【华为OD机考 统一考试机试C卷】最小矩阵宽度（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷） 。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多 ，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

给定一个矩阵，包含 N * M 个整数，和一个包含 K 个整数的数组。

现在要求在这个矩阵中找一个宽度最小的子矩阵，要求子矩阵包含数组中所有的整数。

输入描述

第一行输入两个正整数 N，M，表示矩阵大小。

接下来 N 行 M 列表示矩阵内容。

下一行包含一个正整数 K。

下一行包含 K 个整数，表示所需包含的数组，K 个整数可能存在重复数字。

所有输入数据小于1000。

输出描述

输出包含一个整数，表示满足要求子矩阵的最小宽度，若找不到，输出-1。

用例1

输入

```
2 5
1 2 2 3 1
2 3 2 3 2
3
1 2 3
```

输出

```
2
```

说明

> 矩阵第0、3列包含了1，2，3，矩阵第3，4列包含了1，2，3

用例2

输入

```
2 5
1 2 2 3 1
1 3 2 3 4
3
1 1 4
```

输出

```
5
```

说明

> 矩阵第1、2、3、4、5列包含了1、1、4

解题思路

C++

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <climits>

using namespace std;

// 检查矩阵的指定列区间是否包含所有要求的整数
bool containsAll(const vector<vector<int>>& matrix, int left, int right, const vector<int>& requiredNums) {
    unordered_map<int, int> countMap; // 使用哈希表记录每个要求整数的数量
    for (int num : requiredNums) { // 初始化要求整数的数量
        ++countMap[num];
    }

    for (int i = 0; i < matrix.size(); ++i) { // 遍历矩阵的每一行
        for (int j = left; j <= right; ++j) { // 遍历指定的列区间
            if (countMap.find(matrix[i][j]) != countMap.end()) { // 如果当前元素是要求的整数之一
                --countMap[matrix[i][j]]; // 减少哈希表中对应整数的数量
                if (countMap[matrix[i][j]] == 0) { // 如果某个整数的数量减少到0
                    countMap.erase(matrix[i][j]); // 从哈希表中移除该整数
                }
            }
        }
    }
    return countMap.empty(); // 如果哈希表为空，则表示所有要求的整数都包含在内
}

// 寻找最小宽度子矩阵的方法
int findMinWidthSubmatrix(const vector<vector<int>>& matrix, const vector<int>& requiredNums) {
    int minWidth = INT_MAX; // 设置最小宽度初始值为最大整数
    for (int left = 0; left < matrix[0].size(); ++left) { // 遍历所有可能的左边界
        for (int right = left; right < matrix[0].size(); ++right) { // 遍历所有可能的右边界
            if (containsAll(matrix, left, right, requiredNums)) { // 检查当前列区间是否包含所有要求的整数
                minWidth = min(minWidth, right - left + 1); // 更新最小宽度
            }
        }
```

```
    }
    return minWidth == INT_MAX ? -1 : minWidth; // 如果找到符合条件的子矩阵，返回最小宽度；否则返回-1
}

int main() {
    int N, M; // 矩阵的行数和列数
    cin >> N >> M; // 读取行数和列数
    vector<vector<int>> matrix(N, vector<int>(M)); // 初始化矩阵
    for (int i = 0; i < N; ++i) { // 读取矩阵中的每个元素
        for (int j = 0; j < M; ++j) {
            cin >> matrix[i][j];
        }
    }

    int K; // 要包含的整数数组的长度
    cin >> K; // 读取长度
    vector<int> requiredNums(K); // 初始化要包含的整数数组
    for (int i = 0; i < K; ++i) { // 读取要包含的整数
        cin >> requiredNums[i];
    }

