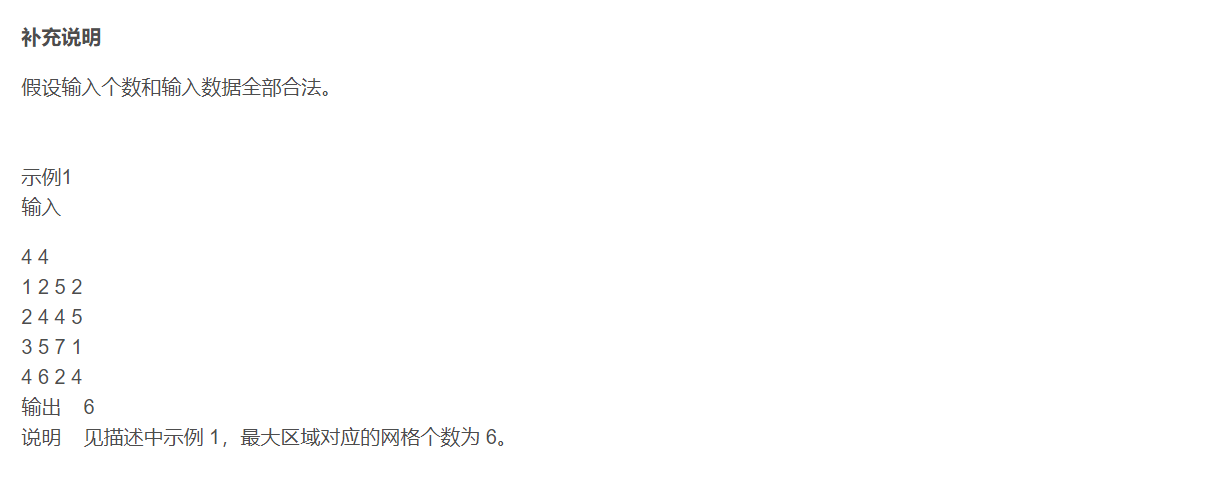
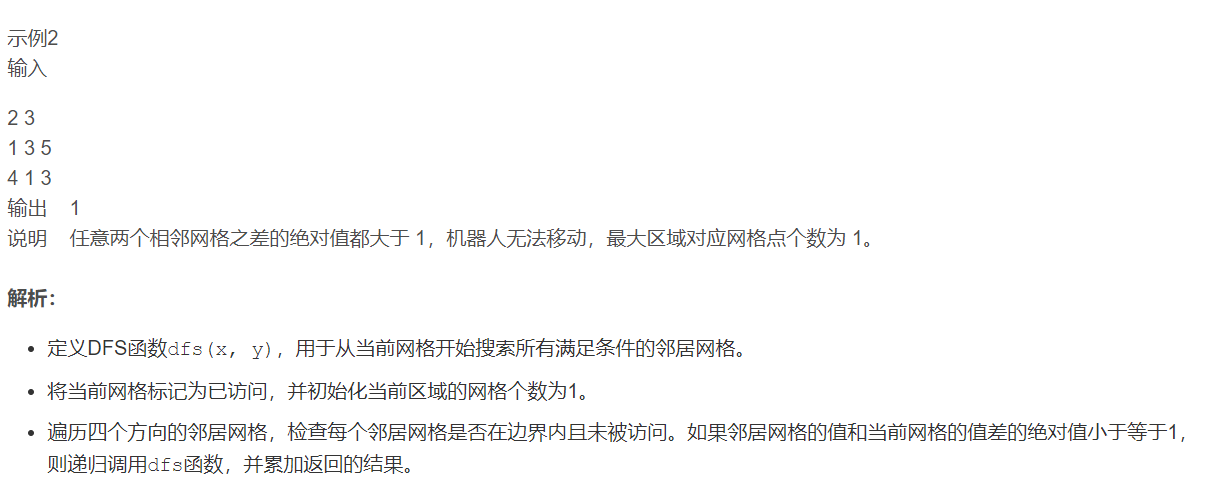
# **E卷-机器人活动区域[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**











import java.util.Scanner;

public class Main {

static int n, m;

static int[][] g;

static boolean[][] vis;

static int[] dx = {-1, 1, 0, 0};

static int[] dy = {0, 0, -1, 1};

