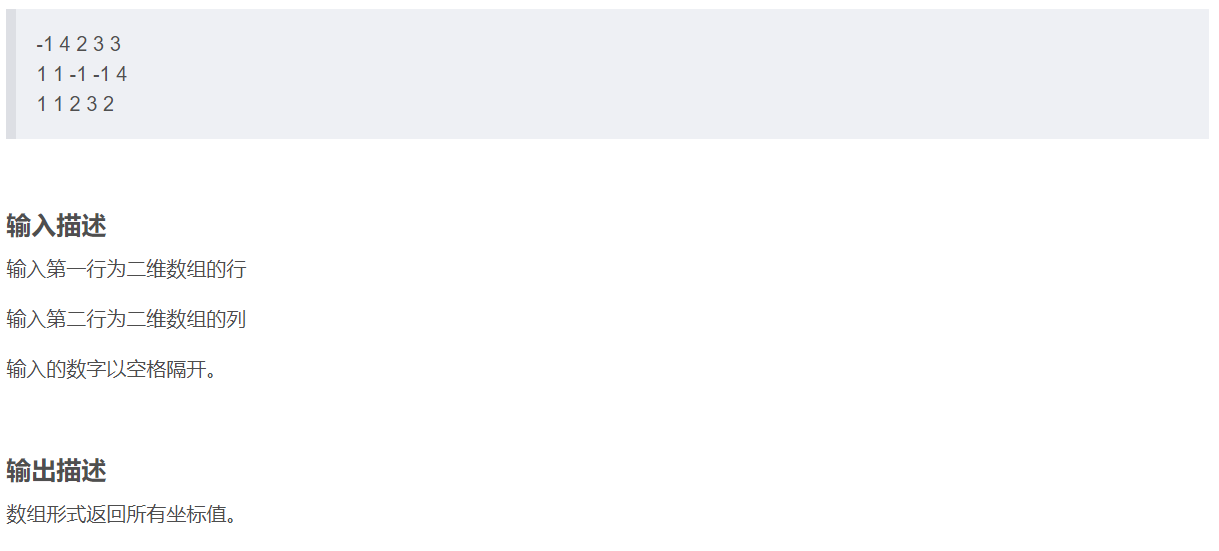
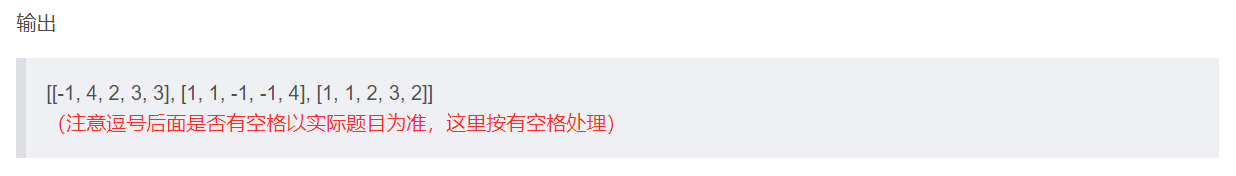
# **E卷-找数字[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**











import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取二维数组的行数

int n = scanner.nextInt();

// 读取二维数组的列数

int m = scanner.nextInt();

// 初始化二维数组

int[][] arr = new int[n][m];

// 循环读取每一行的输入，并将其转换为整数数组

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

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

arr[i][j] = scanner.nextInt();

}

}

// 初始化结果数组

int[][] result = new int[n][m];

// 遍历二维数组的每个元素

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

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

// 设置默认结果为-1

int res = -1;

// 遍历二维数组的每个元素，寻找相等的值

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

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

// 跳过自身的位置

if (ii == i && jj == j) continue;

// 如果找到相等的值

if (arr[i][j] == arr[ii][jj]) {

// 如果res尚未更新过，则更新为当前距离

int distance = Math.abs(i - ii) + Math.abs(j - jj);

if (res == -1) {

res = distance;

