [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun areaOfMaxDiagonal(dimensions: Array<IntArray>): Int {
}
}
```

Swift Solution:

```
class Solution {
func areaOfMaxDiagonal(_ dimensions: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Area of Longest Diagonal Rectangle
// Difficulty: Easy
// 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 area_of_max_diagonal(dimensions: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} dimensions
# @return {Integer}
def area_of_max_diagonal(dimensions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $dimensions
* @return Integer
*/
function areaOfMaxDiagonal($dimensions) {

}
}
```

Dart Solution:

```
class Solution {
int areaOfMaxDiagonal(List<List<int>> dimensions) {

}
}
```

Scala Solution:

```
object Solution {
def areaOfMaxDiagonal(dimensions: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec area_of_max_diagonal(dimensions :: [[integer]]) :: integer
def area_of_max_diagonal(dimensions) do

end
end
```

Erlang Solution:

```
-spec area_of_max_diagonal(Dimensions :: [[integer()]]) -> integer().
area_of_max_diagonal(Dimensions) ->
.
```

Racket Solution:

```
(define/contract (area-of-max-diagonal dimensions)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```