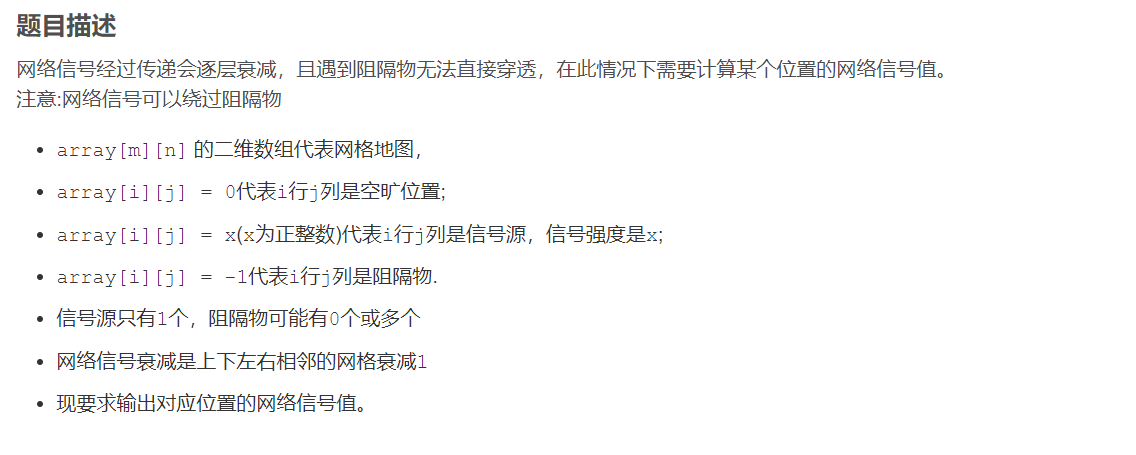
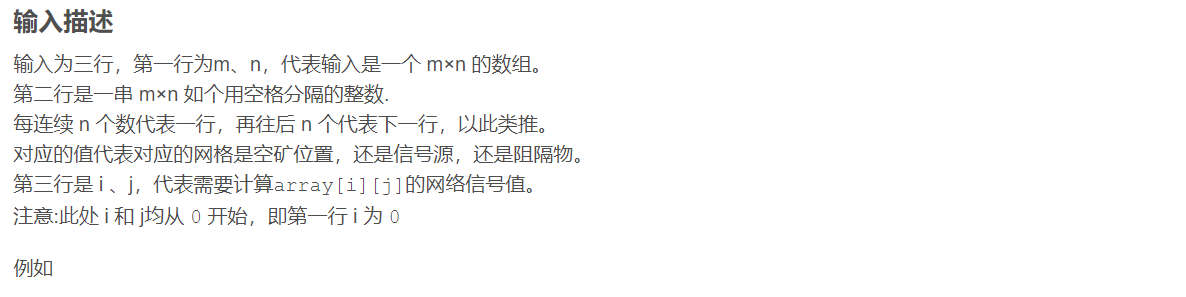
# **E卷-计算网络信号[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go ）**

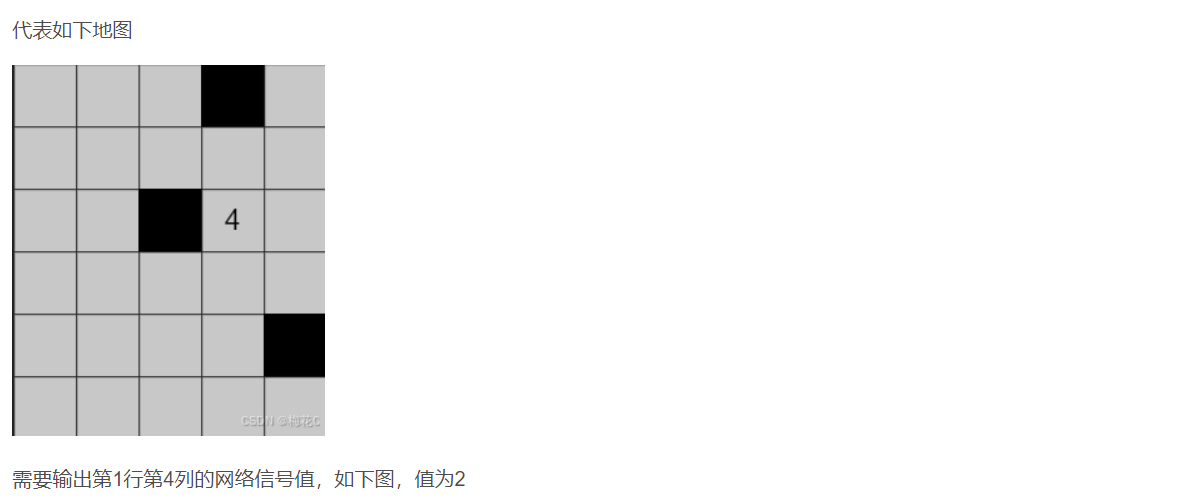




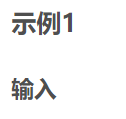
6 5

0 0 0 -1 0 0 0 0 0 0 0 0 -1 4 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0

