# 题目

给你一个矩阵 mat，其中每一行的元素都已经按 严格递增 顺序排好了。请你帮忙找出在所有这些行中 最小的公共元素。

如果矩阵中没有这样的公共元素，就请返回 -1。

示例：

输入：mat = [[1,2,3,4,5],[2,4,5,8,10],[3,5,7,9,11],[1,3,5,7,9]]

输出：5

提示：

1 <= mat.length, mat[i].length <= 500

1 <= mat[i][j] <= 10^4

mat[i] 已按严格递增顺序排列。

# 分析

综述

每行都按照 升序 排列，说明每行都没有重复元素。因此，如果一个元素出现在所有行，则该元素一定出现了 n 次（n 是行数）。

统计所有元素的出现次数，找出出现 n 次的最小元素。该方法具有线性复杂度，需要额外空间存储元素出现的次数。

另外，也可以在矩阵中直接使用二分查找。该方法不需要额外空间，但是该方法的时间复杂度较高。

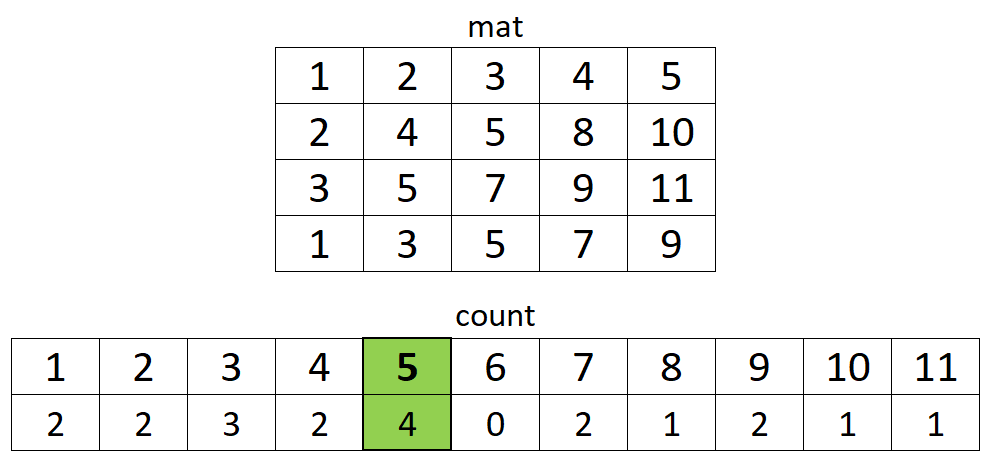
最后，在每一行设置指针跟踪元素。每次找到较小的元素，让其指向该行的下一位，直到所有的指针都指向一个公共元素。如果存在这样的元素，该方法具有线性复杂度，且使用的空间比统计元素出现次数的方法更少。

## 方法一：统计元素出现次数

逐行遍历所有元素，并统计每个元素的出现次数。因为元素在 [1...10000] 范围内，所以使用数组记录每个元素的出现次数。

然后，从左至右遍历数组，返回第一个出现 n 次的元素。顺便说一下，这就是计数排序方法。

对于无约束映射问题，可以使用有序 map 存储元素的出现次数。



算法

1、使用 i 遍历行。

使用 j 遍历列。

数组 count 中第 mat[i][j] 个元素计数加 1。

2、使用 k 从 1 遍历到 10000。

如果 count[k] 等于 n，返回 k。

1. 返回 -1。

代码：

int smallestCommonElement(vector<vector<int>>& mat) {

int count[10001] = {};

int n = mat.size(), m = mat[0].size();

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

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

++count[mat[i][j]];

}

}

for (int k = 1; k <= 10000; ++k) {

if (count[k] == n) {

return k;

}

}

return -1;

}

改进的解法

如果逐列计算元素，可以提高平均时间复杂度。这样，首先计算较小元素的出现次数，一旦某个元素出现次数为n，则直接返回。

对于非约束问题，如果逐列统计元素，可以使用无序map，它比初始解法的有序map更快。

int smallestCommonElement(vector<vector<int>>& mat) {

int count[10001] = {};

int n = mat.size(), m = mat[0].size();

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

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

if (++count[mat[i][j]] == n) {

return mat[i][j];

}

}

}

return -1;

}

处理重复项

如果元素是非降序排列，则需要修改方法处理重复项。例如：下面例子实际上最小公共重复项为 5。但初始解法返回 4，改进解法返回 7：

[[1,2,3,4,5],[5,7,7,7,7],[5,7,7,7,7],[1,2,4,4,5],[1,2,4,4,5]]

这两种解法中很容易处理重复项。因为行内元素有序，如果当前元素等于前一个元素，则直接忽略。

复杂度分析

时间复杂度：O(nm)，其中n和m是行和列的数量。

空间复杂度：

约束问题：O(10000)=O(1)。

无约束问题：O(k)，其中k是出现过的元素数量。

## 方法二：二分搜索

遍历第一行所有元素，然后在其余所有行使用二分搜索检查是否存在该元素。

算法

1、遍历第一行所有元素。

初始 found 为 true。

对每一行：

使用二分搜索查找该元素是否存在。

如果不存在，令 found 为 false，退出循环。

如果 found 为 true，返回该元素。

1. 返回 -1。

代码：

int smallestCommonElement(vector<vector<int>>& mat) {

int n = mat.size(), m = mat[0].size();

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

bool found = true;

for (int i = 1; i < n && found; ++i) {

found = binary\_search(begin(mat[i]), end(mat[i]), mat[0][j]);

}

if (found) {

return mat[0][j];

}

}

return -1;

}

改进的解法

上面解法中，每次都搜索整行。如果从上一次搜索返回位置开始搜索可以降低平均时间复杂度。如果一行所有元素都小于查找值，则返回 -1。

注意：C++ 中，如果存在最小公共元素，返回的 lower\_bound 等于该元素位置；否则返回大于搜索值第一个数的位置。在 Java 中，binarySearch 返回值与 lower\_bound 相同。这两种情况下，如果所有元素都小于搜索元素，则指向最后一个元素。

int smallestCommonElement(vector<vector<int>>& mat) {

int n = mat.size(), m = mat[0].size();

vector<int> pos(n);

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

bool found = true;

for (int i = 1; i < n && found; ++i) {

pos[i] = lower\_bound(begin(mat[i]) + pos[i], end(mat[i]), mat[0][j]) - begin(mat[i]);

if (pos[i] >= m) {

return -1;

}

found = mat[i][pos[i]] == mat[0][j];

}

if (found) {

return mat[0][j];

}

}

return -1;

}

处理重复项

因为每行都搜索，所以重复元素不影响最终结果。

复杂度分析

时间复杂度：O(mnlogm)。

遍历第一行的m个元素。

对于每个元素，在n行中使用二分搜索查找m个元素。

空间复杂度：

初始解法：O(1)。

改进解法：O(n)，存储每行的搜索位置。