【华为OD机考 统一考试机试C卷】分配土地（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
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

题目描述

从前有个村庄，村民们喜欢在各种田地上插上小旗子，旗子上标识了各种不同的数字。

某天集体村民决定将覆盖相同数字的最小矩阵形的土地分配给村里做出巨大贡献的村民，请问此次分配土地，做出贡献的村民种最大会分配多大面积?

输入描述

第一行输入 m 和 n，

- m 代表村子的土地的长
- n 代表土地的宽

第二行开始输入地图上的具体标识

输出描述

此次分配土地，做出贡献的村民种最大会分配多大面积

备注

旗子上的数字为1~500，土地边长不超过500

未插旗子的土地用0标识

用例1

输入

```
3 3
1 0 1
0 0 0
0 1 0
```

输出

```
9
```

说明

> 土地上的旗子为1，其坐标分别为(0,0)，(2,1)以及(0,2)，为了覆盖所有旗子，矩阵需要覆盖的横坐标为0和2，纵坐标为0和2，所以面积为9，即（2-0+1）*（2-0+1）= 9

用例2

输入

```
3 3
1 0 2
0 0 0
0 3 4
```

输出

```
1
```

说明

> 由于不存在成对的小旗子，故而返回1，即一块土地的面积。

解题思路

1. 初始化：
    - 读取用户输入的土地的长 `m` 和宽 `n` 。
    - 创建一个二维数组 `land[m][n]` 来存储土地上每个位置的标识。
    - 创建两个二维数组 `minPos[501][2]` 和 `maxPos[501][2]` 来分别存储每个标识数字的最小和最大位置。数组的大小设为501是因为题目中提到数字的范围是1到500。
    - 将 `minPos` 的每个位置初始化为 `(m, n)` ， `maxPos` 的每个位置初始化为 `(-1, -1)` ，这是因为我们需要在后续的遍历中通过比较来找到实际的最小和最大值。
2. 遍历土地：
    - 通过双层循环遍历土地上的每个位置 `(i, j)` 。
    - 对于每个位置的数字 `num` ，如果 `num` 不是0，更新 `minPos[num]` 和 `maxPos[num]` 。 `minPos[num]` 记录该数字出现的最小行号和列号， `maxPos[num]` 记录该数字出现的最大行号和列号。
3. 计算面积：
    - 初始化一个变量 `maxArea` 来记录最大面积，初始值设为0。
    - 遍历每个可能的数字（1到500），如果该数字的最小位置小于或等于最大位置（确保该数字在土地上至少出现一次），则计算该数字对应的土地面积。
    - 面积计算公式为 `(maxPos[i][0] - minPos[i][0] + 1) * (maxPos[i][1] - minPos[i][1] + 1)` 。
    - 更新 `maxArea` 为所有计算出的面积中的最大值。
4. 输出结果：
    - 打印出最大面积 `maxArea` 。

模拟求解过程

假设输入的用例1如下：

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

这表示土地的长 `m` 为3，宽 `n` 为3，土地上的数字分布如输入所示。

按照解题思路进行模拟：

1. 初始化：

   `land` 数组被填充为：

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

   - `minPos` 和 `maxPos` 数组被初始化。

2. 遍历土地：

   更新 `minPos` 和 `maxPos` 数组：

   - `minPos[1]` 更新为 `(0, 0)`，`maxPos[1]` 更新为 `(2, 2)`。
   - `minPos[2]` 更新为 `(0, 1)`，`maxPos[2]` 更新为 `(0, 1)`。
   - `minPos[3]` 更新为 `(1, 0)`，`maxPos[3]` 更新为 `(1, 2)`。

3. 计算面积：

   计算每个数字的面积：

   - 对于数字1：面积为 `(2 - 0 + 1) * (2 - 0 + 1) = 9`。
   - 对于数字2：面积为 `(0 - 0 + 1) * (1 - 1 + 1) = 1`。
   - 对于数字3：面积为 `(1 - 1 + 1) * (2 - 0 + 1) = 3`。

   - `maxArea` 更新为9。

C++

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
using namespace std;

int main() {
    // 读取土地的长和宽
    int m, n;
    cin >> m >> n;

    // 创建一个二维数组来存储土地上的标识
    vector<vector<int>> land(m, vector<int>(n));
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> land[i][j];
        }
    }