} else {

// 否则更新为最小距离

res = Math.min(res, distance);

}

}

}

}

// 将结果加入当前行的结果数组

result[i][j] = res;

}

}

// 输出结果

System.out.println(Arrays.deepToString(result));

}

}



# 读取二维数组的行数

n = int(input())

# 读取二维数组的列数

m = int(input())

# 初始化二维数组

arr = []

# 循环读取每一行的输入，并将其转换为整数数组

for i in range(n):

arr.append([int(\_) for \_ in input().split()])

# 初始化结果数组

mp1 = []

# 遍历二维数组的每个元素

for i in range(n):

# 初始化当前行的结果数组

mp2 = []

for j in range(m):

# 设置默认结果为-1

res = -1

# 遍历二维数组的每个元素，寻找相等的值

for ii in range(n):

for jj in range(m):

# 跳过自身的位置

if ii == i and jj == j:

continue

# 如果找到相等的值

if arr[i][j] == arr[ii][jj]:

# 如果res尚未更新过，则更新为当前距离

if res == -1:

res = abs(i - ii) + abs(j - jj)

# 否则更新为最小距离

else:

res = min(res, abs(i - ii) + abs(j - jj))

# 将结果加入当前行的结果数组

mp2.append(res)

# 将当前行的结果数组加入总结果数组

mp1.append(mp2)

# 转换结果数组为字符串并输出

ans = str(mp1)

print(ans)



#include <iostream>

#include <vector>

#include <cmath>

#include <climits>

using namespace std;

int main() {

// 读取二维数组的行数

int n, m;

cin >> n >> m;

// 初始化二维数组

vector<vector<int>> arr(n, vector<int>(m));

// 循环读取每一行的输入，并将其转换为整数数组

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

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

cin >> arr[i][j];

}

}

// 初始化结果数组

vector<vector<int>> result(n, vector<int>(m, -1));

// 遍历二维数组的每个元素

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

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

// 设置默认结果为-1

int res = -1;

// 遍历二维数组的每个元素，寻找相等的值

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

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

// 跳过自身的位置

if (ii == i && jj == j) continue;

// 如果找到相等的值

if (arr[i][j] == arr[ii][jj]) {

// 计算曼哈顿距离

int distance = abs(i - ii) + abs(j - jj);

// 如果res尚未更新过，则更新为当前距离

if (res == -1) {

res = distance;

} else {

// 否则更新为最小距离

res = min(res, distance);

}

}

}

}

// 将结果加入当前行的结果数组

result[i][j] = res;

}

}

// 输出结果

cout << "[";

// 遍历结果数组，打印结果

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

cout << "[";

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

cout << result[i][j];

if (j < m - 1) cout << ", ";

}

cout << "]";

if (i < n - 1) cout << ", ";

}

cout << "]" << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int main() {

// 读取二维数组的行数和列数

int n, m;

scanf("%d %d", &n, &m);

// 初始化二维数组

int arr[n][m];

// 循环读取每一行的输入，并将其转换为整数

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

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

scanf("%d", &arr[i][j]);

}

}

// 初始化结果数组

int result[n][m];

// 遍历二维数组的每个元素

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

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

// 设置默认结果为 -1

int res = -1;

// 遍历二维数组的每个元素，寻找相等的值

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

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

// 跳过自身的位置

if (ii == i && jj == j) continue;

// 如果找到相等的值

if (arr[i][j] == arr[ii][jj]) {

// 计算曼哈顿距离

int distance = abs(i - ii) + abs(j - jj);

// 如果 res 尚未更新过，则更新为当前距离

if (res == -1) {

res = distance;

} else {

// 否则更新为最小距离

if (distance < res) {

res = distance;

}

}

}

}

}

// 将结果加入当前行的结果数组

result[i][j] = res;

}

}

// 输出结果

printf("[");

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

printf("[");

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

printf("%d", result[i][j]);

if (j < m - 1) printf(", ");

