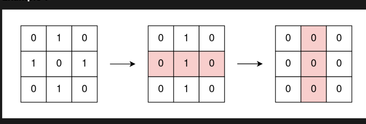
2128. Remove All Ones With Row and Column Flips

You are given an `m x n` binary matrix `grid`.

In one operation, you can choose **any** row or column and flip each value in that row or colum (i.e., changing all 0's to 1's, and all 1's to 0's).

Return `true` if it is possible to remove all `1`'s from `grid` using **any** number of operations or `false` otherwise.

Example 1:



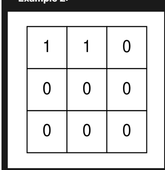
Input: `grid = [[0,1,0],[1,0,1],[0,1,0]]`

Output: `true`

Explanation: One possible way to remove all 1's from `grid` is to:

- Flip the middle row
- Flip the middle column

Example 2:



Input: `grid = [[1,1,0],[0,0,0],[0,0,0]]`

Output: `false`

Explanation: It is impossible to remove all 1's from `grid`.

Input: `grid = [[0]]`

Output: `true`

Explanation: There are no 1's in `grid`.

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 300`
- `grid[i][j]` is either 0 or 1.

Solution

The first observation we can make is that for every row/column, we will either flip that row/column once or zero times. This is because if we flip a row/column twice, it will have no effect on the grid.

With this in mind, for each row and column, we can either choose to flip it or not flip it. If we brute force every combination, we will take $O(MN)$ to simulate each combination. Since there are $O(2^{M+N})$ combinations, this gives us a time complexity of $O(2^{M+N} \times MN)$. Let's look for a faster solution as this is way too slow.

Let's call `grid` reachable if it's possible to apply some row/column flips to remove all 0's.

We can observe that if each row/column is flipped at most once, there is only one set of rows and columns that can be flipped to set all cells to 0 (assuming a solution exists). Let's call these necessary rows and columns.

To check whether or not `grid` is reachable, we'll first flip all necessary columns, and then the necessary rows.

Let's first assume that we already flipped all necessary columns on `grid`. We can observe that `grid` is now reachable by only row flips if all rows have the same integer (i.e. all 0's or all 1's).

Example

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	1	1	1	1	1	1

Here, we have a `grid` reachable by only row flips as all rows have the same integer. We can remove all 1's from `grid` by flipping rows with all 1's. Here, we will flip rows 1 and 5.

Our goal is to figure out a way to flip some set of columns such that `grid` will only contain rows with all 0's or all 1's.

With that in mind, our algorithm will go as follows. First, find all cells in the first row of `grid` with a 1. For every cell `(0,j)` in the first row that's a 1, let's flip the j^{th} column so that the cell `(0,j)` becomes a 0. This is important as it causes the first row to have all 0's. At this point there are two cases to consider.

- The first case is that the rest of the rows have all 0's or all 1's. In this case, we can conclude that `grid` is reachable and we'll return `true`.
- The second case is that there exists a row that has a mix of 0's and 1's, which we don't want. In this case, we'll return `false` since we can observe that it's impossible to get all rows to have all 0's or all 1's. This is because if we try to change the other rows to have all 0's or 1's with more column flips, the content of the first row will change and it will no longer have all 0's.

Time Complexity

In the worst scenario, we will perform $O(M)$ row flips and $O(N)$ column flips. Since row flips take $O(N)$ to simulate and column flips take $O(M)$ to simulate, our total time complexity is $O(NM)$.

Time Complexity: $O(NM)$.

Space Complexity

Since we don't actually use additional arrays or lists, our space complexity is $O(1)$.

Space Complexity: $O(1)$.

```
class Solution {
  public:
    bool removeOnes(vector<vector<int>>& grid) {
        int m = grid.size();  // get dimensions of grid
        int n = grid[0].size();
        for (int i = 0; i < n; i++) {  // flip columns so that first row only has 0's
            if (grid[0][i] == 1) {
                for (int j = 0; j < m; j++) {  // flips a column
                    grid[j][i] = 1 - grid[j][i];
                }
            }
        }
        bool ans = true;
        for (int i = 0; i < m; i++) {  // checks if each row has all 0's or all 1's
            int sum = 0;
            for (int j = 0; j < n; j++) {
                sum += grid[i][j];
            }
            if (sum == 0 || sum == n) {
                continue;
            }
            ans = false;
        }
        return ans;
    }
};
```

```
class Solution {
    public boolean removeOnes(int[][] grid) {
        int m = grid.length; // get dimensions of grid
        int n = grid[0].length;
        for (int i = 0; i < n; i++) { // flip columns so that first row only has 0's
            if (grid[0][i] == 1) {
                for (int j = 0; j < m; j++) { // flips a column
                    grid[j][i] = 1 - grid[j][i];
                }
            }
        }
        boolean ans = true;
        for (int i = 0; i < m; i++) { // checks if each row has all 0's or all 1's
            int sum = 0;
            for (int j = 0; j < n; j++) {
                sum += grid[i][j];
            }
            if (sum == 0 || sum == n) {
                continue;
            }
            ans = false;
        }
        return ans;
    }
}
```

```
class Solution:
    def removeOnes(self, grid: List[List[int]]) -> bool:
        m = len(grid)  # get dimensions of grid
        n = len(grid[0])
        for i in range(n):  # flip columns so that first row only has 0's
            if grid[0][i] == 1:
                for j in range(m):  # flips a column
                    grid[j][i] = 1 - grid[j][i]
        ans = True
        for i in range(m):  # checks if each row has all 0's or all 1's
            sum = 0
            for j in range(n):
                sum += grid[i][j]
            if sum == 0 or sum == n:
                continue
            ans = False
        return ans
```