    cout << findMinWidthSubmatrix(matrix, requiredNums) << endl; // 输出找到的最小宽度子矩阵的宽度
    return 0;
}
```

Java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 使用Scanner类读取输入
        Scanner scanner = new Scanner(System.in);
        // 读取矩阵的行数N和列数M
        int N = scanner.nextInt();
        int M = scanner.nextInt();
        // 初始化矩阵
        int[][] matrix = new int[N][M];
```

```
        // 读取矩阵中的每个元素
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < M; j++) {
                matrix[i][j] = scanner.nextInt();
            }
        }
        // 读取数组的长度K
        int K = scanner.nextInt();
        // 初始化要包含的整数数组
        int[] requiredNums = new int[K];
        // 读取要包含的整数
        for (int i = 0; i < K; i++) {
            requiredNums[i] = scanner.nextInt();
        }
        // 输出找到的最小宽度子矩阵的宽度
        System.out.println(findMinWidthSubmatrix(matrix, requiredNums));
    }

    // 寻找最小宽度子矩阵的方法
    private static int findMinWidthSubmatrix(int[][] matrix, int[] requiredNums) {
        // 设置最小宽度初始值为最大整数
        int minWidth = Integer.MAX_VALUE;
        // 遍历矩阵的所有列组合
        for (int left = 0; left < matrix[0].length; left++) {
            for (int right = left; right < matrix[0].length; right++) {
                // 检查当前列组合是否包含所有要求的整数
                if (containsAll(matrix, left, right, requiredNums)) {
                    // 更新最小宽度
                    minWidth = Math.min(minWidth, right - left + 1);


                }
            }
        }
        // 如果找到符合条件的子矩阵，返回最小宽度；否则返回-1
        return minWidth == Integer.MAX_VALUE ? -1 : minWidth;
    }

    // 检查矩阵的指定列区间是否包含所有要求的整数
    private static boolean containsAll(int[][] matrix, int left, int right, int[] requiredNums) {
```

```
        // 使用HashMap记录每个要求整数的数量
        Map<Integer, Integer> countMap = new HashMap<>();
        for (int num : requiredNums) {
            countMap.put(num, countMap.getOrDefault(num, 0) + 1);
        }
        // 遍历矩阵的指定列区间
        for (int i = 0; i < matrix.length; i++) {
            for (int j = left; j <= right; j++) {
                // 如果当前元素是要求的整数之一，减少其计数
                if (countMap.containsKey(matrix[i][j])) {
                    countMap.put(matrix[i][j], countMap.get(matrix[i][j]) - 1);
                    // 如果某个整数的计数降至0，从HashMap中移除
                    if (countMap.get(matrix[i][j]) == 0) {
                        countMap.remove(matrix[i][j]);
                    }
                }
            }
        }
        // 如果HashMap为空，表示所有要求的整数都包含在内
        return countMap.isEmpty();
    }
}
```

javaScript

```
// 导入readline模块，用于从命令行读取输入
const readline = require('readline');
// 创建readline接口实例
const rl = readline.createInterface({
    input: process.stdin, // 输入流为标准输入
    output: process.stdout // 输出流为标准输出
});

// 检查矩阵的指定列区间是否包含所有要求的整数
function containsAll(matrix, left, right, requiredNums) {
    // 创建一个Map来存储每个需要的数字及其出现的次数
    const countMap = new Map();
    requiredNums.forEach(num => {
        // 如果Map中已经有这个数字，那么获取它的次数并加一；否则，次数为1
        countMap.set(num, (countMap.get(num) || 0) + 1);
```

```
    });

    // 遍历矩阵的每一行
    for (let i = 0; i < matrix.length; i++) {
        // 遍历指定的列区间
        for (let j = left; j <= right; j++) {
            // 如果当前元素是需要的数字之一
            if (countMap.has(matrix[i][j])) {
                // 减少Map中该数字的次数
                countMap.set(matrix[i][j], countMap.get(matrix[i][j]) - 1);
                // 如果某个数字的次数降至0，则从Map中删除该数字
                if (countMap.get(matrix[i][j]) === 0) {
                    countMap.delete(matrix[i][j]);
                }
            }
        }
    }
    // 如果Map为空，表示所有需要的数字都在指定的列区间中
    return countMap.size === 0;
}

// 寻找最小宽度子矩阵的方法
function findMinWidthSubmatrix(matrix, requiredNums) {
    // 初始化最小宽度为Infinity
    let minWidth = Infinity;
    // 遍历矩阵的所有列组合
    for (let left = 0; left < matrix[0].length; left++) {
        for (let right = left; right < matrix[0].length; right++) {
            // 如果当前列组合包含所有需要的数字
            if (containsAll(matrix, left, right, requiredNums)) {
                // 更新最小宽度
                minWidth = Math.min(minWidth, right - left + 1);
            }
        }
    }
    // 如果找到符合条件的子矩阵，返回最小宽度；否则返回-1
    return minWidth === Infinity ? -1 : minWidth;
}

// 创建一个数组来存储输入的每一行
```

```javascript
let input = [];
// 当接收到一行输入时
rl.on('line', (line) => {
    // 将这一行添加到输入数组中
    input.push(line);
    // 如果输入的行数等于矩阵的行数加3（包括矩阵的大小、矩阵本身和需要的数字）
    if (input.length === parseInt(input[0].split(' ')[0]) + 3) {
        // 解析矩阵的行数和列数
        const [N, M] = input[0].split(' ').map(Number);
        // 解析矩阵
        const matrix = input.slice(1, N + 1).map(row => row.split(' ').map(Number));
        // 解析需要的数字
        const requiredNums = input[input.length - 1].split(' ').map(Number);
        // 调用findMinWidthSubmatrix函数并打印结果
        console.log(findMinWidthSubmatrix(matrix, requiredNums));
        // 关闭readline接口
        rl.close();
    }
});
```

