51、矩阵扩散，考点 or 实现——图论/图的多元 BFS

题目描述

存在一个m×n的二维数组，其成员取值范围为0或1。

其中值为1的成员具备扩散性，每经过1S，将上下左右值为0的成员同化为1。

二维数组的成员初始值都为0，将第[i,j]和[k,l]两个个位置上元素修改成1后，求矩阵的所有元素变为1需要多长时间。

输入描述

输入数据中的前2个数字表示这是一个m×n的矩阵，m和n不会超过1024大小；

中间两个数字表示一个初始扩散点位置为i,j；

最后2个数字表示另一个扩散点位置为k,l。

输出描述

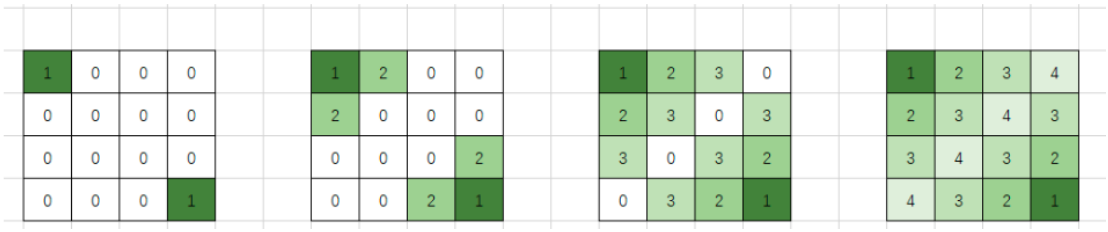
输出矩阵的所有元素变为1所需要秒数。

用例

输入	4,4,0,0,3,3
输出	3
说明	输入数据中的前2个数字表示这是一个4*4的矩阵； 中间两个数字表示一个初始扩散点位置为0,0； 最后2个数字表示另一个扩散点位置为3,3。 给出的样例是一个简单模型，初始点在对角线上，达到中间的位置分别为3次迭代，即3秒。所以输出为3。

题目解析

用例图示如下：



一共需要4-1=3秒。

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 [m, n, i, j, k, l] = line.split(",").map(Number);
  console.log(getResult(m, n, i, j, k, l));
});

/**
 * @param {*} m m×n的二维数组
 * @param {*} n m×n的二维数组
 * @param {*} i 扩散点位置为i,j
 * @param {*} j 扩散点位置为i,j
 * @param {*} k 扩散点位置为k,l
 * @param {*} l 扩散点位置为k,l
 */
function getResult(m, n, i, j, k, l) {
  const matrix = new Array(m).fill(0).map(() => new Array(n).fill(0));
  matrix[i][j] = 1;
  matrix[k][l] = 1;

  // count记录未被扩散的点的数量
  let count = m * n - 2;
```

```
29
30    // 多源BFS实现队列
31    let queue = [
32      [i, j],
33      [k, l],
34    ];
35
36    // 上下左右偏移量
37    const offsets = [
38      [1, 0],
39      [-1, 0],
40      [0, 1],
41      [0, -1],
42    ];
43
44    let day = 1;
45    // 如果扩散点没有了，或者所有点已被扩散，则停止循环
46    while (queue.length && count) {
47      const newQueue = [];
48
49      for (const [x, y] of queue) {
50        // 我们假设初始扩散点的1代表第1秒被扩散到的，则下一波被扩散点的值就是1+1，即第2秒被扩散到的
51        day = matrix[x][y] + 1;
52
53        for (let offset of offsets) {
54          const [offsetX, offsetY] = offset;
55          const newX = x + offsetX;
```

```
56          const newY = y + offsetY;
57
58          if (
59            newX >= 0 &&
60            newX < m &&
61            newY >= 0 &&
62            newY < n &&
63            matrix[newX][newY] === 0
64          ) {
65            // 将点被扩散的时间记录为该点的值
66            matrix[newX][newY] = day;
67            // 被扩散到的点将变为新的扩散源
68            newQueue.push([newX, newY]);
69            // 未被扩散点的数量--
70            count--;
71          }
72        }
73      }
74
75      queue = newQueue;
76    }
77
78    return day - 1;
79  }
```

Java算法源码

```java
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.next().split(",")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr[0], arr[1], arr[2], arr[3], arr[4], arr[5]));
  }

  /**
   * @param m m m×n的二维数组
   * @param n n m×n的二维数组
   * @param i 扩散点位置为i,j
   * @param j 扩散点位置为i,j
   * @param k 扩散点位置为k,l
   * @param l 扩散点位置为k,l
   * @return 扩散所有点需要的时间
   */
  public static int getResult(int m, int n, int i, int j, int k, int l) {
    int[][] matrix = new int[m][n];
    matrix[i][j] = 1;
    matrix[k][l] = 1;

    // count记录未被扩散的点的数量
    int count = m * n - 2;

    // 多源BFS实现队列
    LinkedList<int[]> queue = new LinkedList<>();
    queue.addLast(new int[] {i, j});
    queue.addLast(new int[] {k, l});

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

    int day = 1;
    // 如果扩散点没有了，或者所有点已被扩散，则停止循环
    while (queue.size() > 0 && count > 0) {
      int[] tmp = queue.removeFirst();
      int x = tmp[0];
      int y = tmp[1];

      // 我们假设初始扩散点的1代表第1秒被扩散到的，则下一波被扩散点的值就是1+1，即第2秒被扩散到的
      day = matrix[x][y] + 1;

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

```
        int newY = y + offset[1];

        if (newX >= 0 && newX < m && newY >= 0 && newY < n && matrix[newX][newY] == 0) {
          // 将点被扩散的时间记录为该点的值
          matrix[newX][newY] = day;
          // 被扩散到的点将变为新的扩散源
          queue.addLast(new int[] {newX, newY});
          // 未被扩散点的数量--
          count--;
        }
      }
    }

    return day - 1;
  }
}
```

Python算法源码

```
# 输入获取
m, n, i, j, k, l = map(int, input().split(","))


# 算法入口
def getResult(m, n, i, j, k, l):
    """
    :param m: 矩阵行数
    :param n: 矩阵列数
    :param i: 扩散点1行号
    :param j: 扩散点1列好
    :param k: 扩散点2行号
    :param l: 扩散点2列号
    :return: 矩阵的所有元素变为1所需要秒数
    """
    matrix = [[0 for _ in range(n)] for _ in range(m)]
    matrix[i][j] = 1
    matrix[k][l] = 1

    # count记录未被扩散的点的数量
    count = m * n - 2

    # 多源BFS实现队列
    queue = [[i, j], [k, l]]

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

```
28
29      day = 1
30      # 如果扩散点没有了，或者所有点已被扩散，则停止循环
31      while len(queue) > 0 and count > 0:
32          newQueue = []
33
34          for x, y in queue:
35              # 我们假设初始扩散点的1代表第1秒被扩散到的，则下一波被扩散点的值就是1+1，即第2秒被扩散到的
36              day = matrix[x][y] + 1
37
38              for offsetX, offsetY in offsets:
39                  newX = x + offsetX
40                  newY = y + offsetY
41
42                  if 0 <= newX < m and 0 <= newY < n and matrix[newX][newY] == 0:
43                      # 将点被扩散的时间记录为该点的值
44                      matrix[newX][newY] = day
45                      # 被扩散到的点将变为新的扩散源
46                      newQueue.append([newX, newY])
47                      # 未被扩散点的数量--
48                      count -= 1
49
50          queue = newQueue
51
52      return day - 1
53
54
55  # 算法调用
56  print(getResult(m, n, i, j, k, l))
```