1252. 奇数值单元格的数目（简单）

1. 题目描述

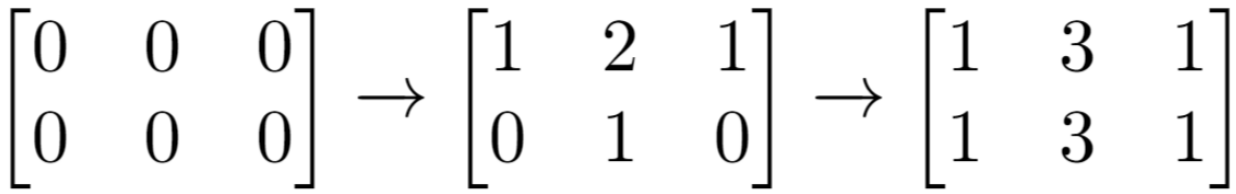
给你一个 `n` 行 `m` 列的矩阵，最开始的时候，每个单元格中的值都是 `0`。

另有一个索引数组 `indices`，`indices[i] = [ri, ci]` 中的 `ri` 和 `ci` 分别表示指定的行和列（从 `0` 开始编号）。

你需要将每对 `[ri, ci]` 指定的行和列上的所有单元格的值加 `1`。

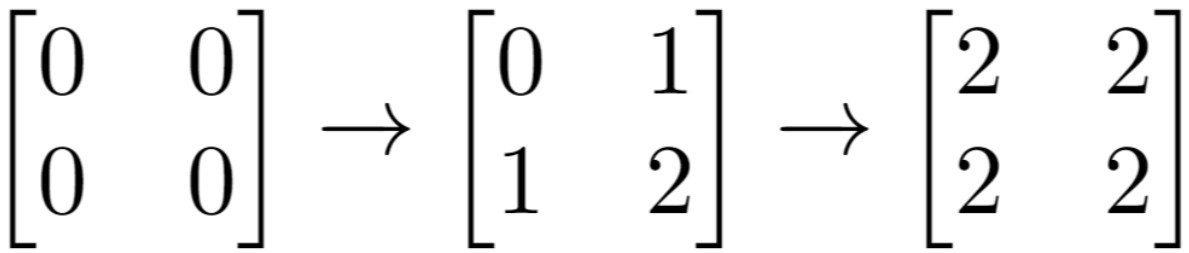
请你在执行完所有 `indices` 指定的增量操作后，返回矩阵中 「奇数值单元格」 的数目。

示例 1：



```
输入：n = 2, m = 3, indices = [[0,1],[1,1]]
输出：6
解释：最开始的矩阵是 [[0,0,0],[0,0,0]]。
第一次增量操作后得到 [[1,2,1],[0,1,0]]。
最后的矩阵是 [[1,3,1],[1,3,1]]，里面有 6 个奇数。
```

示例 2：



```
输入：n = 2, m = 2, indices = [[1,1],[0,0]]
输出：0
解释：最后的矩阵是 [[2,2],[2,2]]，里面没有奇数。
```

提示：

- `1 <= n <= 50`
- `1 <= m <= 50`
- `1 <= indices.length <= 100`
- `0 <= indices[i][0] < n`
- `0 <= indices[i][1] < m`

2. 简单实现

```
class Solution {
public:
    int oddCells(int n, int m, vector<vector<int>>& indices) {
        unordered_map<int, int> row, col;//记录每一行、每一列被加的次数
        for(int i = 0; i < indices.size(); i++){//统计每一行、每一列被加的次数
            if(row.count(indices[i][0]) <= 0)
                row[indices[i][0]] = 1;
            else
                row[indices[i][0]]++;
            if(col.count(indices[i][1]) <= 0)
                col[indices[i][1]] = 1;
            else
                col[indices[i][1]]++;
        }
        int ans = 0;
        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){//矩阵的索引[i,j]对应的数值为row[i]+col[j]
                int cur = 0;
                if(row.count(i) > 0)
                    cur += row[i];
                if(col.count(j) > 0)
                    cur += col[j];
                if(cur % 2 == 1)
                    ans++;
            }
        }
        return ans;
    }
};
```

