Problem 2128: Remove All Ones With Row and Column Flips

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

grid

.

In one operation, you can choose

any

row or column and flip each value in that row or column (i.e., changing all

0

's to

1

's, and all

1

's to

0

's).

Return

true

if it is possible to remove all

1

's from

grid

using

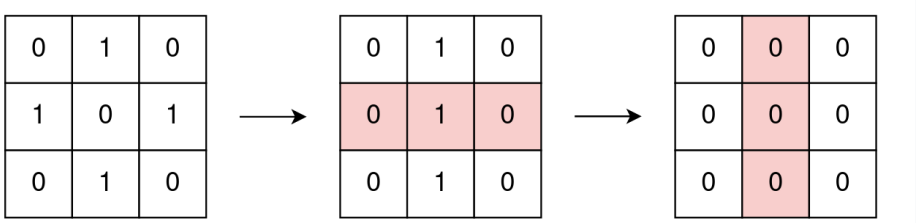
any

number of operations or

false

otherwise.

Example 1:



Input:

grid = [[0,1,0],[1,0,1],[0,1,0]]

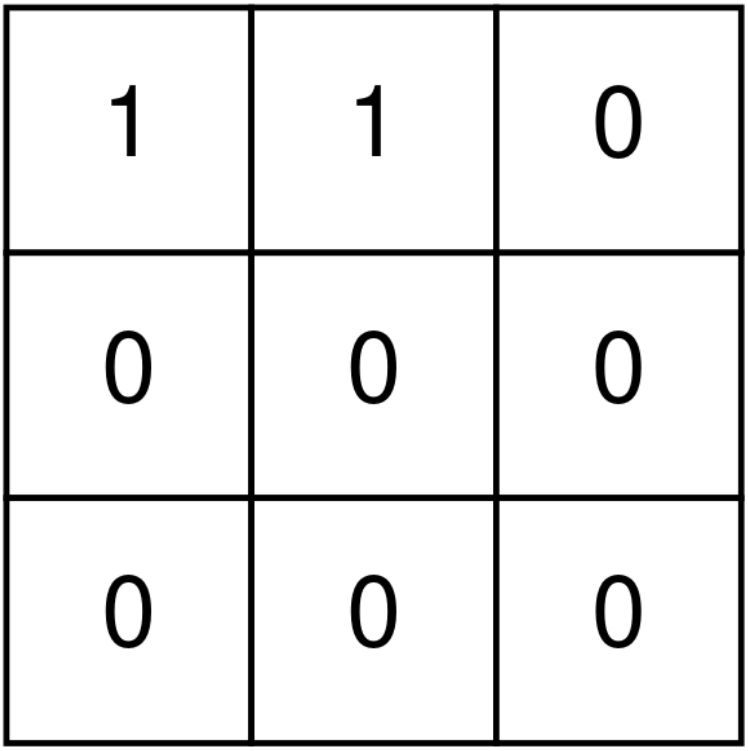
Output:

true

Explanation:

One possible way to remove all 1's from grid is to: - Flip the middle row - Flip the middle column

Example 2:



Input:

grid = [[1,1,0],[0,0,0],[0,0,0]]

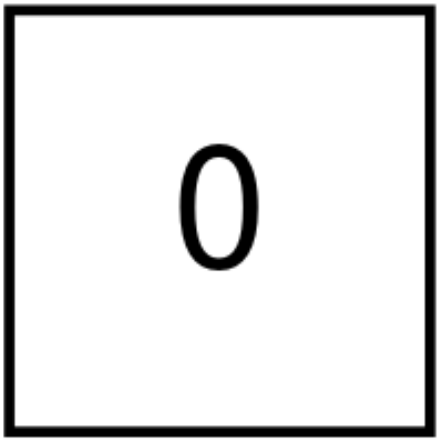
Output:

false

Explanation:

It is impossible to remove all 1's from grid.

Example 3:



Input:

grid = [[0]]

Output:

true

Explanation:

There are no 1's in grid.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 300

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean removeOnes(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function removeOnes(grid: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool RemoveOnes(int[][] grid) {

}
}
```

C:

```
bool removeOnes(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func removeOnes(grid [][]int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[][] $grid
* @return Boolean
*/
function removeOnes($grid) {

}
}
```

Dart:

```
class Solution {
bool removeOnes(List<List<int>> grid) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (remove-ones grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove All Ones With Row and Column Flips
* Difficulty: Medium
* Tags: array, math
*
* 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:
bool removeOnes(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Remove All Ones With Row and Column Flips
* Difficulty: Medium
* Tags: array, math
*
* 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 boolean removeOnes(int[][] grid) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Remove All Ones With Row and Column Flips
Difficulty: Medium
Tags: array, math

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 removeOnes(self, grid: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove All Ones With Row and Column Flips
* Difficulty: Medium
* Tags: array, math
*
* 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 {boolean}
*/
var removeOnes = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove All Ones With Row and Column Flips
* Difficulty: Medium
* Tags: array, math
*
* 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 removeOnes(grid: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Remove All Ones With Row and Column Flips
* Difficulty: Medium
* Tags: array, math
*
* 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 bool RemoveOnes(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Remove All Ones With Row and Column Flips
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

bool removeOnes(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Remove All Ones With Row and Column Flips
// Difficulty: Medium
// Tags: array, math
//
// 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 removeOnes(grid [][]int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Remove All Ones With Row and Column Flips
// Difficulty: Medium
// Tags: array, math
//
// 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 remove_ones(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool removeOnes(List<List<int>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (remove-ones grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```