Problem 3122: Minimum Number of Operations to Satisfy Conditions

Problem Information

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

Problem Description

You are given a 2D matrix

grid

of size

m x n

. In one

operation

, you can change the value of

any

cell to

any

non-negative number. You need to perform some

operations

such that 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 the

minimum

number of operations needed.

Example 1:

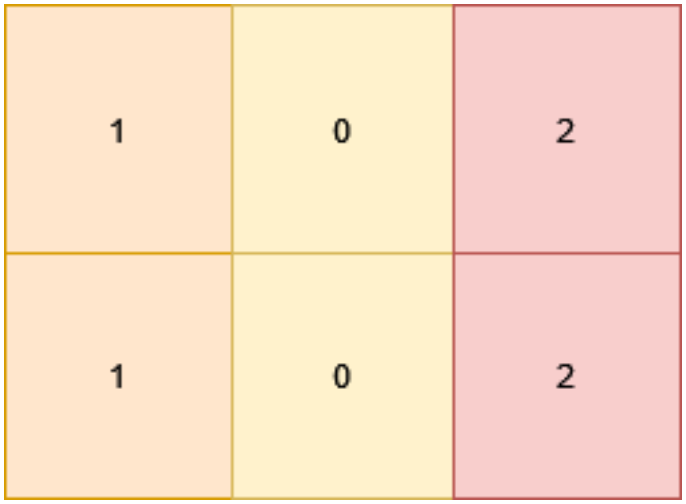
Input:

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

Output:

0

Explanation:



All the cells in the matrix already satisfy the properties.

Example 2:

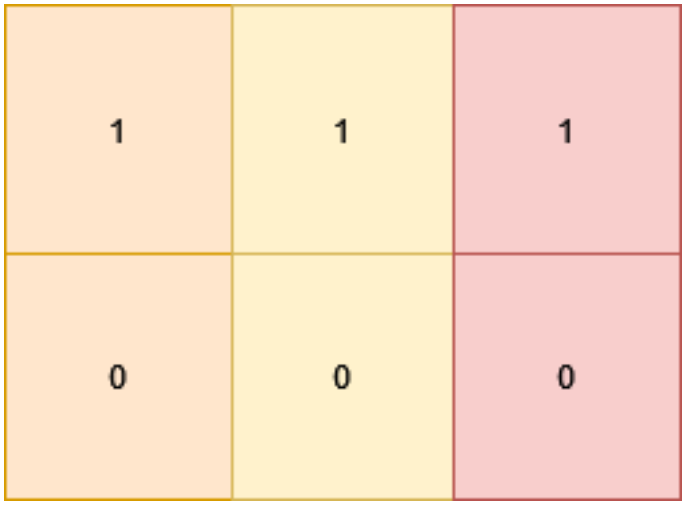
Input:

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

Output:

3

Explanation:



The matrix becomes

[[1,0,1],[1,0,1]]

which satisfies the properties, by doing these 3 operations:

Change

grid[1][0]

to 1.

Change

grid[0][1]

to 0.

Change

grid[1][2]

to 1.

Example 3:

Input:

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

Output:

2

Explanation:

There is a single column. We can change the value to 1 in each cell using 2 operations.

Constraints:

1 <= n, m <= 1000

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

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minimumOperations(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumOperations(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumOperations(int[][] grid) {

}
}
```

C:

```
int minimumOperations(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func minimumOperations(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumOperations(grid: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minimumOperations(List<List<int>> grid) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-operations grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Satisfy Conditions
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimumOperations(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Satisfy Conditions
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumOperations(int[][] grid) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Satisfy Conditions
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumOperations(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Satisfy Conditions
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Satisfy Conditions
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumOperations(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Satisfy Conditions
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MinimumOperations(int[][] grid) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Satisfy Conditions
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumOperations(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Satisfy Conditions
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumOperations(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumOperations(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Satisfy Conditions
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_operations(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minimumOperations(List<List<int>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-operations grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```