Problem 2133: Check if Every Row and Column Contains All Numbers

Problem Information

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

Problem Description

An

n x n

matrix is

valid

if every row and every column contains

all

the integers from

1

to

n

(

inclusive

).

Given an

n x n

integer matrix

matrix

, return

true

if the matrix is

valid

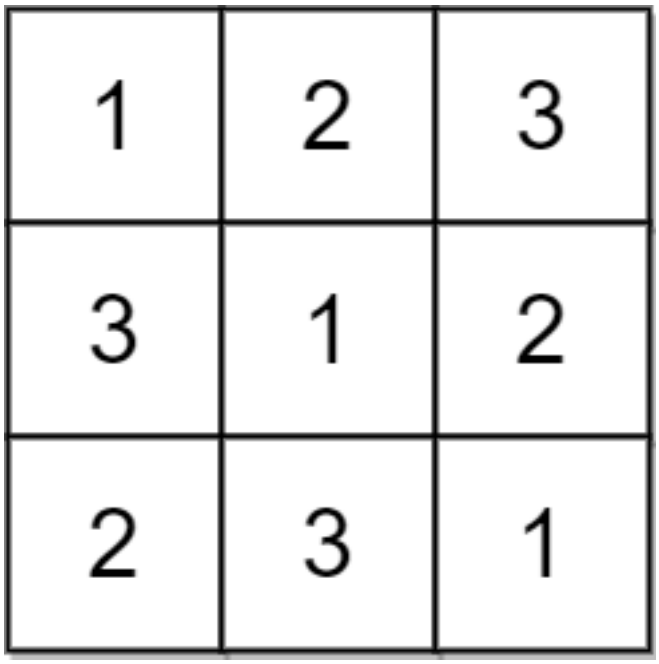
.

Otherwise, return

false

.

Example 1:



Input:

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

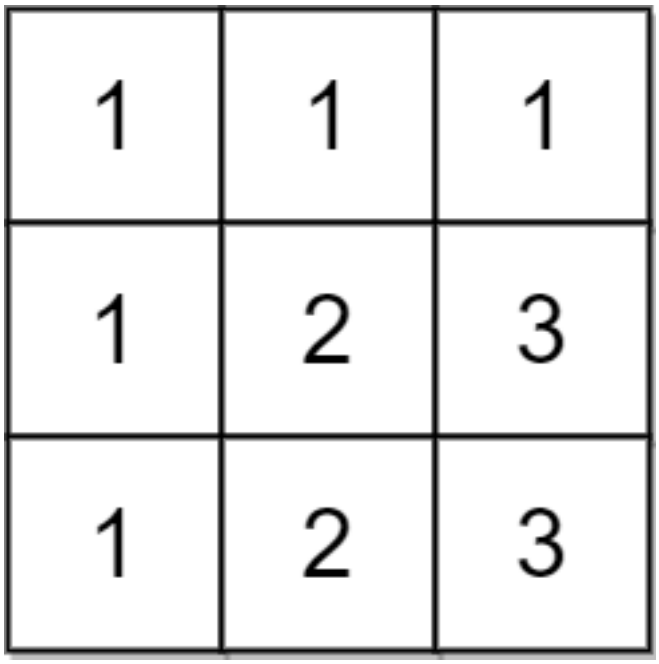
Output:

true

Explanation:

In this case, n = 3, and every row and column contains the numbers 1, 2, and 3. Hence, we return true.

Example 2:



Input:

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

Output:

false

Explanation:

In this case, n = 3, but the first row and the first column do not contain the numbers 2 or 3. Hence, we return false.

Constraints:

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

1 <= n <= 100

1 <= matrix[i][j] <= n

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean checkValid(int[][] matrix) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function checkValid(matrix: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CheckValid(int[][] matrix) {

}
}
```

C:

```
bool checkValid(int** matrix, int matrixSize, int* matrixColSize) {

}
```

Go:

```
func checkValid(matrix [][]int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool checkValid(List<List<int>> matrix) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-valid matrix)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if Every Row and Column Contains All Numbers
* Difficulty: Easy
* 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:
bool checkValid(vector<vector<int>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Check if Every Row and Column Contains All Numbers
* Difficulty: Easy
* 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 boolean checkValid(int[][] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Check if Every Row and Column Contains All Numbers
Difficulty: Easy
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 checkValid(self, matrix: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Check if Every Row and Column Contains All Numbers
* Difficulty: Easy
* 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[][]} matrix
* @return {boolean}
*/
var checkValid = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if Every Row and Column Contains All Numbers
* Difficulty: Easy
* 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 checkValid(matrix: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Every Row and Column Contains All Numbers
* Difficulty: Easy
* 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 bool CheckValid(int[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Check if Every Row and Column Contains All Numbers
* Difficulty: Easy
* 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
*/

bool checkValid(int** matrix, int matrixSize, int* matrixColSize) {

}
```

Go Solution:

```
// Problem: Check if Every Row and Column Contains All Numbers
// Difficulty: Easy
// 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 checkValid(matrix [][]int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if Every Row and Column Contains All Numbers
// Difficulty: Easy
// 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 check_valid(matrix: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool checkValid(List<List<int>> matrix) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-valid matrix)
(-> (listof (listof exact-integer?)) boolean?)
)
```