Problem 1284: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix

Problem Information

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

Problem Description

Given a

m x n

binary matrix

mat

. In one step, you can choose one cell and flip it and all the four neighbors of it if they exist (Flip is changing

1

to

0

and

0

to

1

). A pair of cells are called neighbors if they share one edge.

Return the

minimum number of steps

required to convert

mat

to a zero matrix or

-1

if you cannot.

A

binary matrix

is a matrix with all cells equal to

0

or

1

only.

A

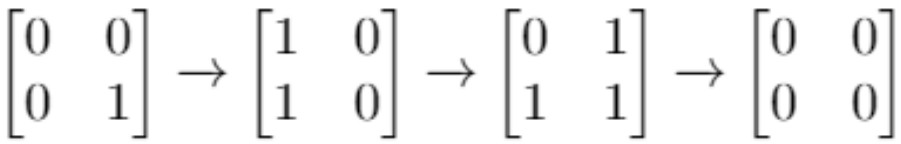
zero matrix

is a matrix with all cells equal to

0

.

Example 1:



Input:

mat = [[0,0],[0,1]]

Output:

3

Explanation:

One possible solution is to flip (1, 0) then (0, 1) and finally (1, 1) as shown.

Example 2:

Input:

mat = [[0]]

Output:

0

Explanation:

Given matrix is a zero matrix. We do not need to change it.

Example 3:

Input:

mat = [[1,0,0],[1,0,0]]

Output:

-1

Explanation:

Given matrix cannot be a zero matrix.

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n <= 3

mat[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
int minFlips(int** mat, int matSize, int* matColSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
* Difficulty: Hard
* Tags: array, hash, search
*
* 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:
int minFlips(vector<vector<int>>& mat) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
* Difficulty: Hard
* Tags: array, hash, search
*
* 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 int minFlips(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
Difficulty: Hard
Tags: array, hash, search

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 minFlips(self, mat: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
* Difficulty: Hard
```

```
* Tags: array, hash, search
*
* 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[][]} mat
* @return {number}
*/
var minFlips = function(mat) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
* Difficulty: Hard
* Tags: array, hash, search
*
* 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 minFlips(mat: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
* Difficulty: Hard
* Tags: array, hash, search
*
* 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 int MinFlips(int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minFlips(int** mat, int matSize, int* matColSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
// Difficulty: Hard
// Tags: array, hash, search
//
// 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 minFlips(mat [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Flips to Convert Binary Matrix to Zero Matrix
// Difficulty: Hard
// Tags: array, hash, search
//
// 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 min_flips(mat: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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