1253. 重构 2 行二进制矩阵（中等）

1. 题目描述

给你一个 `2` 行 `n` 列的二进制数组：

- 矩阵是一个二进制矩阵，这意味着矩阵中的每个元素不是 `0` 就是 `1`。
- 第 `0` 行的元素之和为 `upper`。
- 第 `1` 行的元素之和为 `lower`。
- 第 `i` 列（从 `0` 开始编号）的元素之和为 `colsum[i]`，`colsum` 是一个长度为 `n` 的整数数组。

你需要利用 `upper`，`lower` 和 `colsum` 来重构这个矩阵，并以二维整数数组的形式返回它。

如果有多个不同的答案，那么任意一个都可以通过本题。

如果不存在符合要求的答案，就请返回一个空的二维数组。

示例 1：

```
输入: upper = 2, lower = 1, colsum = [1,1,1]
输出: [[1,1,0],[0,0,1]]
解释: [[1,0,1],[0,1,0]] 和 [[0,1,1],[1,0,0]] 也是正确答案。
```

示例 2:

```
输入: upper = 2, lower = 3, colsum = [2,2,1,1]
输出: []
```

示例 3:

```
输入: upper = 5, lower = 5, colsum = [2,1,2,0,1,0,1,2,0,1]
输出: [[1,1,1,0,1,0,0,1,0,0],[1,0,1,0,0,0,1,1,0,1]]
```

提示:

- `1 <= colsum.length <= 10^5`
- `0 <= upper, lower <= colsum.length`
- `0 <= colsum[i] <= 2`

2. 简单实现

- 把upper, lower看作当前两行各自剩余需要填充的1的数量
- 对于所有为2的colsum[i]，必须将对应第i列的两行都填充2，剩余的1可以选择任意一行填充1即可
- 但由于要保证colsum[i]=2的填充，因此应优先将剩余1的数量多的行填充为1

```
class Solution {
public:
    vector<vector<int>> reconstructMatrix(int upper, int lower, vector<int>&
colsum) {
        int n = colsum.size();
        vector<vector<int>> ans = vector<vector<int>>(2, vector<int>(n, 0));
        for(int i = 0; i < n; i++){
            if(colsum[i] == 2){
                ans[0][i] = 1;
                ans[1][i] = 1;
                upper--;
                lower--;
                if(upper < 0 || lower < 0)//某一行1的数量不够用
                    return vector<vector<int>>();
            }
            else if(colsum[i] == 1){
                if(upper <= 0 && lower <= 0)//此情况对应于upper+lower <
sum(colsum[i])
                    return vector<vector<int>>();
                if(upper >= lower){//优先填充剩余1多的
                    ans[0][i] = 1;
                    upper--;
                }
                else{
                    ans[1][i] = 1;
                    lower--;
```

```
                }
            }
        }
        if(upper > 0 || lower > 0)//此情况对应于upper+lower > sum(colsum[i])
            return vector<vector<int>>();
        else
            return ans;
    }
};
```

1254. 统计封闭岛屿的数目（中等）

1. 题目描述

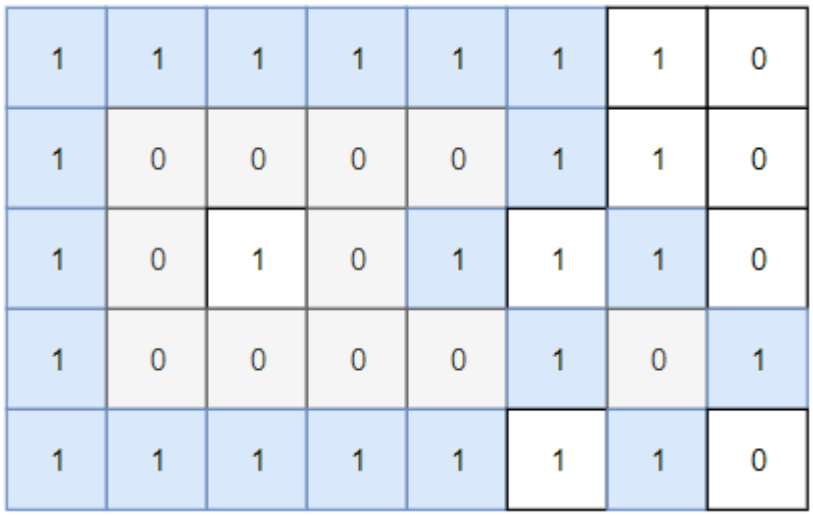
有一个二维矩阵 `grid` ，每个位置要么是陆地（记号为 `0` ）要么是水域（记号为 `1` ）。

