【DFS/BFS】-可以组成网络的服务器

题目描述与示例

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。

示例

输入

```
2 2
1 0
1 1
```

输出

```
3
```

补充说明

`[0][0]`、`[1][0]`、`[1][1]` 三台服务器相互连接，可以组成局域网

解题思路

> 注意，本题和LeetCode695、岛屿的最大面积 完全一致，直接套模板即可。

代码

解法一：DFS

Python

```python
# 题目：2024E-可以组成网络的服务器
# 分值：200
# 作者：闭着眼睛学数理化
# 算法：DFS
# 代码看不懂的地方，请直接在群上提问

import sys
sys.setrecursionlimit(10000)

# 初始化上下左右四个方向的数组
DERICTIONS = [(0,1), (1,0), (0,-1), (-1,0)]

# 构建DFS函数
def DFS(grid, i, j, checkList):
    global area
    # 将该点标记为已经检查过
    checkList[i][j] = True
    # 面积增大
    area += 1
    # 遍历上下左右四个方向的邻点坐标
    for dx, dy in DERICTIONS:
        next_i, next_j = i + dx, j + dy
        # 若近邻点满足三个条件：
        # 1.没有越界    2. 近邻点尚未被检查过    3.近邻点也为陆地
        if ((0 <= next_i < n and 0 <= next_j < m) and checkList[next_i][next_j] == False
            and grid[next_i][next_j] == 1):
            # 对近邻点(ni, nj)进行DFS搜索
            DFS(grid, next_i, next_j, checkList)


# 输入长、宽
n, m = map(int, input().split())
# 构建网格
grid = list()
for _ in range(n):
    grid.append(list(map(int, input().split())))

ans = 0
# 初始化数组checkList用于DFS遍历过程中的检查
# 0表示尚未访问，1表示已经访问
checkList = [[False] * m for _ in range(n)]

# 对整个grid二维数组进行双重循环遍历
for i in range(n):
    for j in range(m):
        # 若该点为陆地且还没有进行过搜寻
        if grid[i][j] == 1 and checkList[i][j] == False:
            # 在每一次DFS之前，先初始化面积为0
            area = 0
            # 可以进行DFS
            DFS(grid, i, j, checkList)
            # 做完DFS，更新ans
            ans = max(ans, area)

print(ans)
```

Java

```java
import java.util.Scanner;

public class Main {
    static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    static int n, m;
    static int[][] grid;
    static boolean[][] checkList;
    static int area;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        n = scanner.nextInt();
        m = scanner.nextInt();
        grid = new int[n][m];
        checkList = new boolean[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        int ans = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 1 && !checkList[i][j]) {
                    area = 0;
                    DFS(i, j);
                    ans = Math.max(ans, area);
                }
            }
        }

        System.out.println(ans);
    }

    static void DFS(int i, int j) {
        checkList[i][j] = true;
        area++;

        for (int[] dir : DIRECTIONS) {
            int nextI = i + dir[0];
            int nextJ = j + dir[1];
            if (isValid(nextI, nextJ) && !checkList[nextI][nextJ] && grid[nextI][nextJ] == 1) {
                DFS(nextI, nextJ);
            }
        }
    }

    static boolean isValid(int i, int j) {
        return i >= 0 && i < n && j >= 0 && j < m;
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

vector<pair<int, int>> DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
int n, m;
vector<vector<int>> grid;
vector<vector<bool>> checkList;
int area;

void DFS(int i, int j) {
    checkList[i][j] = true;
    area++;

    for (auto dir : DIRECTIONS) {
        int nextI = i + dir.first;
        int nextJ = j + dir.second;
        if (nextI >= 0 && nextI < n && nextJ >= 0 && nextJ < m && !checkList[nextI][nextJ] && grid[nextI][nextJ] == 1) {
            DFS(nextI, nextJ);
        }
    }
}

int main() {
    cin >> n >> m;
    grid.resize(n, vector<int>(m));
    checkList.resize(n, vector<bool>(m, false));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    int ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1 && !checkList[i][j]) {
                area = 0;
                DFS(i, j);
                ans = max(ans, area);
            }
        }
    }

    cout << ans << endl;

    return 0;
}

