Problem 1072: Flip Columns For Maximum Number of Equal Rows

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

matrix

.

You can choose any number of columns in the matrix and flip every cell in that column (i.e., Change the value of the cell from

0

to

1

or vice versa).

Return

the maximum number of rows that have all values equal after some number of flips

.

Example 1:

Input:

matrix = [[0,1],[1,1]]

Output:

1

Explanation:

After flipping no values, 1 row has all values equal.

Example 2:

Input:

matrix = [[0,1],[1,0]]

Output:

2

Explanation:

After flipping values in the first column, both rows have equal values.

Example 3:

Input:

matrix = [[0,0,0],[0,0,1],[1,1,0]]

Output:

2

Explanation:

After flipping values in the first two columns, the last two rows have equal values.

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m, n <= 300

matrix[i][j]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int maxEqualRowsAfterFlips(vector<vector<int>>& matrix) {

}
};
```

Java:

```
class Solution {
public int maxEqualRowsAfterFlips(int[][] matrix) {
```

```
}
}
```

Python3:

```
class Solution:
def maxEqualRowsAfterFlips(self, matrix: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxEqualRowsAfterFlips(matrix: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxEqualRowsAfterFlips(int[][] matrix) {

}
}
```

C:

```
int maxEqualRowsAfterFlips(int** matrix, int matrixSize, int* matrixColSize)
{

}
```

Go:

```
func maxEqualRowsAfterFlips(matrix [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxEqualRowsAfterFlips(matrix: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxEqualRowsAfterFlips(_ matrix: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_equal_rows_after_flips(matrix: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer}
def max_equal_rows_after_flips(matrix)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer
*/
function maxEqualRowsAfterFlips($matrix) {

}
}
```

Dart:

```
class Solution {
int maxEqualRowsAfterFlips(List<List<int>> matrix) {

}
}
```

Scala:

```
object Solution {
def maxEqualRowsAfterFlips(matrix: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_equal_rows_after_flips(matrix :: [[integer]]) :: integer
def max_equal_rows_after_flips(matrix) do

end
end
```

Erlang:

```
-spec max_equal_rows_after_flips(Matrix :: [[integer()]]) -> integer().
max_equal_rows_after_flips(Matrix) ->
.
```

Racket:

```
(define/contract (max-equal-rows-after-flips matrix)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Flip Columns For Maximum Number of Equal Rows
* Difficulty: Medium
* Tags: array, hash
*
* 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 maxEqualRowsAfterFlips(vector<vector<int>>& matrix) {

}
};
```

Java Solution:

```
/**
* Problem: Flip Columns For Maximum Number of Equal Rows
* Difficulty: Medium
* Tags: array, hash
*
* 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 maxEqualRowsAfterFlips(int[][] matrix) {

}
}
```

Python3 Solution:

```
"""
Problem: Flip Columns For Maximum Number of Equal Rows
Difficulty: Medium
Tags: array, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Flip Columns For Maximum Number of Equal Rows
* Difficulty: Medium
* Tags: array, hash
*
```

```
* 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[][]} matrix
* @return {number}
*/
var maxEqualRowsAfterFlips = function(matrix) {

};
```

TypeScript Solution:

```
/**
* Problem: Flip Columns For Maximum Number of Equal Rows
* Difficulty: Medium
* Tags: array, hash
*
* 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 maxEqualRowsAfterFlips(matrix: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Flip Columns For Maximum Number of Equal Rows
* Difficulty: Medium
* Tags: array, hash
*
* 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 MaxEqualRowsAfterFlips(int[][] matrix) {

}
}
```

C Solution:

```
/*
* Problem: Flip Columns For Maximum Number of Equal Rows
* Difficulty: Medium
* Tags: array, hash
*
* 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 maxEqualRowsAfterFlips(int** matrix, int matrixSize, int* matrixColSize)
{

}
```

Go Solution:

```
// Problem: Flip Columns For Maximum Number of Equal Rows
// Difficulty: Medium
// Tags: array, hash
//
// 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 maxEqualRowsAfterFlips(matrix [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxEqualRowsAfterFlips(matrix: Array<IntArray>): Int {
```

```
}
}
```

Swift Solution:

```
class Solution {
func maxEqualRowsAfterFlips(_ matrix: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Flip Columns For Maximum Number of Equal Rows
// Difficulty: Medium
// Tags: array, hash
//
// 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 max_equal_rows_after_flips(matrix: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer}
def max_equal_rows_after_flips(matrix)

end
```

PHP Solution:

```
class Solution {

/**
```

```
* @param Integer[][] $matrix
* @return Integer
*/
function maxEqualRowsAfterFlips($matrix) {

}
}
```

Dart Solution:

```
class Solution {
int maxEqualRowsAfterFlips(List<List<int>> matrix) {

}
}
```

Scala Solution:

```
object Solution {
def maxEqualRowsAfterFlips(matrix: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_equal_rows_after_flips(matrix :: [[integer]]) :: integer
def max_equal_rows_after_flips(matrix) do

end
end
```

Erlang Solution:

```
-spec max_equal_rows_after_flips(Matrix :: [[integer()]]) -> integer().
max_equal_rows_after_flips(Matrix) ->
.
```

Racket Solution:

```
(define/contract (max-equal-rows-after-flips matrix)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```