Problem 3240: Minimum Number of Flips to Make Binary Grid Palindromic II

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

grid

.

A row or column is considered

palindromic

if its values read the same forward and backward.

You can

flip

any number of cells in

grid

from

0

to

1

, or from

1

to

0

.

Return the

minimum

number of cells that need to be flipped to make

all

rows and columns

palindromic

, and the total number of

1

's in

grid

divisible

by

4

.

Example 1:

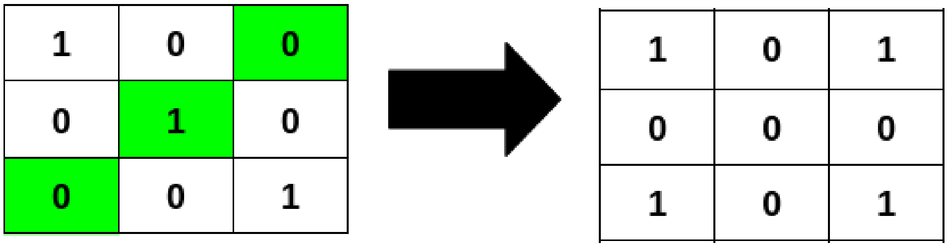
Input:

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

Output:

3

Explanation:



1	0	0
0	1	0
0	0	1

1	0	1
0	0	0
1	0	1

Example 2:

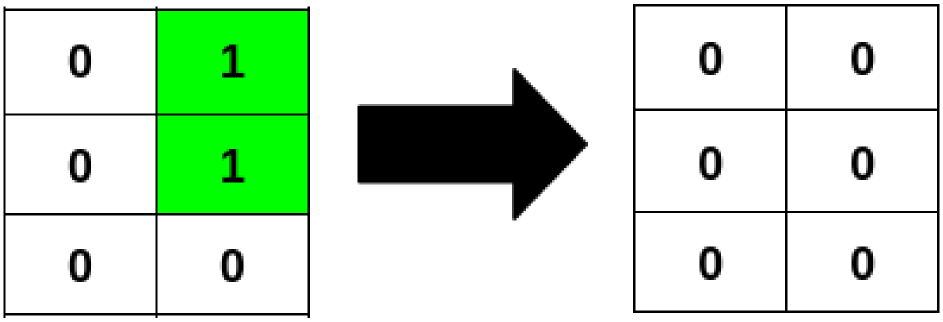
Input:

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

Output:

2

Explanation:



0	1
0	1
0	0

0	0
0	0
0	0

Example 3:

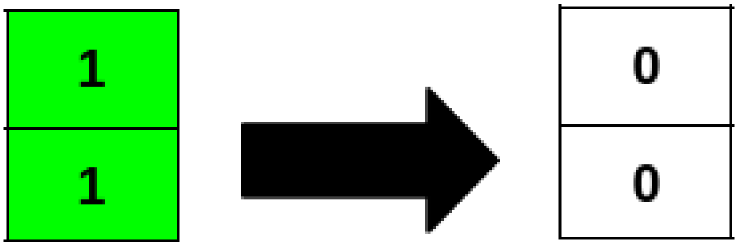
Input:

grid = [[1],[1]]

Output:

2

Explanation:



1
1

0
0

Constraints:

m == grid.length

n == grid[i].length

1 <= m * n <= 2 * 10

5

0 <= grid[i][j] <= 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-flips grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
* Difficulty: Medium
* 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 minFlips(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
* Difficulty: Medium
* 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 minFlips(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
Difficulty: Medium
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 minFlips(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
* Difficulty: Medium
* 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 {number}
*/
var minFlips = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
* Difficulty: Medium
* 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 minFlips(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
* Difficulty: Medium
* 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 MinFlips(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
* Difficulty: Medium
* 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
*/

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

}
```

Go Solution:

```
// Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
// Difficulty: Medium
// 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 minFlips(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Flips to Make Binary Grid Palindromic II
// Difficulty: Medium
// 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 min_flips(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-flips grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```