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

m = sc.nextInt();

g = new int[n][m];

vis = new boolean[n][m];

// 读取网格数据

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

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

g[i][j] = sc.nextInt();

}

}

int res = 0;

// 遍历整个网格

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

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

if (!vis[i][j]) {

res = Math.max(res, dfs(i, j));

}

}

}

// 输出结果

System.out.println(res);

}

// 深度优先搜索函数

private static int dfs(int x, int y) {

vis[x][y] = true;

int now = 1;

// 遍历四个方向的邻居网格

for (int i = 0; i < 4; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

// 检查邻居网格是否在边界内且未被访问

if (nx >= 0 && nx < n && ny >= 0 && ny < m && !vis[nx][ny]) {

// 检查邻居网格和当前网格的值差是否小于等于1

if (Math.abs(g[nx][ny] - g[x][y]) <= 1) {

now += dfs(nx, ny);

}

}

}

return now;

}

}



import sys

# 读取输入的网格大小，n表示行数，m表示列数

n, m = map(int, input().split())

# 读取网格数据并存储到二维列表g中

g = []

for \_ in range(n):

g.append(list(map(int, input().split(" "))))

# 确认网格的行数n和列数m，确保数据一致性

n, m = len(g), len(g[0])

# 定义一个二维列表vis用于标记某个网格是否已被访问，初始全为False

vis = [[False] \* m for \_ in range(n)]

# 结果存储，初始化为0

res = 0

# 定义移动的四个方向（上、下、左、右）

d = [(-1, 0), (1, 0), (0, -1), (0, 1)]

# 深度优先搜索函数，参数为当前网格的坐标（x, y）

def dfs(x, y):

# 将当前网格标记为已访问

vis[x][y] = True

# 初始化当前区域的网格个数为1

now = 1

# 遍历四个方向的邻居网格

for dx, dy in d:

nx, ny = x + dx, y + dy

# 检查邻居网格是否在边界内且未被访问

if 0 <= nx < n and 0 <= ny < m and not vis[nx][ny]:

# 检查邻居网格和当前网格的值差是否小于等于1

if abs(g[nx][ny] - g[x][y]) <= 1:

# 如果条件满足，进行递归搜索并累加结果

now += dfs(nx, ny)

return now

# 遍历整个网格以寻找未访问的起点

for i in range(n):

for j in range(m):

# 若当前网格未被访问，则从该点开始深度优先搜索

if not vis[i][j]:

# 更新结果为最大值

res = max(res, dfs(i, j))

# 输出结果，即机器人可活动的最大区域的网格个数

print(res)



#include <iostream>

#include <vector>

#include <algorithm>

#include <cmath>

using namespace std;

int n, m;

vector<vector<int>> g;

vector<vector<bool>> vis;

int dx[4] = {-1, 1, 0, 0};

int dy[4] = {0, 0, -1, 1};

int dfs(int x, int y) {

vis[x][y] = true;

int now = 1;

// 遍历四个方向的邻居网格

for (int i = 0; i < 4; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

// 检查邻居网格是否在边界内且未被访问

if (nx >= 0 && nx < n && ny >= 0 && ny < m && !vis[nx][ny]) {

// 检查邻居网格和当前网格的值差是否小于等于1

if (abs(g[nx][ny] - g[x][y]) <= 1) {

now += dfs(nx, ny);

}

}

}

return now;

}

int main() {

cin >> n >> m;

g.resize(n, vector<int>(m));

vis.resize(n, vector<bool>(m, false));

// 读取网格数据

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

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

cin >> g[i][j];

}

}

int res = 0;

// 遍历整个网格

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

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

if (!vis[i][j]) {

res = max(res, dfs(i, j));

}

}

}

// 输出结果

cout << res << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define MAXN 150

int n, m;

int g[MAXN][MAXN];

int vis[MAXN][MAXN];

int dx[4] = {-1, 1, 0, 0};

int dy[4] = {0, 0, -1, 1};