    // 使用哈希表来存储每个数字的最小和最大位置
    unordered_map<int, pair<int, int>> minPos, maxPos;

    // 遍历每块土地，更新每个数字的最小和最大位置
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            int num = land[i][j];
            if (num != 0) {
                if (minPos.find(num) == minPos.end()) {
                    minPos[num] = {i, j};
                    maxPos[num] = {i, j};
                } else {
                    minPos[num] = {min(minPos[num].first, i), min(minPos[num].second, j)};
                    maxPos[num] = {max(maxPos[num].first, i), max(maxPos[num].second, j)};
                }
            }
        }
    }

    // 初始化最大面积为0

```

```
    int maxArea = 0;

    // 遍历每个数字，计算其对应的面积，并更新最大面积
    for (auto &p : minPos) {
        int num = p.first;
        int area = (maxPos[num].first - minPos[num].first + 1) * (maxPos[num].second - minPos[num].second + 1);
        maxArea = max(maxArea, area);
    }

    // 打印最大面积
    cout << maxArea << endl;
    return 0;
}
```

Java

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

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // 读取土地的长和宽
        int m = scanner.nextInt();
        int n = scanner.nextInt();

        // 创建一个二维数组来存储土地上的标识
        int[][] land = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                land[i][j] = scanner.nextInt();
            }
        }

        // 使用哈希表来存储每个数字的最小和最大位置
        Map<Integer, int[]> minPos = new HashMap<>();
        Map<Integer, int[]> maxPos = new HashMap<>();

        // 遍历每块土地，更新每个数字的最小和最大位置
```

```
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                int num = land[i][j];
                if (num != 0) {
                    if (!minPos.containsKey(num)) {
                        minPos.put(num, new int[]{i, j});
                        maxPos.put(num, new int[]{i, j});
                    } else {
                        minPos.get(num)[0] = Math.min(minPos.get(num)[0], i);
                        minPos.get(num)[1] = Math.min(minPos.get(num)[1], j);
                        maxPos.get(num)[0] = Math.max(maxPos.get(num)[0], i);
                        maxPos.get(num)[1] = Math.max(maxPos.get(num)[1], j);
                    }
                }
            }
        }

        // 初始化最大面积为0
        int maxArea = 0;

        // 遍历每个数字，计算其对应的面积，并更新最大面积
        for (Integer num : minPos.keySet()) {
            int[] min = minPos.get(num);
            int[] max = maxPos.get(num);
            int area = (max[0] - min[0] + 1) * (max[1] - min[1] + 1);
            maxArea = Math.max(maxArea, area);
        }

        // 打印最大面积
        System.out.println(maxArea);
        scanner.close();
    }
}
```

javaScript

```
const readline = require('readline');
// 创建 readline 接口实例
const rl = readline.createInterface({
    input: process.stdin, // Node.js 标准输入
```

```
    output: process.stdout // Node.js 标准输出
});

// 存储输入的所有行
let lines = [];
// 初始化土地的长和宽
let m = 0, n = 0;
// 记录已读取的行数
let lineCount = 0;

// 监听 line 事件，每次输入后触发
rl.on('line', (line) => {
    if (lineCount === 0) {
        // 第一行读取土地的长和宽
        [m, n] = line.split(' ').map(Number);
    } else {
        // 从第二行开始读取土地上的标识，并存储到 lines 数组中
        lines.push(line.split(' ').map(Number));
        // 当读取的行数等于土地的长度时，处理土地数据
        if (lines.length === m) {
            processLand(lines); // 调用处理土地数据的函数
            rl.close(); // 关闭 readline 接口实例
        }
    }
    lineCount++; // 行数加一
});

// 处理土地数据的函数
function processLand(land) {
    // 存储每个数字的最小位置
    let minPos = {};
    // 存储每个数字的最大位置
    let maxPos = {};
    // 初始化最大面积为 0
    let maxArea = 0;

    // 遍历土地的每个位置
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            let num = land[i][j];
```

```
            // 如果当前位置的数字不为 0
            if (num !== 0) {
                // 如果当前数字是第一次出现，则记录其位置为最小和最大位置
                if (!minPos[num]) {
                    minPos[num] = [i, j];
                    maxPos[num] = [i, j];
                } else {
                    // 更新当前数字的最小和最大位置
                    minPos[num] = [Math.min(minPos[num][0], i), Math.min(minPos[num][1], j)];
                    maxPos[num] = [Math.max(maxPos[num][0], i), Math.max(maxPos[num][1], j)];
                }
            }
        }
    }

    // 遍历记录的每个数字的位置
    for (let num in minPos) {
        // 计算每个数字对应的矩形面积
        let area = (maxPos[num][0] - minPos[num][0] + 1) * (maxPos[num][1] - minPos[num][1] + 1);
        // 更新最大面积
        maxArea = Math.max(maxArea, area);
    }

    // 输出最大面积
    console.log(maxArea);
}
```

