

# 【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  $n \times m$  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  $m$ ， $0 < n, m \leq 100$   
之后为  $n \times m$  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

```
1 2 2
2 1 0
3 1 1
```

#### 输出

## 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

```
1 # 题目：2024E-可以组成网络的服务器
2 # 分值：200
3 # 作者：闭着眼睛学数理化
4 # 算法：DFS
5 # 代码看不懂的地方，请直接在群上提问
6
7 import sys
8 sys.setrecursionlimit(10000)
9
10 # 初始化上下左右四个方向的数组
11 DIRECTIONS = [(0,1), (1,0), (0,-1), (-1,0)]
12
13 # 构建DFS函数
14 def DFS(grid, i, j, checkList):
15     global area
16     # 将该点标记为已经检查过
17     checkList[i][j] = True
18     # 面积增大
19     area += 1
20     # 遍历上下左右四个方向的邻点坐标
21     for dx, dy in DIRECTIONS:
22         next_i, next_j = i + dx, j + dy
23         # 若近邻点满足三个条件：
24         # 1. 没有越界    2. 近邻点尚未被检查过    3. 近邻点也为陆地
```

```

25         if ((0 <= next_i < n and 0 <= next_j < m) and checkList[next_i]
[next_j] == False
26             and grid[next_i][next_j] == 1):
27             # 对近邻点(ni, nj)进行DFS搜索
28             DFS(grid, next_i, next_j, checkList)
29
30
31 # 输入长、宽
32 n, m = map(int, input().split())
33 # 构建网格
34 grid = list()
35 for _ in range(n):
36     grid.append(list(map(int, input().split())))
37
38 ans = 0
39 # 初始化数组checkList用于DFS遍历过程中的检查
40 # 0表示尚未访问, 1表示已经访问
41 checkList = [[False] * m for _ in range(n)]
42
43 # 对整个grid二维数组进行双重循环遍历
44 for i in range(n):
45     for j in range(m):
46         # 若该点为陆地且还没有进行过搜寻
47         if grid[i][j] == 1 and checkList[i][j] == False:
48             # 在每一次DFS之前, 先初始化面积为0
49             area = 0
50             # 可以进行DFS
51             DFS(grid, i, j, checkList)
52             # 做完DFS, 更新ans
53             ans = max(ans, area)
54
55 print(ans)

```

## Java

```

1 import java.util.Scanner;
2
3 public class Main {
4     static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
5     static int n, m;
6     static int[][] grid;
7     static boolean[][] checkList;
8     static int area;
9
10    public static void main(String[] args) {

```

```

11     Scanner scanner = new Scanner(System.in);
12     n = scanner.nextInt();
13     m = scanner.nextInt();
14     grid = new int[n][m];
15     checkList = new boolean[n][m];
16
17     for (int i = 0; i < n; i++) {
18         for (int j = 0; j < m; j++) {
19             grid[i][j] = scanner.nextInt();
20         }
21     }
22
23     int ans = 0;
24
25     for (int i = 0; i < n; i++) {
26         for (int j = 0; j < m; j++) {
27             if (grid[i][j] == 1 && !checkList[i][j]) {
28                 area = 0;
29                 DFS(i, j);
30                 ans = Math.max(ans, area);
31             }
32         }
33     }
34
35     System.out.println(ans);
36 }
37
38 static void DFS(int i, int j) {
39     checkList[i][j] = true;
40     area++;
41
42     for (int[] dir : DIRECTIONS) {
43         int nextI = i + dir[0];
44         int nextJ = j + dir[1];
45         if (isValid(nextI, nextJ) && !checkList[nextI][nextJ] &&
grid[nextI][nextJ] == 1) {
46             DFS(nextI, nextJ);
47         }
48     }
49 }
50
51 static boolean isValid(int i, int j) {
52     return i >= 0 && i < n && j >= 0 && j < m;
53 }
54 }
55

```

# C++

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 vector<pair<int, int>> DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
7 int n, m;
8 vector<vector<int>> grid;
9 vector<vector<bool>> checkList;
10 int area;
11
12 void DFS(int i, int j) {
13     checkList[i][j] = true;
14     area++;
15
16     for (auto dir : DIRECTIONS) {
17         int nextI = i + dir.first;
18         int nextJ = j + dir.second;
19         if (nextI >= 0 && nextI < n && nextJ >= 0 && nextJ < m &&
20             !checkList[nextI][nextJ] && grid[nextI][nextJ] == 1) {
21             DFS(nextI, nextJ);
22         }
23     }
24
25 int main() {
26     cin >> n >> m;
27     grid.resize(n, vector<int>(m));
28     checkList.resize(n, vector<bool>(m, false));
29
30     for (int i = 0; i < n; i++) {
31         for (int j = 0; j < m; j++) {
32             cin >> grid[i][j];
33         }
34     }
35
36     int ans = 0;
37
38     for (int i = 0; i < n; i++) {
39         for (int j = 0; j < m; j++) {
40             if (grid[i][j] == 1 && !checkList[i][j]) {
41                 area = 0;
42                 DFS(i, j);
43                 ans = max(ans, area);
44             }
45         }
46     }
```

```

44         }
45     }
46 }
47
48     cout << ans << endl;
49
50     return 0;
51 }
52

```

## 解法二：BFS

### Python

