Problem 1632: Rank Transform of a Matrix

Problem Information

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

Problem Description

Given an

m x n

matrix

, return

a new matrix

answer

where

answer[row][col]

is the

rank

of

matrix[row][col]

.

The

rank

is an

integer

that represents how large an element is compared to other elements. It is calculated using the following rules:

The rank is an integer starting from

1

.

If two elements

p

and

q

are in the

same row or column

, then:

If

$p < q$

then

$rank(p) < rank(q)$

If

p == q

then

rank(p) == rank(q)

If

p > q

then

rank(p) > rank(q)

The

rank

should be as

small

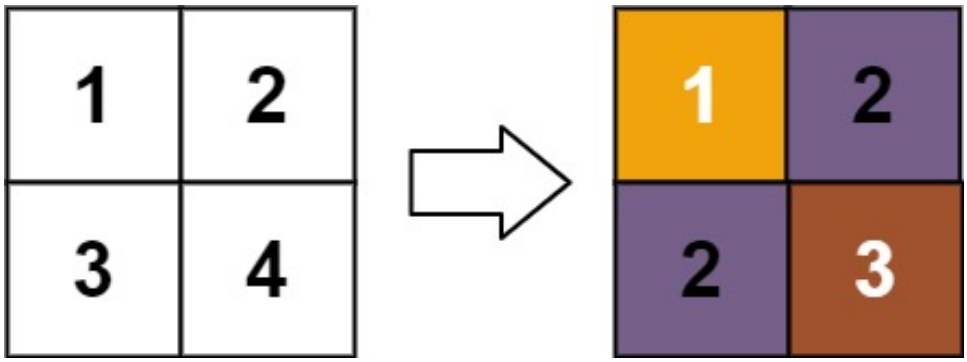
as possible.

The test cases are generated so that

answer

is unique under the given rules.

Example 1:



Input:

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

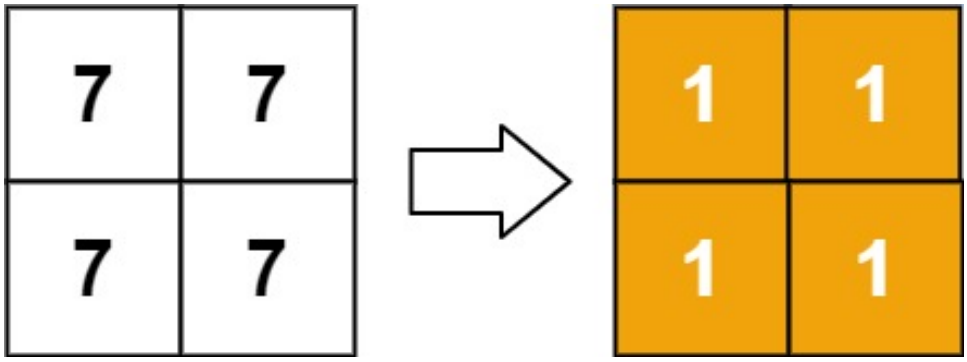
Output:

[[1,2],[2,3]]

Explanation:

The rank of matrix[0][0] is 1 because it is the smallest integer in its row and column. The rank of matrix[0][1] is 2 because matrix[0][1] > matrix[0][0] and matrix[0][0] is rank 1. The rank of matrix[1][0] is 2 because matrix[1][0] > matrix[0][0] and matrix[0][0] is rank 1. The rank of matrix[1][1] is 3 because matrix[1][1] > matrix[0][1], matrix[1][1] > matrix[1][0], and both matrix[0][1] and matrix[1][0] are rank 2.

