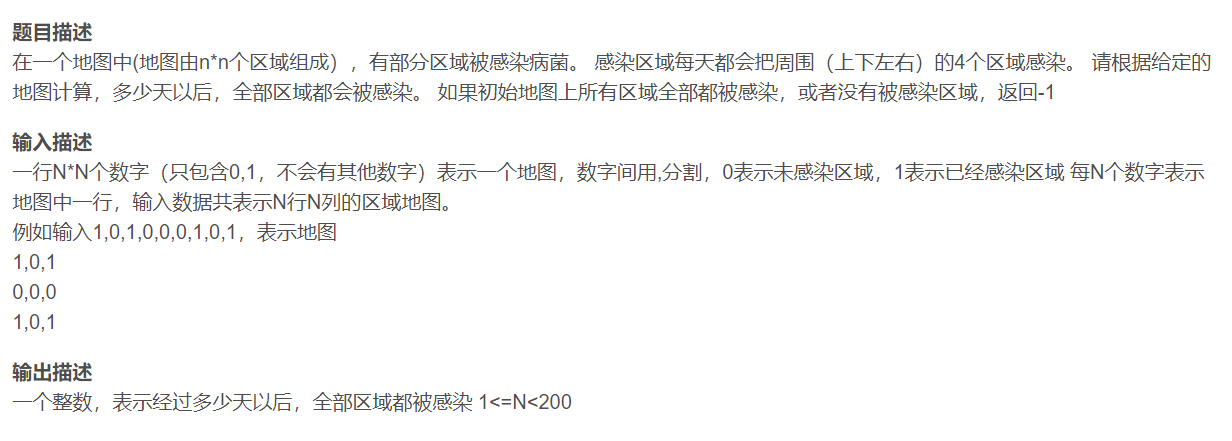
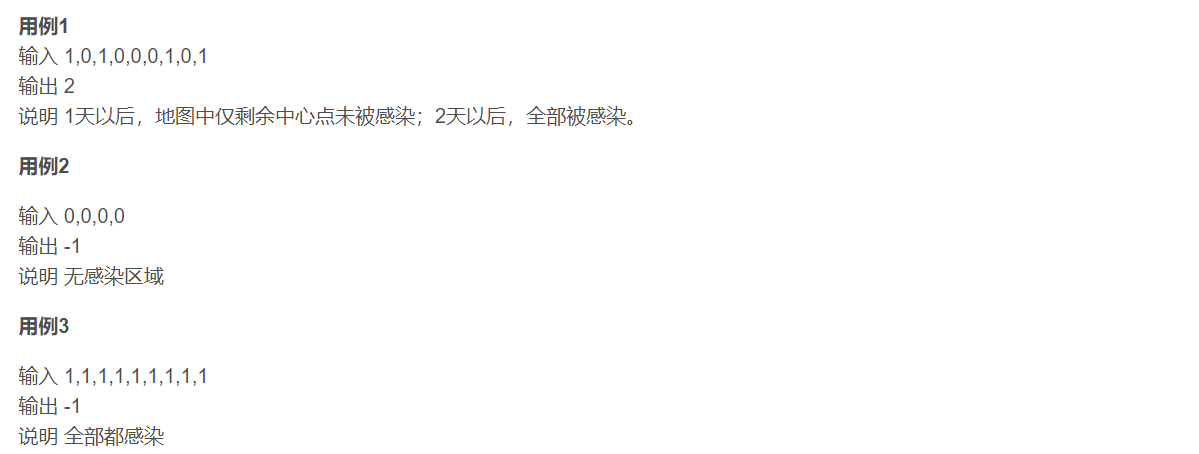
# **E卷-计算疫情扩散时间[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**







import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String[] s = scanner.nextLine().split(",");

int n = (int) Math.sqrt(s.length);

int[][] mp = new int[n][n];

int id = 0;

int cnt0 = 0;

Queue<int[]> queue = new LinkedList<>();

// 填充地图并统计未感染区域（0）的数量

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

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

mp[i][j] = Integer.parseInt(s[id]);

id++;

if (mp[i][j] == 0) {

cnt0++;

} else {

queue.add(new int[] {i, j, 0});