// 深度优先搜索函数

int dfs(int x, int y) {

vis[x][y] = 1;

int now = 1;

// 遍历四个方向的邻居网格

for (int i = 0; i < 4; i++) {

int nx = x + dx[i];

int ny = y + dy[i];

// 检查邻居网格是否在边界内且未被访问

if (nx >= 0 && nx < n && ny >= 0 && ny < m && !vis[nx][ny]) {

// 检查邻居网格和当前网格的值差是否小于等于1

if (abs(g[nx][ny] - g[x][y]) <= 1) {

now += dfs(nx, ny);

}

}

}

return now;

}

int main() {

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

// 读取网格数据

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

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

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

vis[i][j] = 0; // 初始化访问数组

}

}

int res = 0;

// 遍历整个网格

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

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

if (!vis[i][j]) {

int c = dfs(i, j);

res = res > c ? res : c;

}

}

}

// 输出结果

printf("%d\n", res);

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = [];

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

input.push(line);

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

let [n, m] = input[0].split(" ").map(Number);

let g = [];

for (let i = 1; i <= n; i++) {

g.push(input[i].split(" ").map(Number));

}

let vis = Array.from({ length: n }, () => Array(m).fill(false));

let dx = [-1, 1, 0, 0];

let dy = [0, 0, -1, 1];

function dfs(x, y) {

vis[x][y] = true;

let now = 1;

// 遍历四个方向的邻居网格

for (let i = 0; i < 4; i++) {

let nx = x + dx[i];

let ny = y + dy[i];

// 检查邻居网格是否在边界内且未被访问

if (nx >= 0 && nx < n && ny >= 0 && ny < m && !vis[nx][ny]) {

// 检查邻居网格和当前网格的值差是否小于等于1

if (Math.abs(g[nx][ny] - g[x][y]) <= 1) {

now += dfs(nx, ny);

}

}

}

return now;

}

let res = 0;

// 遍历整个网格

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

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

if (!vis[i][j]) {

res = Math.max(res, dfs(i, j));

}

}

}

// 输出结果

console.log(res);

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// 初始化变量，g表示网格，vis表示是否访问的二维数组

var (

n, m int

g [][]int

vis [][]bool

// 定义四个方向：上、下、左、右

dx = []int{-1, 1, 0, 0}

dy = []int{0, 0, -1, 1}

)

// 深度优先搜索函数

func dfs(x, y int) int {

// 将当前网格标记为已访问

vis[x][y] = true

now := 1

// 遍历四个方向的邻居网格

for i := 0; i < 4; i++ {

nx, ny := x+dx[i], y+dy[i]

// 检查邻居网格是否在边界内且未被访问

if nx >= 0 && nx < n && ny >= 0 && ny < m && !vis[nx][ny] {

// 检查邻居网格和当前网格的值差是否小于等于1

if abs(g[nx][ny]-g[x][y]) <= 1 {

// 如果条件满足，进行递归搜索并累加结果

now += dfs(nx, ny)

}

}

}

return now

}

// 计算绝对值的函数

func abs(a int) int {

if a < 0 {

return -a

}

return a

}

func main() {

reader := bufio.NewReader(os.Stdin)

// 读取第一行输入，获取网格的行数和列数

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

firstLine = strings.TrimSpace(firstLine)

parts := strings.Split(firstLine, " ")

n, \_ = strconv.Atoi(parts[0])

m, \_ = strconv.Atoi(parts[1])

// 初始化网格和访问数组

g = make([][]int, n)

vis = make([][]bool, n)

// 读取网格数据

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

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

line = strings.TrimSpace(line)

parts := strings.Split(line, " ")

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

vis[i] = make([]bool, m)

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

g[i][j], \_ = strconv.Atoi(parts[j])

}

}

// 初始化结果变量

res := 0

// 遍历整个网格以寻找未访问的起点

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

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

if !vis[i][j] {

// 每次调用dfs后，更新最大结果

res = max(res, dfs(i, j))

}

}

}

// 输出结果

fmt.Println(res)

}

// 计算最大值的函数

func max(a, b int) int {

if a > b {

return a

}

return b

}