1 4







6 5

0 0 0 -1 0 0 0 0 0 0 0 0 -1 4 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0

1 4



2



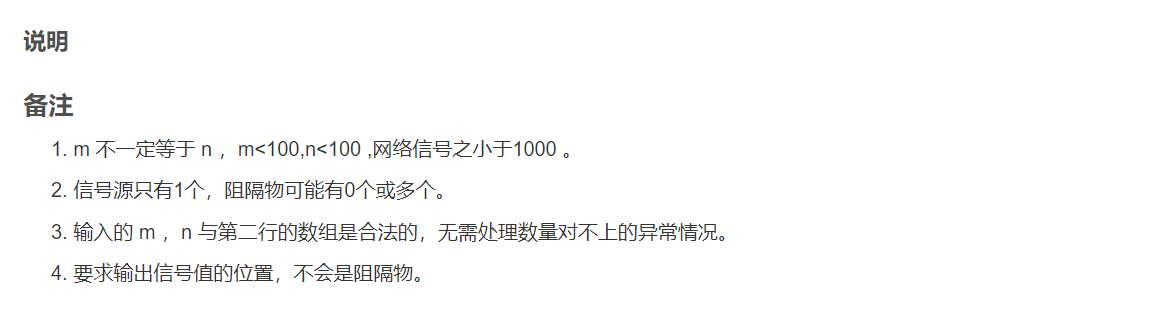
6 5

0 0 0 -1 0 0 0 0 0 0 0 0 -1 4 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0

2 1



0





import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// 读取行数和列数

int n = scanner.nextInt();

int m = scanner.nextInt();

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

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

Queue<int[]> q = new LinkedList<>(); // 创建队列用于BFS

int index = 0; // 初始化index用于遍历输入数据

// 读取并初始化网格数组

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

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

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

if (arr[i][j] > 0) {

q.add(new int[] {i, j}); // 信号源加入队列

dis[i][j] = arr[i][j]; // 初始化信号值

}

}

}

// 读取目标位置

int ex = scanner.nextInt();

int ey = scanner.nextInt();

// 定义四个方向

int[][] dir = new int[][] {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

// 使用BFS来传播信号

while (!q.isEmpty()) {

int[] pos = q.poll();

int x = pos[0], y = pos[1];

for (int[] d : dir) {

int nx = x + d[0];

int ny = y + d[1];

if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue; // 检查边界

if (arr[nx][ny] == -1) continue; // 跳过阻隔物

int ndis = dis[x][y] - 1;

if (ndis > 0 && dis[nx][ny] < ndis) { // 更新信号值

dis[nx][ny] = ndis;

q.add(new int[] {nx, ny});

}

}

}

// 输出结果

System.out.println(dis[ex][ey]);

}

}



from collections import deque # 引入双端队列模块

# 读取第一个输入，代表网格的行数(m)和列数(n)

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

# 读取第二行输入，表示网格的所有元素，存储到一个一维列表中

tmp = list(map(int, input().split()))

p = 0 # 初始化一维列表的当前下标

q = deque() # 创建一个空的双端队列用于BFS

dis = [[0] \* m for \_ in range(n)] # 创建一个二维列表表示每个单元格的信号值，初始化为0

arr = [[0] \* m for \_ in range(n)] # 创建一个二维列表来存储输入的网格

# 将输入的一维列表转换为二维列表

for i in range(n):

for j in range(m):

arr[i][j] = tmp[p]

p += 1

# 如果当前单元格是信号源，将其初始信号值赋给对应的dis元素，并加入队列

if arr[i][j] > 0:

q.append((i, j))

dis[i][j] = arr[i][j]

# 读取第三行输入，提取需要查询的单元格坐标 (ex, ey)

ex, ey = map(int, input().split())

# 定义上下左右四个方向的坐标偏移

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

# 使用BFS来传播信号

while len(q):

x, y = q.popleft() # 从队列中取出当前处理的单元格坐标

for dx, dy in dir: # 遍历四个方向

# 计算相邻单元格的坐标

nx, ny = x + dx, y + dy

# 检查坐标是否超出网格范围

if nx < 0 or nx >= n or ny < 0 or ny >= m:

continue

# 跳过阻隔物

if arr[nx][ny] == -1:

continue

# 计算相邻单元格的信号值

ndis = dis[x][y] - 1

# 如果信号值大于0，更新相邻单元格的信号值并将其加入队列

if ndis:

# 如果该单元格尚未被访问过或当前路径的信号值更强，更新信号值

if dis[nx][ny] < ndis:

dis[nx][ny] = ndis

q.append((nx, ny))

# 输出指定位置的网络信号值

print(dis[ex][ey])



#include <iostream>

#include <queue>

#include <vector>

using namespace std;

int main() {

int n, m;

cin >> n >> m;

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

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

queue<pair<int, int>> q; // 创建队列用于BFS

// 读取并初始化网格数组

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

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

cin >> arr[i][j];

if (arr[i][j] > 0) {

q.push({i, j}); // 信号源加入队列

dis[i][j] = arr[i][j]; // 初始化信号值

}

}

}

// 读取目标位置

int ex, ey;

cin >> ex >> ey;