我们从一块陆地出发，每次可以往上下左右 4 个方向相邻区域走，能走到的所有陆地区域，我们将其称为一座「**岛屿**」。

如果一座岛屿 **完全** 由水域包围，即陆地边缘上下左右所有相邻区域都是水域，那么我们将其称为 「**封闭岛屿**」。

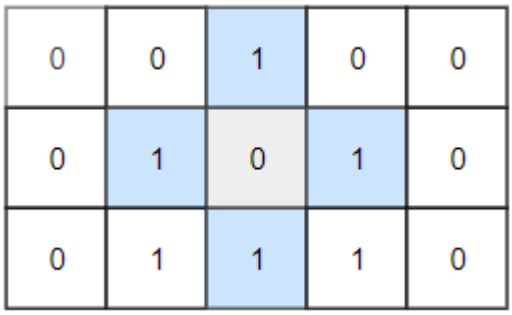
请返回封闭岛屿的数目。

示例 1：




```
输入：grid = [[1,1,1,1,1,1,1,0],[1,0,0,0,0,1,1,0],[1,0,1,0,1,1,1,0],
[1,0,0,0,0,1,0,1],[1,1,1,1,1,1,1,0]]
输出：2
解释：
灰色区域的岛屿是封闭岛屿，因为这座岛屿完全被水域包围（即被 1 区域包围）。
```

示例 2：




```
输入: grid = [[0,0,1,0,0],[0,1,0,1,0],[0,1,1,1,0]]
输出: 1
```

示例 3:

```
输入: grid = [[1,1,1,1,1,1,1],
              [1,0,0,0,0,0,1],
              [1,0,1,1,1,0,1],
              [1,0,1,0,1,0,1],
              [1,0,1,1,1,0,1],
              [1,0,0,0,0,0,1],
              [1,1,1,1,1,1,1]]
输出: 2
```

提示:

- `1 <= grid.length, grid[0].length <= 100`
- `0 <= grid[i][j] <=1`

2. 简单实现

先对边界的0进行广度优先搜索，去掉非封闭的岛屿，剩下的都是封闭岛屿

```
class Solution {
public:
    int m,n;
    void bfs(int i, int j, vector<vector<int>>& grid){
        queue<pair<int, int>> q;
        q.push(make_pair(i, j));
        grid[i][j] = 1;
        while(!q.empty()){
            int size = q.size();
            for(int i = 0; i < size; i++){
                int x = q.front().first;
                int y = q.front().second;
                q.pop();
                if(x > 0 && grid[x-1][y] == 0){
                    q.push(make_pair(x-1, y));
                    grid[x-1][y] = 1;
                }
                if(x < m-1 && grid[x+1][y] == 0){
                    q.push(make_pair(x+1, y));
                    grid[x+1][y] = 1;
                }
```

```
                if(y > 0 && grid[x][y-1] == 0){
                    q.push(make_pair(x, y-1));
                    grid[x][y-1] = 1;
                }
                if(y < n-1 && grid[x][y+1] == 0){
                    q.push(make_pair(x, y+1));
                    grid[x][y+1] = 1;
                }
            }
        }
    }
    int closedIsland(vector<vector<int>>& grid) {
        m = grid.size();
        n = grid[0].size();
        if(m <= 2 || n <= 2)//不可能存在封闭岛屿
            return 0;
        //去掉非封闭岛屿
        for(int j = 0; j < n; j++){
            if(grid[0][j] == 0)
                bfs(0, j, grid);
            if(grid[m-1][j] == 0)
                bfs(m-1, j, grid);
        }
        for(int i = 0; i < m; i++){
            if(grid[i][0] == 0)
                bfs(i, 0, grid);
            if(grid[i][n-1] == 0)
                bfs(i, n-1, grid);
        }
        //统计封闭岛屿
        int ans = 0;
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++){
                if(grid[i][j] == 0){
                    ans++;
                    bfs(i, j, grid);
                }
            }
        return ans;
    }
};
```