```

1  # 题目：2024E-可以组成网络的服务器
2  # 分值：200
3  # 作者：闭着眼睛学数理化
4  # 算法：BFS
5  # 代码看不懂的地方，请直接在群上提问
6
7
8  from collections import deque
9
10 # 初始化上下左右四个方向的数组
11 DIRECTIONS = [(0,1), (1,0), (0,-1), (-1,0)]
12
13 # 输入长、宽
14 n, m = map(int, input().split())
15 # 构建网格
16 grid = list()
17 for _ in range(n):
18     grid.append(list(map(int, input().split())))
19
20 ans = 0
21 # 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
22 checkList = [[0] * m for _ in range(n)]
23 # 双重遍历grid数组
24 for i in range(n):
25     for j in range(m):
26         # 若该点为1且还没有进行过搜寻
27         # 找到了一个BFS搜索的起始位置(i,j)

```

```

28     if grid[i][j] == 1 and checkList[i][j] == 0:
29         # 对于该片连通块，构建一个队列，初始化包含该点
30         q = deque()
31         q.append((i, j))
32         # 修改checkList[i][j]为1，表示该点已经搜寻过
33         checkList[i][j] = 1
34         # 进行BFS之前，初始化该连通块的面积为0
35         area = 0
36         # 进行BFS，退出循环的条件是队列为空
37         while len(q) > 0:
38             # 弹出栈队头的点(x,y) 搜寻该点上下左右的近邻点
39             x, y = q.popleft()
40             area += 1
41             # 遍历(x,y)上下左右的四个方向的近邻点
42             for dx, dy in DIRECTIONS:
43                 x_next, y_next = x+dx, y+dy
44                 # 如果近邻点满足三个条件
45                 if (0 <= x_next < n and 0 <= y_next < m and
checkList[x_next][y_next] == 0
46                     and grid[x_next][y_next] == 1):
47                     # 对近邻点做两件事：
48                     # 1. 入队          2. 标记为已检查过
49                     q.append((x_next, y_next))
50                     checkList[x_next][y_next] = 1
51             # 更新答案
52             ans = max(ans, area)
53 print(ans)

```

## Java

```

1 import java.util.ArrayDeque;
2 import java.util.Queue;
3 import java.util.Scanner;
4
5 public class Main {
6     static final int[][] DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
7
8     public static void main(String[] args) {
9         Scanner scanner = new Scanner(System.in);
10         int n = scanner.nextInt();
11         int m = scanner.nextInt();
12
13         int[][] grid = new int[n][m];
14         for (int i = 0; i < n; i++) {
15             for (int j = 0; j < m; j++) {

```

```

16         grid[i][j] = scanner.nextInt();
17     }
18 }
19
20 int ans = 0;
21 int[][] checkList = new int[n][m];
22
23 for (int i = 0; i < n; i++) {
24     for (int j = 0; j < m; j++) {
25         if (grid[i][j] == 1 && checkList[i][j] == 0) {
26             Queue<int[]> queue = new ArrayDeque<>();
27             queue.offer(new int[]{i, j});
28             checkList[i][j] = 1;
29             int area = 0;
30
31             while (!queue.isEmpty()) {
32                 int[] point = queue.poll();
33                 int x = point[0];
34                 int y = point[1];
35                 area++;
36
37                 for (int[] dir : DIRECTIONS) {
38                     int xNext = x + dir[0];
39                     int yNext = y + dir[1];
40                     if (xNext >= 0 && xNext < n && yNext >= 0 && yNext
41 < m
42                     && checkList[xNext][yNext] == 0 &&
43 grid[xNext][yNext] == 1) {
44                         queue.offer(new int[]{xNext, yNext});
45                         checkList[xNext][yNext] = 1;
46                     }
47                 }
48                 ans = Math.max(ans, area);
49             }
50         }
51     }
52
53     System.out.println(ans);
54 }
55 }
56

```



```

1 #include <iostream>
2 #include <vector>
3 #include <queue>
4
5 using namespace std;
6
7 vector<pair<int, int>> DIRECTIONS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
8
9 int main() {
10     int n, m;
11     cin >> n >> m;
12
13     vector<vector<int>> grid(n, vector<int>(m));
14     vector<vector<int>> checkList(n, vector<int>(m, 0));
15
16     for (int i = 0; i < n; ++i) {
17         for (int j = 0; j < m; ++j) {
18             cin >> grid[i][j];
19         }
20     }
21
22     int ans = 0;
23     for (int i = 0; i < n; ++i) {
24         for (int j = 0; j < m; ++j) {
25             if (grid[i][j] == 1 && checkList[i][j] == 0) {
26                 queue<pair<int, int>> q;
27                 q.push({i, j});
28                 checkList[i][j] = 1;
29                 int area = 0;
30                 while (!q.empty()) {
31                     int x = q.front().first;
32                     int y = q.front().second;
33                     q.pop();
34                     area++;
35                     for (auto dir : DIRECTIONS) {
36                         int x_next = x + dir.first;
37                         int y_next = y + dir.second;
38                         if (x_next >= 0 && x_next < n && y_next >= 0 && y_next
< m &&
39                             checkList[x_next][y_next] == 0 && grid[x_next]
[y_next] == 1) {
40                             q.push({x_next, y_next});
41                             checkList[x_next][y_next] = 1;
42                         }
43                     }
44                 }
45                 ans = max(ans, area);

```

```
46         }
47     }
48 }
49
50     cout << ans << endl;
51
52     return 0;
53 }
54
```

## 时空复杂度

时间复杂度： $O(NM)$ 。

空间复杂度： $O(NM)$ 。