// 定义四个方向

int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

// 使用BFS来传播信号

while (!q.empty()) {

auto [x, y] = q.front();

q.pop();

for (const auto& d : dir) {

int nx = x + d[0];

int ny = y + d[1];

if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue; // 检查边界

if (arr[nx][ny] == -1) continue; // 跳过阻隔物

int ndis = dis[x][y] - 1;

if (ndis > 0 && dis[nx][ny] < ndis) { // 更新信号值

dis[nx][ny] = ndis;

q.push({nx, ny});

}

}

}

// 输出结果

cout << dis[ex][ey] << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct {

int x, y;

} Node;

int arr[MAX][MAX];

int dis[MAX][MAX];

Node queue[MAX \* MAX];

int front, rear;

void enqueue(int x, int y) {

queue[rear].x = x;

queue[rear].y = y;

rear++;

}

Node dequeue() {

return queue[front++];

}

int main() {

int n, m;

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

front = rear = 0;

// 初始化网格数组

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

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

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

if (arr[i][j] > 0) {

enqueue(i, j); // 信号源加入队列

dis[i][j] = arr[i][j]; // 初始化信号值

}

}

}

// 读取目标位置

int ex, ey;

scanf("%d %d", &ex, &ey);

// 定义四个方向

int dir[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

// 使用BFS来传播信号

while (front < rear) {

Node node = dequeue();

int x = node.x;

int y = node.y;

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

int nx = x + dir[d][0];

int ny = y + dir[d][1];

if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue; // 检查边界

if (arr[nx][ny] == -1) continue; // 跳过阻隔物

int ndis = dis[x][y] - 1;

if (ndis > 0 && dis[nx][ny] < ndis) { // 更新信号值

dis[nx][ny] = ndis;

enqueue(nx, ny);

}

}

}

// 输出结果

printf("%d\n", dis[ex][ey]);

return 0;

}



const readline = require("readline");

// 创建Readline接口以读取用户输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = [];

let lineIndex = 0;

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

input.push(line);

if (++lineIndex === 3) {

rl.close();

}

});

// 关闭Readline接口后处理输入数据

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

// 解析第一行输入，代表网格的行数(m)和列数(n)

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

// 解析第二行输入，表示网格的所有元素

const tmp = input[1].split(" ").map(Number);

const arr = Array.from({ length: n }, () => Array(m).fill(0));

const dis = Array.from({ length: n }, () => Array(m).fill(0));

const q = []; // 创建队列用于BFS

let p = 0; // 初始化一维列表的当前下标

// 将一维数据转换为二维数组并初始化信号源的信号值

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

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

arr[i][j] = tmp[p++];

if (arr[i][j] > 0) {

q.push([i, j]); // 信号源加入队列

dis[i][j] = arr[i][j]; // 初始化信号值

}

}

}

// 读取第三行输入，提取目标单元格地址 (ex, ey)

const [ex, ey] = input[2].split(" ").map(Number);

// 定义四个方向

const dir = [

[-1, 0],

[1, 0],

[0, -1],

[0, 1],

];

// 使用BFS来传播信号

while (q.length > 0) {

const [x, y] = q.shift(); // 从队列中取出当前处理的单元格坐标

for (const [dx, dy] of dir) {

const nx = x + dx;

const ny = y + dy;

if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue; // 检查边界

if (arr[nx][ny] === -1) continue; // 跳过阻隔物

const ndis = dis[x][y] - 1;

if (ndis > dis[nx][ny]) {

// 更新信号值

dis[nx][ny] = ndis;

if (ndis > 0) {

q.push([nx, ny]);

}

}

}

}

// 输出结果

console.log(dis[ex][ey]);

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

// 定义四个方向

var dir = [][2]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}

type P struct {

x, y int

}

func main() {

// 创建Scanner从标准输入读取数据

reader := bufio.NewReader(os.Stdin)

// 读取行数和列数

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

line = strings.TrimSpace(line)

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

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

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

// 初始化网格和信号值数组

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

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

for i := range arr {

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

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

}

// 读取网格数据

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

line = strings.TrimSpace(line)

parts = strings.Split(line, " ")

queue := []P{}

idx := 0

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

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

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

if arr[i][j] > 0 {

queue = append(queue, P{i, j}) // 信号源加入队列

dis[i][j] = arr[i][j] // 初始化信号值

}

idx++

}

}

// 读取目标位置

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

line = strings.TrimSpace(line)

parts = strings.Split(line, " ")

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

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

// 使用BFS来传播信号

for len(queue) > 0 {

p := queue[0]

queue = queue[1:]

for \_, d := range dir {

nx, ny := p.x+d[0], p.y+d[1]

if nx < 0 || nx >= n || ny < 0 || ny >= m {

continue // 检查边界

}

if arr[nx][ny] == -1 {

continue // 跳过阻隔物

}

ndis := dis[p.x][p.y] - 1

if ndis > dis[nx][ny] {

dis[nx][ny] = ndis

if ndis > 0 {

queue = append(queue, P{nx, ny})

}

}

}

}

// 输出结果

fmt.Println(dis[ex][ey])

}