Problem 2371: Minimize Maximum Value in a Grid

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

containing

distinct

positive integers.

You have to replace each integer in the matrix with a positive integer satisfying the following conditions:

The

relative

order of every two elements that are in the same row or column should stay the

same

after the replacements.

The

maximum

number in the matrix after the replacements should be as

small

as possible.

The relative order stays the same if for all pairs of elements in the original matrix such that

grid[r

1

][c

1

] > grid[r

2

][c

2

]

where either

r

1

== r

2

or

c

1

== c

2

, then it must be true that

grid[r

1

][c

1

] > grid[r

2

][c

2

]

after the replacements.

For example, if

grid = [[2, 4, 5], [7, 3, 9]]

then a good replacement could be either

grid = [[1, 2, 3], [2, 1, 4]]

or

grid = [[1, 2, 3], [3, 1, 4]]

.

Return

the

resulting

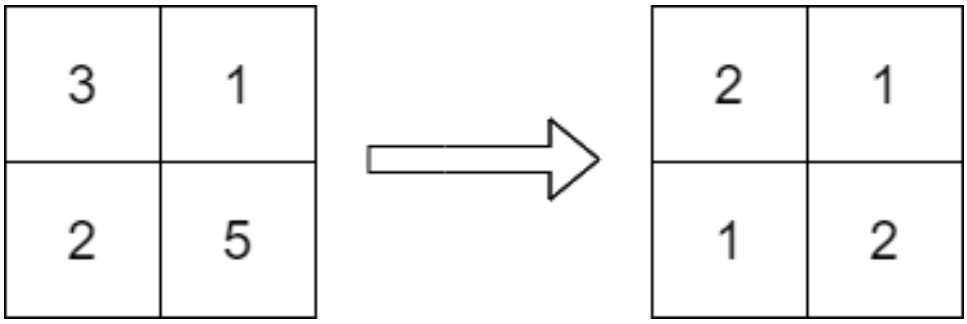
matrix.

If there are multiple answers, return

any

of them.

Example 1:



Input:

grid = [[3,1],[2,5]]

Output:

[[2,1],[1,2]]

Explanation:

The above diagram shows a valid replacement. The maximum number in the matrix is 2. It can be shown that no smaller value can be obtained.

Example 2:

Input:

grid = [[10]]

Output:

[[1]]

Explanation:

We replace the only number in the matrix with 1.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 1000

1 <= m * n <= 10

5

1 <= grid[i][j] <= 10

9

grid

consists of distinct integers.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number[][]}
*/
```

```
var minScore = function(grid) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** minScore(int** grid, int gridSize, int* gridColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

```
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-score grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Maximum Value in a Grid
```

```
* Difficulty: Hard
* Tags: array, graph, 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:
vector<vector<int>> minScore(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Maximum Value in a Grid
* Difficulty: Hard
* Tags: array, graph, 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[][] minScore(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Maximum Value in a Grid
Difficulty: Hard
Tags: array, graph, 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 minScore(self, grid: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize Maximum Value in a Grid
* Difficulty: Hard
* Tags: array, graph, 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[][]} grid
* @return {number[][]}
*/
var minScore = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Maximum Value in a Grid
* Difficulty: Hard
* Tags: array, graph, 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 minScore(grid: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Minimize Maximum Value in a Grid
* Difficulty: Hard
* Tags: array, graph, 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[][] MinScore(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Maximum Value in a Grid
* Difficulty: Hard
* Tags: array, graph, 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
```

```
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** minScore(int** grid, int gridSize, int* gridColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Minimize Maximum Value in a Grid
// Difficulty: Hard
// Tags: array, graph, 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 minScore(grid [][]int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
```

```
}
```

Rust Solution:

```
// Problem: Minimize Maximum Value in a Grid
// Difficulty: Hard
// Tags: array, graph, 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 min_score(grid: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-score grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```