59、计算疫情扩散时间，考点 or 实现——图论/图的多源 BFS

题目描述

在一个地图中(地图由n*n个区域组成），有部分区域被感染病菌。 感染区域每天都会把周围（上下左右）的4个区域感染。 请根据给定的地图计算，多少天以后，全部区域都会被感染。 如果初始地图上所有区域全部都被感染，或者没有被感染区域，返回-1

输入描述

一行N*N个数字（只包含0,1，不会有其他数字）表示一个地图，数字间用,分割，0表示未感染区域，1表示已经感染区域 每N个数字表示地图中一行，输入数据共表示N行N列的区域地图。

例如输入1,0,1,0,0,0,1,0,1，表示地图

1,0,1
0,0,0
1,0,1

输出描述

一个整数，表示经过多少天以后，全部区域都被感染 1<=N<200

用例

输入	1,0,1,0,0,0,1,0,1
输出	2
说明	1天以后，地图中仅剩余中心点未被感染；2天以后，全部被感染。

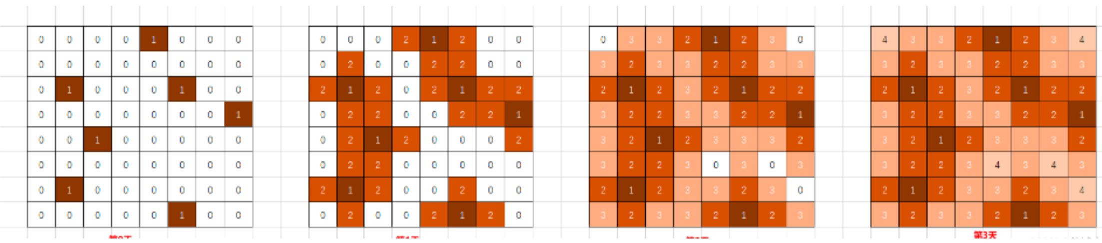
输入	0,0,0,0
输出	-1
说明	无感染区域

输入	1,1,1,1,1,1,1,1,1
输出	-1
说明	全部都感染

题目解析

自测用例：0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0

一共8x8个区域



感染区传播过程如上图所示。

本题其实就是图结构的多源 BFS。

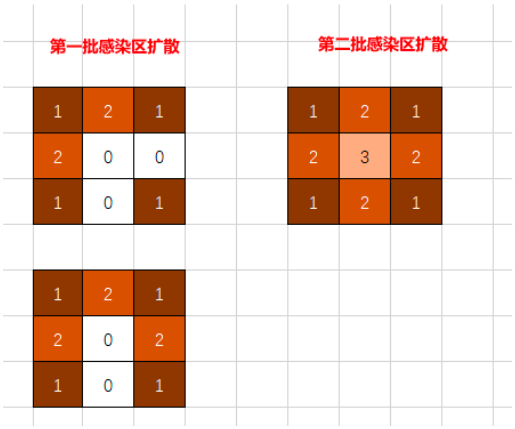
每个感染区就是图结构中需要进行广度优先搜索的起点。感染区就相当于我们往水面扔了一颗石子，广度优先搜索就相当于荡起的一圈涟漪。

本题的广度优先搜索是基于队列实现的。

创建一个队列queue，初始时，遍历矩阵，找到所有感染区位置，并加入队列。

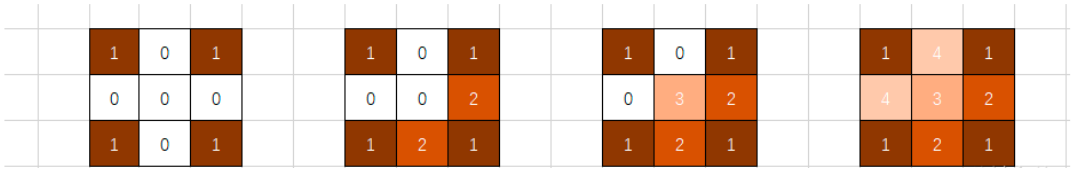
queue初始化完成后，我们对queue进行出队操作，每一个出队元素就是一个感染区位置，我们需要将其上下左右的区域全部改为感染区，并将新的感染区入队queue。

这样的话才能保证 第一批感染区 的传播才能优先进行，达到广度优先搜索的目的。



我们再举一个例子，如果采用stack栈来保存感染区的话，则必然先弹栈一个感染区位置，然后将其上下左右区域感染，这个过程中，将新的感染区压栈，而之后再次弹栈，必然是第二批的感染区位置，也就是后进先出，这将会产生 深度优先搜索 的效果。

最终会产生如下效果



了解图的多源广度优先搜索的实现后，我们就需要考虑如何统计感染时间了，这里我们可以将感染区标记和感染时间捆绑在一起，比如第0天的新增感染区标记为1（即矩阵初始时元素1），第1天的新增感染区标记为2，这个标记是我们广度优先搜索过程中，遍历每个感染区位置上下左右时标记的。

因此，最后一次被标记的时间就是感染全区的时间，但是要减去1，因为我们是第1天标记为2了，因此第n天标记为n+1了。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const arr = line.split(",").map(Number);
  console.log(howLong(arr));
});

function howLong(arr) {
  // 题目说会输入n*n个值
  const n = Math.sqrt(arr.length);