}

}

}

// 如果没有未感染区域或者全是未感染区域，返回-1

if (cnt0 == 0 || cnt0 == n \* n) {

System.out.println(-1);

return;

}

// 方向数组，上下左右四个方向

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

int maxDays = 0;

// 广度优先搜索（BFS）遍历感染扩展的过程

while (!queue.isEmpty()) {

int[] current = queue.poll();

int x = current[0];

int y = current[1];

int days = current[2];

maxDays = Math.max(maxDays, days);

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

int xx = x + dir[i];

int yy = y + dir[i + 1];

if (xx >= 0 && xx < n && yy >= 0 && yy < n && mp[xx][yy] == 0) {

queue.add(new int[] {xx, yy, days + 1});

mp[xx][yy] = 1;

}

}

}

// 输出最终的最大天数，即地图全部感染所需天数

System.out.println(maxDays);

}

}



# 输入地图，以逗号分隔

s = input().split(",")

import math

# 计算地图的维度n

n = int(math.sqrt(len(s)))

# 初始化地图mp为n\*n的0矩阵

mp = [[0 for \_ in range(n)] for \_ in range(n)]

id = 0

cnt0 = 0

from collections import deque

# 初始化队列

que = deque()

# 填充地图并统计未感染区域（0）的数量

for i in range(n):

for j in range(n):

mp[i][j] = s[id]

id += 1

if mp[i][j] == "0":

cnt0 += 1

else:

que.append((i, j, 0))

# 如果没有未感染区域或者全是未感染区域，返回-1

if cnt0 == 0 or cnt0 == n \* n:

print(-1)

exit(0)

# 方向数组，上下左右四个方向

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

maxx = 0

# 广度优先搜索（BFS）遍历感染扩展的过程

while len(que) > 0:

(x, y, d) = que.popleft()

maxx = max(maxx, d)

for i in range(4):

xx, yy = x + dir[i], y + dir[i + 1]

if 0 <= xx < n and 0 <= yy < n and mp[xx][yy] == "0":

que.append((xx, yy, d + 1))

mp[xx][yy] = "1"

# 输出最终的最大天数，即地图全部感染所需天数

print(maxx)



#include <iostream>

#include <vector>

#include <queue>

#include <cmath>

using namespace std;

int main() {

string line;

getline(cin, line);

vector<int> s;

size\_t start = 0, end = 0;

// 以逗号分隔输入字符串并转换成整型

while ((end = line.find(',', start)) != string::npos) {

s.push\_back(stoi(line.substr(start, end - start)));

start = end + 1;

}

s.push\_back(stoi(line.substr(start)));

// 计算地图的维度n

int n = sqrt(s.size());

vector<vector<int>> mp(n, vector<int>(n));

int id = 0, cnt0 = 0;

queue<pair<int, pair<int, int>>> que; // 存储坐标及当前天数

// 填充地图并统计未感染区域（0）的数量

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

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

mp[i][j] = s[id++];

if (mp[i][j] == 0) {

++cnt0;

} else {

que.push({i, {j, 0}});

}

}

}

// 如果没有未感染区域或者全是未感染区域，返回-1

if (cnt0 == 0 || cnt0 == n \* n) {

cout << -1 << endl;

return 0;

}

// 方向数组，上下左右四个方向

vector<int> dir = {0, 1, 0, -1, 0};

int maxDays = 0;

// 广度优先搜索（BFS）遍历感染扩展的过程

while (!que.empty()) {

auto [x, y\_d] = que.front();

auto [y, days] = y\_d;

que.pop();

maxDays = max(maxDays, days);

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

int xx = x + dir[i], yy = y + dir[i + 1];

if (xx >= 0 && xx < n && yy >= 0 && yy < n && mp[xx][yy] == 0) {

que.push({xx, {yy, days + 1}});

mp[xx][yy] = 1;

}

}

}

// 输出最终的最大天数，即地图全部感染所需天数

cout << maxDays << endl;

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

typedef struct {

int x, y, days;

} Node;

int main() {

char line[40001];

fgets(line, sizeof(line), stdin);

int s[40000], s\_size = 0;

char\* token = strtok(line, ",");

while (token != NULL) {

s[s\_size++] = atoi(token);

token = strtok(NULL, ",");

