# 题目

给你一个m \* n的矩阵grid，矩阵中的元素无论是按行还是按列，都以非递增顺序排列。请你统计并返回grid中负数的数目。

示例 1：

输入：grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]

输出：8

解释：矩阵中共有 8 个负数。

示例 2：

输入：grid = [[3,2],[1,0]]

输出：0

提示：

m == grid.length

n == grid[i].length

1 <= m, n <= 100

-100 <= grid[i][j] <= 100

进阶：你可以设计一个时间复杂度为 O(n + m) 的解决方案吗？

# 分析

## 方法一：暴力

观察数据范围注意到矩阵大小不会超过 100∗100=104，所以我们可以遍历矩阵所有数，统计负数的个数。

代码：

class Solution {

public:

int countNegatives(vector<vector<int>>& grid) {

int num=0;

for (auto x:grid){

for (auto y:x){

if (y<0) num++;

}

}

return num;

}

};

复杂度分析：

时间复杂度：O(nm)，即矩阵元素的总个数。

空间复杂度：O(1)。

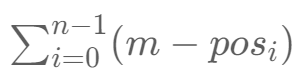
## 方法二：二分查找

注意到题目中给了一个性质，即矩阵中的元素无论是按行还是按列，都以非递增顺序排列，可以考虑把这个性质利用起来优化暴力。已知这个性质告诉了我们每一行的数都是有序的，所以我们通过二分查找可以找到每一行中从前往后的第一个负数，那么这个位置之后到这一行的末尾里所有的数必然是负数了，可以直接统计。

1、遍历矩阵的每一行。

2、二分查找到该行从前往后的第一个负数，考虑第i行，我们记这个位置为 posi，那么第i行 [posi,m−1]中的所有数都是负数，所以这一行对答案的贡献就是m−1−posi+1=m−posi。

3、最后的答案就是



代码：

class Solution {

public:

int countNegatives(vector<vector<int>>& grid) {

int num=0;

for (auto x:grid){

int l=0,r=(int)x.size()-1,pos=-1;

while (l<=r){

int mid=l+((r-l)>>1);

if (x[mid]<0){

pos=mid;

r=mid-1;

}

else l=mid+1;

}

if (~pos) num+=(int)x.size()-pos;// pos=-1表示这一行全是>=0的数，不能统计

}

return num;

}

};

另外一种写法：

class Solution {

public:

int countNegatives(vector<vector<int>>& grid) {

int m = grid.size();

int n = grid[0].size();

int count = 0;

for (int i = 0; i < m; ++i) {

int left = 0;

int right = n - 1;

// 二分查找最左边的负数的索引

while (left <= right) {

int mid = left + (right - left) / 2;

if (grid[i][mid] < 0) {

right = mid - 1;

} else {

left = mid + 1;

}

}

// 统计负数的个数

count += (n - left);

}

return count;

}

};

**复杂度分析：**

时间复杂度：二分查找一行的时间复杂度为logm，需要遍历n行，所以总时间复杂度是O(nlogm)。

空间复杂度：O(1)。