  // 将一维输入转为二维矩阵
  const matrix = new Array(n).fill(0).map(() => new Array(n).fill(0));

  // 将矩阵中所有感染区域位置记录到queue中,这里选择queue先进先出的原因是保证当天的感染区域并发扩散
  let queue = [];
  for (let i = 0; i < n; i++) {
    for (let j = 0; j < n; j++) {
      matrix[i][j] = arr[i * n + j];
      if (matrix[i][j] === 1) queue.push([i, j]);
    }
  }

  // 全是感染区，或全是健康区
  if (queue.length === 0 || queue.length === arr.length) {
    return -1;
  }

  // 健康区个数
  let healthy = arr.length - queue.length;

  // 上下左右位置偏移量
  const offset = [
    [-1, 0], // 上
    [1, 0], // 下
    [0, -1], // 左
    [0, 1], // 右
  ];

  // day用于统计感染全部花费的时间，理论上应该从1开始统计，但是这里从2开始统计，因此最终结果要减去1，为什么从2开始统计，是因为这
  let day;
  // 如果健康区个数为0，说明感染完了
  while (queue.length && healthy) {
    const newQueue = [];
    for (const [x, y] of queue) {
      day = matrix[x][y] + 1;

```

```
      for (let i = 0; i < offset.length; i++) {
        const [offsetX, offsetY] = offset[i];
        const newX = x + offsetX;
        const newY = y + offsetY;

        if (newX < 0 || newX >= n || newY < 0 || newY >= n) continue;

        if (matrix[newX][newY] === 0) {
          healthy--;
          matrix[newX][newY] = day;
          newQueue.push([newX, newY]);
        }
      }
    }
    queue = newQueue;
  }

  return day - 1;
}
```

Java算法源码

```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(",")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  public static int getResult(Integer[] arr) {
    // 题目说会输入n*n个值
    int n = (int) Math.sqrt(arr.length);

    // 将一维arr输入转为二维矩阵matrix
    int[][] matrix = new int[n][n];

    // 将矩阵中所有感染区域位置记录到queue中,这里选择queue先进先出的原因是保证当天的感染区域并发扩散
    LinkedList<Integer[]> queue = new LinkedList<>();

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        matrix[i][j] = arr[i * n + j];
        if (matrix[i][j] == 1) queue.add(new Integer[] {i, j});
      }
    }

    // 全是感染区，或全是健康区
    if (queue.size() == 0 || queue.size() == arr.length) {
      return -1;
    }

    // 健康区个数
```

```
    int healthy = arr.length - queue.size();

    // 上下左右偏移量
    int[][] offsets = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    // day用于统计感染全部花费的时间，理论上应该从1开始统计，但是这里从2开始统计，因此最终结果要减去1，为什么从2开始统计，是因为
    int day = 0;
    // 如果健康区个数为0，说明感染完了
    while (queue.size() > 0 && healthy > 0) {
      //  利用queue的先进先出特点，才能保证同一天的感染区并发传播，即广度优先搜索
      Integer[] tmp = queue.removeFirst();
      int x = tmp[0], y = tmp[1];
      day = matrix[x][y] + 1;

      for (int[] offset : offsets) {
        int newX = x + offset[0];
        int newY = y + offset[1];

        if (newX < 0 || newX >= n || newY < 0 || newY >= n) continue;

        if (matrix[newX][newY] == 0) {
          healthy--;
          matrix[newX][newY] = day;
          queue.add(new Integer[] {newX, newY});
        }
      }
    }

    return day - 1;
  }
}
```

Python算法源码

```
import math

# 输入获取
arr = list(map(int, input().split(",")))


# 算法入口
def getResult(arr):
    # 题目说会输入n*n个值
    n = int(math.sqrt(len(arr)))

    #  将一维arr输入转为二维矩阵matrix
    matrix = [[0 for _ in range(n)] for _ in range(n)]

    # 将矩阵中所有感染区域位置记录到queue中,这里选择queue先进先出的原因是保证当天的感染区域并发扩散
    queue = []

    for i in range(n):
        for j in range(n):
            matrix[i][j] = arr[i * n + j]
            if matrix[i][j] == 1:
                queue.append([i, j])

    # 全是感染区，或全是健康区
    if len(queue) == 0 or len(queue) == len(arr):
        return -1
```

```
    # 健康区个数
    healthy = len(arr) - len(queue)

    # 上下左右偏移量
    offsets = ((-1, 0), (1, 0), (0, -1), (0, 1))

    # day用于统计感染全部花费的时间，理论上应该从1开始统计，但是这里从2开始统计，因此最终结果要减去1，为什么从2开始统计，是因为这
    day = 0
    # 如果健康区个数为0，说明感染完了
    while len(queue) > 0 and healthy > 0:
        newQueue = []

        for x, y in queue:
            day = matrix[x][y] + 1

            for offsetX, offsetY in offsets:
                newX = x + offsetX
                newY = y + offsetY

                if newX < 0 or newX >= n or newY < 0 or newY >= n:
                    continue

                if matrix[newX][newY] == 0:
                    healthy -= 1
                    matrix[newX][newY] = day
                    newQueue.append([newX, newY])
```

```
        queue = newQueue

    return day - 1


# 算法调用
print(getResult(arr))
```