数值同化题目描述：

存在一个 m*n 的二维数组，其成员取值范围为 0,1,2。其中值为 1 的元素具备同化特性，每经过 1S，将上下左右值为 0 的元素同化为 1。而值为 2 的元素，免疫同化。将数组所有成员随机初始化为 0 或 2，再将矩阵的[0,0]元素修改成 1，在经过足够长的时间后，求矩阵中有多少个元素是 0 或 2（即 0 和 2 数量之和）。

输入描述：

输入的前两个数字是矩阵大小。后面的数字是矩阵内容。

输出描述：

返回矩阵中非 1 的元素个数

补充说明：

m 和 n 不会超过 30(含 30)。

示例 1

输入：

```
4 4
0 0 0 0
0 2 2 2
0 2 0 0
0 2 0 0
```

输出：

```
9
```

说明：

输入数字前两个数字是矩阵大小。后面的是数字是矩阵内容。
这个矩阵的内容如下：

{

0,0,0,0

0,2,2,2

0,2,0,0

```
 0,2,0,0
}
```

起始位置 (0,0)被修改为 1 后，最终只能同化矩阵为：

```
{
 1,1,1,1
 1,2,2,2
 1,2,0,0
 1,2,0,0
}
```

所以矩阵中非 1 的元素个数为 9。

```
#include <stdarg.h>
#include <iostream>
#include <vector>
#include <limits>
#include <unordered_map>
#include <unordered_set>
#include <algorithm>
#include <queue>
#include <array>
#include <numeric>
using namespace std;
```

```
struct point
{
    int x,y;
};


int main() {

    int m, n;
    cin >> m >> n;

    vector<vector<int>> grid(m, vector<int>(n));
    for (int r = 0; r < m; r++){
        for (int c = 0; c < n; c++)
            cin >> grid[r][c];
    }
    grid[0][0] = 1;


    queue<point> q;
    q.push({0, 0});
    int one_cnt = 0;
```

```
    vector<vector<bool>> is_visited(m, vector<bool>(n, false));
    is_visited[0][0] = true;

    while (q.empty() == false)
    {
        auto [r, c] = q.front();
        q.pop();
        one_cnt++;

        if (r + 1 < grid.size() && grid[r+1][c] == 0 && is_visited[r + 1][c] ==
false){
            is_visited[r + 1][c] = true;
            q.push({r+1, c});
            grid[r + 1][c] = 1;
        }
        if (r - 1 >= 0 && grid[r-1][c] == 0 && is_visited[r - 1][c] == false){
            is_visited[r-1][c] = true;
            q.push({r - 1, c});
            grid[r - 1][c] = 1;
        }
        if (c + 1 < grid[r].size() && grid[r][c + 1] == 0 && is_visited[r][c+1] ==
```

```
false){
                is_visited[r][c + 1] = true;
                q.push({r, c+1});
                grid[r][c+1] = 1;
            }
            if (c - 1 >= 0 && grid[r][c - 1] == 0 && is_visited[r][c - 1] == false){
                is_visited[r][c-1] = true;
                q.push({r,c - 1});
                grid[r][c-1] = 1;
            }
        }

        int ans = 0;
        for (int r = 0; r < m; r++){
            for (int c = 0; c < n; c++)
                ans += grid[r][c] != 1;
        }
        cout << ans << endl;
        // cout << ((m * n) - one_cnt) << endl;

}
```