}

int n = sqrt(s\_size);

int mp[200][200];

int id = 0, cnt0 = 0;

Node queue[40000];

int front = 0, rear = 0;

// 填充地图并统计未感染区域（0）的数量

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

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

mp[i][j] = s[id++];

if (mp[i][j] == 0) {

cnt0++;

} else {

Node node = {i, j, 0};

queue[rear++] = node;

}

}

}

// 如果没有未感染区域或者全是未感染区域，返回-1

if (cnt0 == 0 || cnt0 == n \* n) {

printf("-1\n");

return 0;

}

// 方向数组，上下左右四个方向

int dir[5] = {0, 1, 0, -1, 0};

int maxDays = 0;

// 广度优先搜索（BFS）遍历感染扩展的过程

while (front < rear) {

Node node = queue[front++];

int x = node.x, y = node.y, days = node.days;

maxDays = maxDays > days ? maxDays : days;

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

int xx = x + dir[i], yy = y + dir[i + 1];

if (xx >= 0 && xx < n && yy >= 0 && yy < n && mp[xx][yy] == 0) {

Node adjNode = {xx, yy, days + 1};

queue[rear++] = adjNode;

mp[xx][yy] = 1;

}

}

}

// 输出最终的最大天数，即地图全部感染所需天数

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

return 0;

}



const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

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

const s = line.trim().split(",").map(Number);

const n = Math.sqrt(s.length);

const mp = [];

let cnt0 = 0;

const queue = [];

// 填充地图并统计未感染区域（0）的数量

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

mp[i] = [];

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

mp[i][j] = s[i \* n + j];

if (mp[i][j] === 0) {

cnt0++;

} else {

queue.push([i, j, 0]);

}

}

}

// 如果没有未感染区域或者全是未感染区域，返回-1

if (cnt0 === 0 || cnt0 === n \* n) {

console.log(-1);

return;

}

// 方向数组，上下左右四个方向

const dir = [0, 1, 0, -1, 0];

let maxDays = 0;

// 广度优先搜索（BFS）遍历感染扩展的过程

while (queue.length > 0) {

const [x, y, days] = queue.shift();

maxDays = Math.max(maxDays, days);

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

const xx = x + dir[i];

const yy = y + dir[i + 1];

if (xx >= 0 && xx < n && yy >= 0 && yy < n && mp[xx][yy] === 0) {

queue.push([xx, yy, days + 1]);

mp[xx][yy] = 1;

}

}

}

// 输出最终的最大天数，即地图全部感染所需天数

console.log(maxDays);

});



package main

import (

"bufio"

"fmt"

"math"

"os"

"strconv"

"strings"

)

type Node struct {

x, y, days int

}

func main() {

reader := bufio.NewReader(os.Stdin)

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

line = strings.TrimSpace(line)

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

s := make([]int, len(parts))

for i, part := range parts {

s[i], \_ = strconv.Atoi(part)

}

n := int(math.Sqrt(float64(len(s))))

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

cnt0 := 0

queue := []Node{}

id := 0

// 填充地图并统计未感染区域（0）的数量

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

mp[i] = make([]int, n)

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

mp[i][j] = s[id]

id++

if mp[i][j] == 0 {

cnt0++

} else {

queue = append(queue, Node{i, j, 0})

}

}

}

// 如果没有未感染区域或者全是未感染区域，返回-1

if cnt0 == 0 || cnt0 == n\*n {

fmt.Println(-1)

return

}

// 方向数组，上下左右四个方向

dir := []int{0, 1, 0, -1, 0}

maxDays := 0

// 广度优先搜索（BFS）遍历感染扩展的过程

for len(queue) > 0 {

node := queue[0]

queue = queue[1:]

x, y, days := node.x, node.y, node.days

maxDays = int(math.Max(float64(maxDays), float64(days)))

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

xx, yy := x+dir[i], y+dir[i+1]

if xx >= 0 && xx < n && yy >= 0 && yy < n && mp[xx][yy] == 0 {

queue = append(queue, Node{xx, yy, days + 1})

mp[xx][yy] = 1

}

}

}

// 输出最终的最大天数，即地图全部感染所需天数

fmt.Println(maxDays)

}