1255. 得分最高的单词集合（困难）

1. 题目描述

   你将会得到一份单词表 `words`，一个字母表 `letters` （可能会有重复字母），以及每个字母对应的得分情况表 `score`。

   请你帮忙计算玩家在单词拼写游戏中所能获得的「最高得分」：能够由 `letters` 里的字母拼写出的 **任意** 属于 `words` 单词子集中，分数最高的单词集合的得分。

   单词拼写游戏的规则概述如下：

- 玩家需要用字母表 `letters` 里的字母来拼写单词表 `words` 中的单词。
- 可以只使用字母表 `letters` 中的部分字母，但是每个字母最多被使用一次。
- 单词表 `words` 中每个单词只能计分（使用）一次。
- 根据字母得分情况表 `score`，字母 `'a'`, `'b'`, `'c'`, ... , `'z'` 对应的得分分别为 `score[0]`, `score[1]`, ..., `score[25]`。
- 本场游戏的「得分」是指：玩家所拼写出的单词集合里包含的所有字母的得分之和。

示例 1：

```
输入：words = ["dog","cat","dad","good"], letters =
["a","a","c","d","d","d","g","o","o"], score =
[1,0,9,5,0,0,3,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0]
输出：23
解释：
字母得分为  a=1, c=9, d=5, g=3, o=2
使用给定的字母表 letters，我们可以拼写单词 "dad" (5+1+5)和 "good" (3+2+2+5)，得分为 23 。
而单词 "dad" 和 "dog" 只能得到 21 分。
```

示例 2：

```
输入：words = ["xxxz","ax","bx","cx"], letters = ["z","a","b","c","x","x","x"],
score = [4,4,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0,10]
输出：27
解释：
字母得分为  a=4, b=4, c=4, x=5, z=10
使用给定的字母表 letters，我们可以组成单词 "ax" (4+5)， "bx" (4+5) 和 "cx" (4+5) ，总得分为 27 。
单词 "xxxz" 的得分仅为 25 。
```

示例 3：

```
输入：words = ["leetcode"], letters = ["l","e","t","c","o","d"], score =
[0,0,1,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0]
输出：0
解释：
字母 "e" 在字母表 letters 中只出现了一次，所以无法组成单词表 words 中的单词。
```

提示：

- `1 <= words.length <= 14`
- `1 <= words[i].length <= 15`
- `1 <= letters.length <= 100`
- `letters[i].length == 1`
- `score.length == 26`
- `0 <= score[i] <= 10`
- `words[i]` 和 `letters[i]` 只包含小写的英文字母。

2. 简单实现

回溯剪枝

```
class Solution {
```

```
public:
    int re;//结果答案
    //返回word的得分，若当前letters不足以构成word，则返回-1
    int getScore(string& word, vector<int>& cnt, vector<int>& score){
        int ans = 0;
        for(int i = 0; i < word.size(); i++)
            if(cnt[word[i]-'a'] <= 0)
                return -1;
            else{
                cnt[word[i]-'a']--;
                ans += score[word[i]-'a'];
            }
        return ans;
    }
    void search(vector<string>& words, int idx, vector<int>& cnt,
                vector<int>& score, int temp_score){
        if(idx == words.size()){
            if(temp_score > re)
                re = temp_score;
        }
        else{
            search(words, idx+1, cnt, score, temp_score);//不组成word[idx]

            vector<int> back = cnt;
            int s = getScore(words[idx], cnt, score);
            if(s > 0)//可以组成word[idx]
                search(words, idx+1, cnt, score, temp_score+s);
            cnt = back;
        }
    }
    int maxScoreWords(vector<string>& words, vector<char>& letters, vector<int>&
score) {
        vector<int> cnt = vector<int>(26, 0);//记录letters中每个字母的个数
        for(int i = 0; i < letters.size(); i++)
            cnt[letters[i]-'a']++;
        re = 0;
        search(words, 0, cnt, score, 0);
        return re;
    }
};
```