

Problem 3033: Modify the Matrix

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a

0-indexed

$m \times n$

integer matrix

matrix

, create a new

0-indexed

matrix called

answer

. Make

answer

equal to

matrix

, then replace each element with the value

-1

with the

maximum

element in its respective column.

Return

the matrix

answer

.

Example 1:

1	2	-1
4	-1	6
7	8	9

→

1	2	9
4	8	6
7	8	9

Input:

```
matrix = [[1,2,-1],[4,-1,6],[7,8,9]]
```

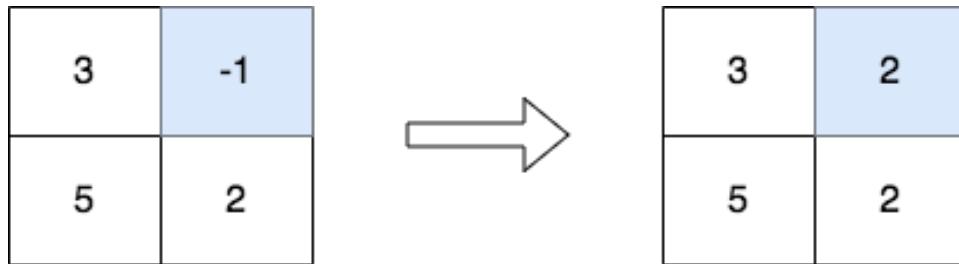
Output:

```
[[1,2,9],[4,8,6],[7,8,9]]
```

Explanation:

The diagram above shows the elements that are changed (in blue). - We replace the value in the cell [1][1] with the maximum value in the column 1, that is 8. - We replace the value in the cell [0][2] with the maximum value in the column 2, that is 9.

Example 2:



Input:

```
matrix = [[3,-1],[5,2]]
```

Output:

```
[[3,2],[5,2]]
```

Explanation:

The diagram above shows the elements that are changed (in blue).

Constraints:

```
m == matrix.length
```

```
n == matrix[i].length
```

```
2 <= m, n <= 50
```

```
-1 <= matrix[i][j] <= 100
```

The input is generated such that each column contains at least one non-negative integer.

Code Snippets

C++:

```
class Solution {  
public:  
vector<vector<int>> modifiedMatrix(vector<vector<int>>& matrix) {  
  
}  
};
```

Java:

```
class Solution {  
public int[][] modifiedMatrix(int[][] matrix) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function modifiedMatrix(matrix: number[][]): number[][] {  
};
```

C#:

```
public class Solution {  
    public int[][] ModifiedMatrix(int[][] matrix) {  
        return 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** modifiedMatrix(int** matrix, int matrixSize, int* matrixColSize, int*  
returnSize, int** returnColumnSizes) {  
  
}
```

Go:

```
func modifiedMatrix(matrix [[[int]]]) [[int]] {  
}
```

Kotlin:

```
class Solution {  
    fun modifiedMatrix(matrix: Array<IntArray>): Array<IntArray> {  
        return matrix  
    }  
}
```

Swift:

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

```
}
```

```
}
```

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }
}
```

Dart:

```
class Solution {
    List<List<int>> modifiedMatrix(List<List<int>> matrix) {
        }
    }
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*  
 * Problem: Modify the Matrix  
 * 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:  
vector<vector<int>> modifiedMatrix(vector<vector<int>>& matrix) {  
}  
};
```

Java Solution:

```
/**  
 * Problem: Modify the Matrix  
 * 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[][] modifiedMatrix(int[][] matrix) {  
}  
}
```

Python3 Solution:

```
"""  
Problem: Modify the Matrix  
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 modifiedMatrix(self, matrix: List[List[int]]) -> List[List[int]]:  
# TODO: Implement optimized solution  
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Modify the Matrix
 * 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[][]} matrix
 * @return {number[][]}
 */
var modifiedMatrix = function(matrix) {

};
```

TypeScript Solution:

```
/**
 * Problem: Modify the Matrix
 * 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 modifiedMatrix(matrix: number[][]): number[][] {
```

```
};
```

C# Solution:

```
/*
 * Problem: Modify the Matrix
 * 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[][] ModifiedMatrix(int[][] matrix) {
        return null;
    }
}
```

C Solution:

```
/*
 * Problem: Modify the Matrix
 * 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
 */

/**
 * 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** modifiedMatrix(int** matrix, int matrixSize, int* matrixColSize, int*
returnSize, int** returnColumnSizes) {
```

```
}
```

Go Solution:

```
// Problem: Modify the Matrix
// 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 modifiedMatrix(matrix [][]int) [][]int {
}
```

Kotlin Solution:

```
class Solution {
    fun modifiedMatrix(matrix: Array<IntArray>): Array<IntArray> {
        return matrix
    }
}
```

Swift Solution:

```
class Solution {
    func modifiedMatrix(_ matrix: [[Int]]) -> [[Int]] {
        return matrix
    }
}
```

Rust Solution:

```
// Problem: Modify the Matrix
// 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 modified_matrix(matrix: Vec<Vec<i32>>) -> Vec<Vec<i32>> {
        ...
    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

```
class Solution {
    List<List<int>> modifiedMatrix(List<List<int>> matrix) {
        ...
    }
}
```

Scala Solution:

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

```
}
```

```
}
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

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