Example 2:



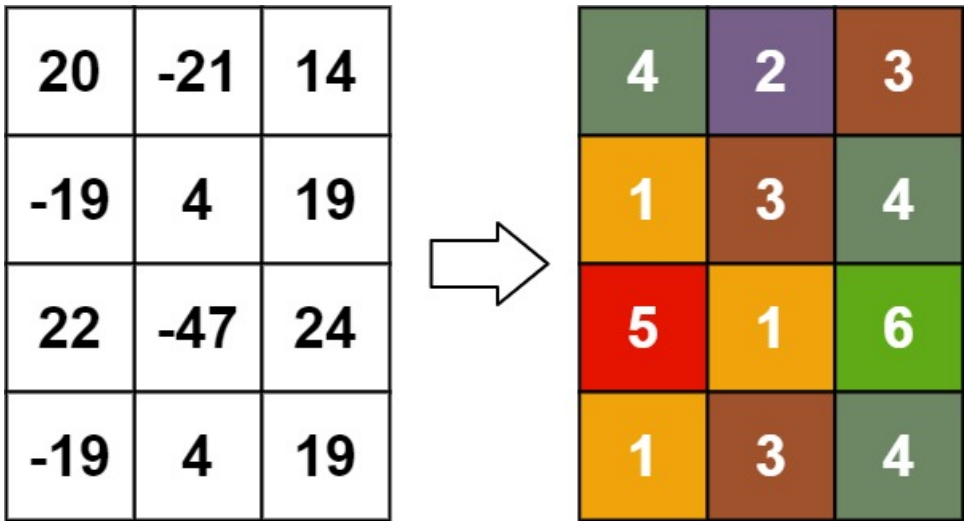
Input:

matrix = [[7,7],[7,7]]

Output:

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

Example 3:



Input:

matrix = [[20,-21,14],[-19,4,19],[22,-47,24],[-19,4,19]]

Output:

[[4,2,3],[1,3,4],[5,1,6],[1,3,4]]

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m, n <= 500

-10

9

<= matrix[row][col] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> matrixRankTransform(vector<vector<int>>& matrix) {

}
};
```

Java:

```
class Solution {
public int[][] matrixRankTransform(int[][] matrix) {

}
}
```

Python3:

```
class Solution:
def matrixRankTransform(self, matrix: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function matrixRankTransform(matrix: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] MatrixRankTransform(int[][] matrix) {

}
}
```

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** matrixRankTransform(int** matrix, int matrixSize, int* matrixColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func matrixRankTransform(matrix [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun matrixRankTransform(matrix: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func matrixRankTransform(_ matrix: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn matrix_rank_transform(matrix: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer[][]}
def matrix_rank_transform(matrix)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer[][]
*/
function matrixRankTransform($matrix) {

}
}
```

Dart:

```
class Solution {
List<List<int>> matrixRankTransform(List<List<int>> matrix) {

}
```

```
}
```

Scala:

```
object Solution {
def matrixRankTransform(matrix: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec matrix_rank_transform(matrix :: [[integer]]) :: [[integer]]
def matrix_rank_transform(matrix) do

end
end
```

Erlang:

```
-spec matrix_rank_transform(Matrix :: [[integer()]]) -> [[integer()]].
matrix_rank_transform(Matrix) ->
.
```

Racket:

```
(define/contract (matrix-rank-transform matrix)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rank Transform of a Matrix
* 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>> matrixRankTransform(vector<vector<int>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Rank Transform of a Matrix
* 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[][] matrixRankTransform(int[][] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Rank Transform of a Matrix
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 matrixRankTransform(self, matrix: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Rank Transform of a Matrix
* 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[][]} matrix
* @return {number[][]}
*/
var matrixRankTransform = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Rank Transform of a Matrix
* 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 matrixRankTransform(matrix: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Rank Transform of a Matrix
* 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[][] MatrixRankTransform(int[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Rank Transform of a Matrix
* 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** matrixRankTransform(int** matrix, int matrixSize, int* matrixColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Rank Transform of a Matrix
// 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 matrixRankTransform(matrix [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun matrixRankTransform(matrix: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func matrixRankTransform(_ matrix: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Rank Transform of a Matrix
// 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 matrix_rank_transform(matrix: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer[][]}
def matrix_rank_transform(matrix)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer[][]
*/
function matrixRankTransform($matrix) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> matrixRankTransform(List<List<int>> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def matrixRankTransform(matrix: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec matrix_rank_transform(matrix :: [[integer]]) :: [[integer]]
def matrix_rank_transform(matrix) do

end
end
```

Erlang Solution:

```
-spec matrix_rank_transform(Matrix :: [[integer()]]) -> [[integer()]].
matrix_rank_transform(Matrix) ->
.
```

Racket Solution:

```
(define/contract (matrix-rank-transform matrix)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```