Problem 3142: Check if Grid Satisfies Conditions

Problem Information

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

Problem Description

You are given a 2D matrix

grid

of size

m x n

. You need to check if each cell

grid[i][j]

is:

Equal to the cell below it, i.e.

grid[i][j] == grid[i + 1][j]

(if it exists).

Different from the cell to its right, i.e.

grid[i][j] != grid[i][j + 1]

(if it exists).

Return

true

if

all

the cells satisfy these conditions, otherwise, return

false

.

Example 1:

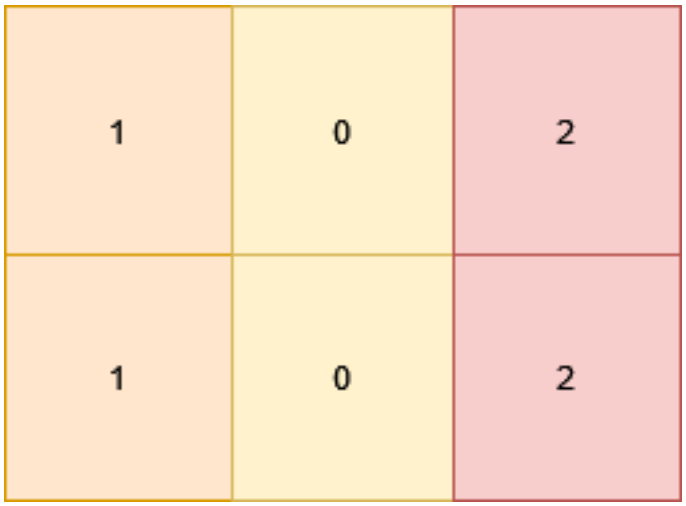
Input:

grid = [[1,0,2],[1,0,2]]

Output:

true

Explanation:



All the cells in the grid satisfy the conditions.

Example 2:

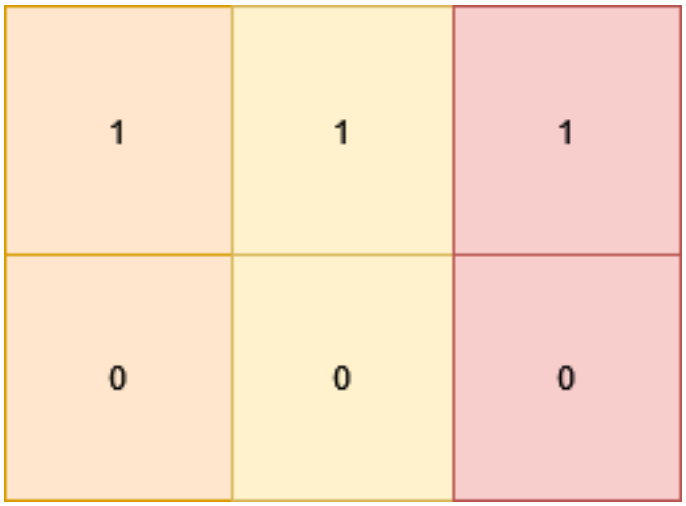
Input:

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

Output:

false

Explanation:



All cells in the first row are equal.

Example 3:

Input:

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

Output:

false

Explanation:

Cells in the first column have different values.

Constraints:

1 <= n, m <= 10

0 <= grid[i][j] <= 9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (satisfies-conditions grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if Grid Satisfies Conditions
* 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:
bool satisfiesConditions(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Check if Grid Satisfies Conditions
* 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 boolean satisfiesConditions(int[][] grid) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Check if Grid Satisfies Conditions
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 satisfiesConditions(self, grid: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if Grid Satisfies Conditions
* 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[][]} grid
* @return {boolean}
*/
var satisfiesConditions = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if Grid Satisfies Conditions
* 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 satisfiesConditions(grid: number[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if Grid Satisfies Conditions
* 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 bool SatisfiesConditions(int[][] grid) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Check if Grid Satisfies Conditions
* 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
*/

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

}
```

Go Solution:

```
// Problem: Check if Grid Satisfies Conditions
// 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 satisfiesConditions(grid [][]int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check if Grid Satisfies Conditions
// 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 satisfies_conditions(grid: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (satisfies-conditions grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```