}

printf("]");

if (i < n - 1) printf(", ");

}

printf("]\n");

return 0;

}



const readline = require("readline");

// 创建接口以读取标准输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = [];

// 监听标准输入数据

rl.on("line", (line) => {

input.push(line.trim());

if (input.length === parseInt(input[0]) + 2) {

// 当读取的行数达到期望值时，开始处理

rl.close();

}

});

// 当读取完所有输入时执行

rl.on("close", () => {

// 读取二维数组的行数和列数

const n = parseInt(input[0]);

const m = parseInt(input[1]);

// 初始化二维数组

let arr = [];

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

arr.push(input[i + 2].split(" ").map(Number));

}

// 初始化结果数组

let result = Array.from({ length: n }, () => Array(m).fill(-1));

// 遍历二维数组的每个元素

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

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

let res = -1;

// 遍历二维数组的每个元素，寻找相等的值

for (let ii = 0; ii < n; ii++) {

for (let jj = 0; jj < m; jj++) {

// 跳过自身的位置

if (ii === i && jj === j) continue;

// 如果找到相等的值

if (arr[i][j] === arr[ii][jj]) {

// 计算曼哈顿距离

let distance = Math.abs(i - ii) + Math.abs(j - jj);

// 如果 res 尚未更新过，则更新为当前距离

if (res === -1) {

res = distance;

} else {

// 否则更新为最小距离

res = Math.min(res, distance);

}

}

}

}

// 将结果加入当前行的结果数组

result[i][j] = res;

}

}

// 输出结果

console.log(JSON.stringify(result));

});



package main

import (

"bufio"

"fmt"

"math"

"os"

"strconv"

"strings"

)

func main() {

// 创建一个新的读取器，从标准输入读取

reader := bufio.NewReader(os.Stdin)

// 读取二维数组的行数

text, \_ := reader.ReadString('\n')

n, \_ := strconv.Atoi(strings.TrimSpace(text))

// 读取二维数组的列数

text, \_ = reader.ReadString('\n')

m, \_ := strconv.Atoi(strings.TrimSpace(text))

// 初始化二维数组

arr := make([][]int, n)

for i := 0; i < n; i++ {

arr[i] = make([]int, m)

}

// 循环读取每一行的输入，并将其转换为整数数组

for i := 0; i < n; i++ {

text, \_ := reader.ReadString('\n')

numbers := strings.Fields(text)

for j := 0; j < m; j++ {

arr[i][j], \_ = strconv.Atoi(numbers[j])

}

}

// 初始化结果数组

result := make([][]int, n)

for i := range result {

result[i] = make([]int, m)

for j := range result[i] {

result[i][j] = -1

}

}

// 遍历二维数组的每个元素

for i := 0; i < n; i++ {

for j := 0; j < m; j++ {

res := -1

// 遍历二维数组的每个元素，寻找相等的值

for ii := 0; ii < n; ii++ {

for jj := 0; jj < m; jj++ {

// 跳过自身的位置

if ii == i && jj == j {

continue

}

// 如果找到相等的值

if arr[i][j] == arr[ii][jj] {

// 计算曼哈顿距离

distance := int(math.Abs(float64(i-ii)) + math.Abs(float64(j-jj)))

// 如果 res 尚未更新过，则更新为当前距离

if res == -1 {

res = distance

} else {

// 否则更新为最小距离

res = min(res, distance)

}

}

}

}

// 将结果加入当前行的结果数组

result[i][j] = res

}

}

// 打印结果

fmt.Print("[")

for i := 0; i < n; i++ {

if i > 0 {

fmt.Print(", ")

}

fmt.Print("[")

for j := 0; j < m; j++ {

if j > 0 {

fmt.Print(", ")

}

fmt.Print(result[i][j])

}

fmt.Print("]")

}

fmt.Print("]\n")

}

// 辅助函数：返回两个整数中的最小值

func min(a, b int) int {

if a < b {

return a

}

return b

}