Python

```
# 读取土地的长和宽
m, n = map(int, input().split())

# 创建一个二维数组来存储土地上的标识
land = [list(map(int, input().split())) for _ in range(m)]

# 使用字典来存储每个数字的最小和最大位置
minPos = {}
maxPos = {}

# 遍历每块土地，更新每个数字的最小和最大位置
```

```python
for i in range(m):
    for j in range(n):
        num = land[i][j]
        if num != 0:
            if num not in minPos:
                minPos[num] = (i, j)
                maxPos[num] = (i, j)
            else:
                minPos[num] = (min(minPos[num][0], i), min(minPos[num][1], j))
                maxPos[num] = (max(maxPos[num][0], i), max(maxPos[num][1], j))

# 初始化最大面积为0
maxArea = 0

# 遍历每个数字，计算其对应的面积，并更新最大面积
for num in minPos:
    area = (maxPos[num][0] - minPos[num][0] + 1) * (maxPos[num][1] - minPos[num][1] + 1)
    maxArea = max(maxArea, area)

# 打印最大面积
print(maxArea)
```

C语言

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_NUM 501

// 定义一个结构体来存储每个数字的最小和最大位置
typedef struct {
    int min_x, min_y, max_x, max_y;
} Pos;

int main() {
    // 读取土地的长和宽
    int m, n;
    scanf("%d %d", &m, &n);

```

```
    // 创建一个二维数组来存储土地上的标识
    int land[m][n];
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &land[i][j]);
        }
    }

    // 初始化每个数字的最小和最大位置
    Pos pos[MAX_NUM];
    memset(pos, -1, sizeof(pos));

    // 遍历每块土地，更新每个数字的最小和最大位置
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            int num = land[i][j];
            if (num != 0) {
                if (pos[num].min_x == -1) {
                    pos[num].min_x = pos[num].max_x = i;
                    pos[num].min_y = pos[num].max_y = j;
                } else {
                    if (i < pos[num].min_x) pos[num].min_x = i;
                    if (i > pos[num].max_x) pos[num].max_x = i;
                    if (j < pos[num].min_y) pos[num].min_y = j;
                    if (j > pos[num].max_y) pos[num].max_y = j;
                }
            }
        }
    }

    // 初始化最大面积为0
    int maxArea = 0;

    // 遍历每个数字，计算其对应的面积，并更新最大面积
    for (int num = 1; num < MAX_NUM; ++num) {
        if (pos[num].min_x != -1) {
            int area = (pos[num].max_x - pos[num].min_x + 1) * (pos[num].max_y - pos[num].min_y + 1);
            if (area > maxArea) maxArea = area;
        }
    }
```

```
    // 打印最大面积
    printf("%d\n", maxArea);
    return 0;
}
```

完整用例

用例1

```
5 5
1 2 0 0 3
4 0 0 0 0
0 0 5 0 6
0 0 0 7 0
8 0 0 0 9
```

用例2

```
4 4
1 0 2 3
0 0 0 0
4 0 0 0
5 6 7 0
```

用例3

```
6 6
1 0 0 0 0 2
0 0 0 0 0 0
3 0 0 0 0 4
0 0 0 0 0 0
5 0 0 0 0 6
0 0 0 0 0 0
```

用例4

```
2 3
1 0 2
0 3 0
```

用例5

```
4 4
0 0 0 0
1 1 1 1
0 0 0 0
0 0 0 0
```

用例6

```
10 10
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

用例7

```
3 3
0 0 0
0 0 0
0 0 0
```

用例8

```
5 5
0 1 0 1 0
1 0 0 0 1
0 0 1 0 0
```

```
1 0 0 0 1
0 1 0 1 0
```

用例9

```
5 5
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
```

用例10

```
4 4
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

文章目录