一、编程题

ACM： 可以组成网络的服务器

题目描述：

在一个机房中，服务器的位置标识在 n*m 的整数矩阵网格中，1 表示单元格上有服务器，0 表示没有。如果两台服务器位于同一行或者同一列中紧邻的位置，则认为它们之间可以组成一个局域网。

请你统计机房中最大的局域网包含的服务器个数。

输入描述：

第一行输入两个正整数，n 和 m，0<n,m<=100

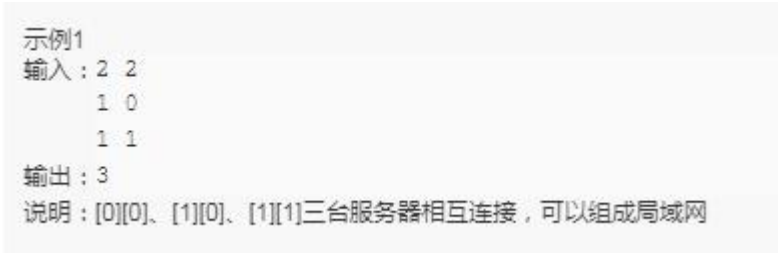
之后为 n*m 的二维数组，代表服务器信息

输出描述：

最大局域网包含的服务器个数。

补充说明：

示例




代码：

```
import java.util.Scanner;
import java.io.*;
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader bufR = new BufferedReader(new InputStreamReader(System.in));

        String strIn;
        while ( (strIn = bufR.readLine()) != null ) {
            String[] array_str_nm = strIn.split(" ");
            int n = Integer.parseInt(array_str_nm[0]);
            int m = Integer.parseInt(array_str_nm[1]);
            int[][] Matrix = new int[n][m];
            for (int i = 0; i < n; i++) {
                String[] array_strS_nRow = bufR.readLine().split(" ");
                for (int j = 0; j < m; j++) {
                    Matrix[i][j] = Integer.parseInt(array_strS_nRow[j]);
                }
            }
            //
```

```
                boolean[][] Matrix_had = new boolean[n][m];
                for (int i = 0; i < n; i++) {
                        Arrays.fill(Matrix_had[i], false);
                }
                //
                for (int i = 0; i < n; i++) {
                        int maxTemp = 0;
                        for (int j = 0; j < m; j++) {
                                if (Matrix[i][j] == 1 && !Matrix_had[i][j]) {
                                        method(i,j,Matrix,Matrix_had,maxTemp);
                                        //methodB(i,j,Matrix,Matrix_had,maxTemp,Max);
                                }
                        }
                }
                System.out.println(Max);
        }

        bufR.close();
    }
    // 求解以点(x,y)为起点的局域网的最大机器个数
    private static int Max = 0;
    private static void method(int index_x, int index_y,int[][] Matrix, boolean[][] Matrix_had, int
max) {
        if (
                (index_x >=0 && index_x < Matrix.length)
                &&
                (index_y >=0 && index_y < Matrix[0].length)
                &&
                Matrix[index_x][index_y] == 1
                &&
                !Matrix_had[index_x][index_y]
        ) {
                max++;
                Matrix_had[index_x][index_y] = true;
                if (max > Max) {
                        Max = max;
                }
                // 上
                if (
                        (index_x - 1 >= 0)
                        &&
                        Matrix[index_x-1][index_y] == 1
                        &&
                        !Matrix_had[index_x-1][index_y]
```

```
                ) {
                    method( index_x-1, index_y, Matrix,    Matrix_had, Max);
                }
                // 右
                if (
                    (index_y + 1 < Matrix[0].length)
                    &&
                    Matrix[index_x][index_y+1] == 1
                    &&
                    !Matrix_had[index_x][index_y+1]
                ) {
                    method( index_x, index_y+1, Matrix,    Matrix_had, Max);
                }
                // 下
                if (
                    (index_x + 1 < Matrix.length)
                    &&
                    Matrix[index_x+1][index_y] == 1
                    &&
                    !Matrix_had[index_x+1][index_y]
                ) {
                    method( index_x+1, index_y, Matrix,    Matrix_had, Max);
                }
                // 左
                if (
                    (index_y - 1 >= 0)
                    &&
                    Matrix[index_x][index_y-1] == 1
                    &&
                    !Matrix_had[index_x][index_y-1]
                ) {
                    method( index_x, index_y-1, Matrix,    Matrix_had, Max);
                }
            }
        }
        private static void methodB(int index_x, int index_y,int[][] Matrix, boolean[][] Matrix_had,
int max, int m) {
            if (
                (index_x >=0 && index_x < Matrix.length)
                &&
                (index_y >=0 && index_y < Matrix[0].length)
                &&
                Matrix[index_x][index_y] == 1
                &&
```

```
        !Matrix_had[index_x][index_y]
    ) {
        max++;
        Matrix_had[index_x][index_y] = true;
        if (max > Max) {
            Max = max;
        }
        // 上
        if (
            (index_x - 1 >= 0)
            &&
            Matrix[index_x-1][index_y] == 1
            &&
            !Matrix_had[index_x-1][index_y]
        ) {
            methodB( index_x-1, index_y, Matrix,    Matrix_had, max,Max);
        }
        // 右
        if (
            (index_y + 1 < Matrix[0].length)
            &&
            Matrix[index_x][index_y+1] == 1
            &&
            !Matrix_had[index_x][index_y+1]
        ) {
            methodB( index_x, index_y+1, Matrix,    Matrix_had, max,Max);
        }
        // 下
        if (
            (index_x + 1 < Matrix.length)
            &&
            Matrix[index_x+1][index_y] == 1
            &&
            !Matrix_had[index_x+1][index_y]
        ) {
            methodB( index_x+1, index_y, Matrix,    Matrix_had, max,Max);
        }
        // 左
        if (
            (index_y - 1 >= 0)
            &&
            Matrix[index_x][index_y-1] == 1
            &&
            !Matrix_had[index_x][index_y-1]
```

```
                ) {
                    methodB( index_x, index_y-1, Matrix,    Matrix_had, max,Max);
                }
            }
        }
    }
```