# 题目

给你一个下标从0开始的二维整数矩阵grid，大小为n \* n ，其中的值在 [1, n2] 范围内。除了a出现两次，b 缺失之外，每个整数都恰好出现一次。

任务是找出重复的数字a和缺失的数字b。

返回一个下标从0开始、长度为2的整数数组ans，其中ans[0]等于a，ans[1]等于b。

示例 1：

输入：grid = [[1,3],[2,2]]

输出：[2,4]

解释：数字 2 重复，数字 4 缺失，所以答案是 [2,4] 。

示例 2：

输入：grid = [[9,1,7],[8,9,2],[3,4,6]]

输出：[9,5]

解释：数字 9 重复，数字 5 缺失，所以答案是 [9,5] 。

提示：

2 <= n == grid.length == grid[i].length <= 50

1 <= grid[i][j] <= n \* n

对于所有满足1 <= x <= n \* n的x，恰好存在一个x与矩阵中的任何成员都不相等。

对于所有满足1 <= x <= n \* n的x ，恰好存在一个x与矩阵中的两个成员相等。

除上述的两个之外，对于所有满足1 <= x <= n \* n的x，都恰好存在一对i, j 满足0 <= i, j <= n - 1且grid[i][j] == x。

# 分析

## 方法一：数组

思路：

用数组统计每个数的出现次数。

代码：

class Solution {

public:

std::vector<int> findMissingAndRepeatedValues(std::vector<std::vector<int>>& grid) {

int n = grid.size();

vector<int> cnt(n \* n + 1, 0); // 数字的大小为[1,n\*n],这里的cnt[0]是多余的不使用

for (auto &row : grid) {

for (auto &col : row) {

cnt[col]++;

}

}

vector<int> ans(2, 0);

// 为了方便直接从下标1开始遍历

for (int i = 1; i <= n \* n; i++) {

if (cnt[i] == 2) {

ans[0] = i;

} else if (cnt[i] == 0) {

ans[1] = i;

}

}

return ans;

}

};

复杂度分析

时间复杂度：O(n2)，其中n为grid的行数和列数。

空间复杂度：O(n2)。

## 方法二：哈希表

思路：

代码：

class Solution {

public:

std::vector<int> findMissingAndRepeatedValues(std::vector<std::vector<int>>& grid) {

std::vector<int> ans(2, 0); // 初始化一个长度为2的数组用于存储结果

std::unordered\_map<int, int> cnt;

int n = grid.size();

int totalNumbers = n \* n; // 总共应该有的数字数量

// 统计每个数字出现的次数

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

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

cnt[grid[i][j]]++;

}

}

// 查找重复的数字

for (int num = 1; num <= totalNumbers; num++) {

if (cnt[num] == 2) {

ans[0] = num; // 重复的数字

break;

}

}

// 查找缺失的数字

for (int num = 1; num <= totalNumbers; num++) {

if (cnt[num] == 0) {

ans[1] = num; // 缺失的数字

break;

}

}

return ans;

}

};

上述代码在最后遍历的时候，遍历了两次，实际上我们可以通过一次遍历grid数组来同时找到重复的数字和缺失的数字。我们可以使用一个unordered\_map来跟踪每个数字出现的次数，并使用另一个固定大小的数组（在这个情况下是长度为n \* n的数组）来标记哪些数字已经出现。通过组合这两种方法，我们可以在单次遍历中完成查找。修改如下：

class Solution {

public:

std::vector<int> findMissingAndRepeatedValues(std::vector<std::vector<int>>& grid) {

int n = grid.size();

int totalNumbers = n \* n;

std::vector<bool> seen(totalNumbers + 1, false); // 用于标记数字是否出现过的数组

std::unordered\_map<int, int> cnt; // 用于统计数字出现次数的哈希表

std::vector<int> ans(2, 0); // 存放结果

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

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

int num = grid[i][j];

if (seen[num]) {

ans[0] = num; // 找到重复的数字

} else {

seen[num] = true; // 标记数字已出现

}

cnt[num]++; // 统计数字出现的次数

}

}

// 查找缺失的数字

for (int num = 1; num <= totalNumbers; ++num) {

if (!seen[num]) {

ans[1] = num; // 找到缺失的数字

break;

}

}

return ans;

}

};

说明：

在这个解决方案中，seen数组用于标记从1到n \* n的每个数字是否已经在grid中出现过。cnt哈希表用于记录每个数字出现的次数，但这在寻找重复数字时并不是必需的，因为seen数组已经足够。然而，如果您想要计算每个数字的确切出现次数，那么保留cnt也是有用的。

请注意，在找到重复数字后，我们仍然需要遍历seen数组来找到缺失的数字，因为仅使用cnt哈希表无法直接得知哪个数字是缺失的。如果输入矩阵grid不满足所有数字恰好出现一次（除了一个重复和一个缺失）的条件，这个代码可能无法给出正确的结果。在实际应用中，您可能需要添加额外的错误检查来确保输入的有效性。