Python

```python
def contains_all(matrix, left, right, required_nums):
    """
    检查矩阵的指定列区间是否包含所有要求的整数
    :param matrix: 二维矩阵
    :param left: 检查区间的左边界
    :param right: 检查区间的右边界
    :param required_nums: 需要包含的整数列表
    :return: 如果包含所有要求的整数返回True，否则返回False
    """
    # 创建一个字典来存储每个要求整数的数量
    count_map = {num: required_nums.count(num) for num in set(required_nums)}
    # 遍历矩阵的指定列区间
    for row in matrix:
        for num in row[left:right + 1]:
            if num in count_map:
                count_map[num] -= 1
                if count_map[num] == 0:
                    del count_map[num]
```

```
    # 如果字典为空，表示所有要求的整数都包含在内
    return not count_map

def find_min_width_submatrix(matrix, required_nums):
    """
    寻找最小宽度子矩阵的方法
    :param matrix: 二维矩阵
    :param required_nums: 需要包含的整数列表
    :return: 满足要求子矩阵的最小宽度，如果不存在则返回-1
    """
    # 设置最小宽度初始值为无穷大
    min_width = float('inf')
    # 遍历矩阵的所有列组合
    for left in range(len(matrix[0])):
        for right in range(left, len(matrix[0])):
            # 检查当前列组合是否包含所有要求的整数
            if contains_all(matrix, left, right, required_nums):
                # 更新最小宽度
                min_width = min(min_width, right - left + 1)
    # 如果找到符合条件的子矩阵，返回最小宽度；否则返回-1
    return min_width if min_width != float('inf') else -1

# 主函数，用于读取输入并调用函数输出结果

N, M = map(int, input().split())
# 读取矩阵内容
matrix = [list(map(int, input().split())) for _ in range(N)]
K = int(input())  # 读取要求的整数个数
required_nums = list(map(int, input().split()))  # 读取要求的整数列表
# 输出找到的最小宽度子矩阵的宽度
print(find_min_width_submatrix(matrix, required_nums))
```

C语言

```
#include <stdio.h>
#include <limits.h>
#include <string.h>

#define MAX_SIZE 1000
```

```cpp

// 检查矩阵的指定列区间是否包含所有要求的整数
int containsAll(int matrix[MAX_SIZE][MAX_SIZE], int N, int M, int left, int right, int requiredNums[], int K) {
    int countMap[MAX_SIZE] = {0}; // 使用数组模拟哈希表记录每个要求整数的数量
    for (int i = 0; i < K; ++i) { // 初始化要求整数的数量
        countMap[requiredNums[i]]++;
    }

    for (int i = 0; i < N; ++i) { // 遍历矩阵的每一行
        for (int j = left; j <= right; ++j) { // 遍历指定的列区间
            if (countMap[matrix[i][j]] > 0) { // 如果当前元素是要求的整数之一
                --countMap[matrix[i][j]]; // 减少数组中对应整数的数量
            }
        }
    }

    for (int i = 0; i < K; ++i) { // 检查是否所有要求的整数都被包含
        if (countMap[requiredNums[i]] > 0) {
            return 0; // 如果有整数未被包含，返回0
        }
    }
    return 1; // 如果所有要求的整数都被包含，返回1
}

// 寻找最小宽度子矩阵的方法
int findMinWidthSubmatrix(int matrix[MAX_SIZE][MAX_SIZE], int N, int M, int requiredNums[], int K) {
    int minWidth = INT_MAX; // 设置最小宽度初始值为最大整数
    for (int left = 0; left < M; ++left) { // 遍历所有可能的左边界
        for (int right = left; right < M; ++right) { // 遍历所有可能的右边界
            if (containsAll(matrix, N, M, left, right, requiredNums, K)) { // 检查当前列区间是否包含所有要求的整数
                if (right - left + 1 < minWidth) {
                    minWidth = right - left + 1; // 更新最小宽度
                }
            }
        }
    }
    return minWidth == INT_MAX ? -1 : minWidth; // 如果找到符合条件的子矩阵，返回最小宽度；否则返回-1
}

int main() {

```

```
    int N, M; // 矩阵的行数和列数
    scanf("%d %d", &N, &M); // 读取行数和列数
    int matrix[MAX_SIZE][MAX_SIZE]; // 初始化矩阵
    for (int i = 0; i < N; ++i) { // 读取矩阵中的每个元素
        for (int j = 0; j < M; ++j) {
            scanf("%d", &matrix[i][j]);
        }
    }

    int K; // 要包含的整数数组的长度
    scanf("%d", &K); // 读取长度
    int requiredNums[MAX_SIZE]; // 初始化要包含的整数数组
    for (int i = 0; i < K; ++i) { // 读取要包含的整数
        scanf("%d", &requiredNums[i]);
    }

    printf("%d\n", findMinWidthSubmatrix(matrix, N, M, requiredNums, K)); // 输出找到的最小宽度子矩阵的宽度
    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师