```

解法二：BFS

Python

```python
# 题目：2024E-可以组成网络的服务器
# 分值：200
# 作者：闭着眼睛学数理化
# 算法：BFS
# 代码看不懂的地方，请直接在群上提问


from collections import deque

# 初始化上下左右四个方向的数组
DIRECTIONS = [(0,1), (1,0), (0,-1), (-1,0)]

# 输入长、宽
n, m = map(int, input().split())
# 构建网格
grid = list()
for _ in range(n):
    grid.append(list(map(int, input().split())))

ans = 0
# 初始化和grid一样大小的二维数组checkList用于DFS遍历过程中的检查
checkList = [[0] * m for _ in range(n)]
# 双重遍历grid数组
for i in range(n):
    for j in range(m):
        # 若该点为1且还没有进行过搜寻
        # 找到了一个BFS搜索的起始位置(i,j)
        if grid[i][j] == 1 and checkList[i][j] == 0:
            # 对于该片连通块，构建一个队列，初始化包含该点
            q = deque()
            q.append((i, j))
            # 修改checkList[i][j]为1，表示该点已经搜寻过
            checkList[i][j] = 1
            # 进行BFS之前，初始化该连通块的面积为0
            area = 0
            # 进行BFS，退出循环的条件是队列为空
            while len(q) > 0:
                # 弹出栈队头的点(x,y) 搜寻该点上下左右的近邻点
                x, y = q.popleft()
                area += 1
                # 遍历(x,y)上下左右的四个方向的近邻点
                for dx, dy in DIRECTIONS:
                    x_next, y_next = x+dx, y+dy
                    # 如果近邻点满足三个条件
                    if (0 <= x_next < n and 0 <= y_next < m and checkList[x_next][y_next] == 0
                            and grid[x_next][y_next] == 1):
                            # 对近邻点做两件事：
                            # 1. 入队       2. 标记为已检查过
                            q.append((x_next, y_next))
                            checkList[x_next][y_next] = 1
            # 更新答案
            ans = max(ans, area)
print(ans)
```

Java

```
import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Scanner;

public class Main {
    static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int m = scanner.nextInt();

        int[][] grid = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        int ans = 0;
        int[][] checkList = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 1 && checkList[i][j] == 0) {
                    Queue<int[]> queue = new ArrayDeque<>();
                    queue.offer(new int[]{i, j});
                    checkList[i][j] = 1;
                    int area = 0;

                    while (!queue.isEmpty()) {
                        int[] point = queue.poll();
                        int x = point[0];
                        int y = point[1];
                        area++;

                        for (int[] dir : DIRECTIONS) {
                            int xNext = x + dir[0];
                            int yNext = y + dir[1];
                            if (xNext >= 0 && xNext < n && yNext >= 0 && yNext < m
                                    && checkList[xNext][yNext] == 0 && grid[xNext][yNext] == 1) {
                                queue.offer(new int[]{xNext, yNext});
                                checkList[xNext][yNext] = 1;
                            }
                        }
                    }

                    ans = Math.max(ans, area);
                }
            }
        }

        System.out.println(ans);
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

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

int main() {
    int n, m;
    cin >> n >> m;

    vector<vector<int>> grid(n, vector<int>(m));
    vector<vector<int>> checkList(n, vector<int>(m, 0));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> grid[i][j];
        }
    }

    int ans = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (grid[i][j] == 1 && checkList[i][j] == 0) {
                queue<pair<int, int>> q;
                q.push({i, j});
                checkList[i][j] = 1;
                int area = 0;
                while (!q.empty()) {
                    int x = q.front().first;
                    int y = q.front().second;
                    q.pop();
                    area++;
                    for (auto dir : DIRECTIONS) {
                        int x_next = x + dir.first;
                        int y_next = y + dir.second;
                        if (x_next >= 0 && x_next < n && y_next >= 0 && y_next < m &&
                            checkList[x_next][y_next] == 0 && grid[x_next][y_next] == 1) {
                            q.push({x_next, y_next});
                            checkList[x_next][y_next] = 1;
                        }
                    }
                }
                ans = max(ans, area);
            }
        }
    }

    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(NM)`。

空间复杂度：`O(NM)`。