【华为OD机考 统一考试机试C卷】 可以组成网络的服务器（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

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。

用例1

输入

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

输出

```
3
```

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

C++

```cpp
#include <iostream>
#include <vector>
using namespace std;

int n, m;
vector<vector<int>> server;
vector<vector<bool>> visited;

int dfs(int i, int j) {
    if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] == 0) {
        return 0;
    }

    if (visited[i][j]) {
        return 0;
    }

    visited[i][j] = true;

    int count = 1;
    count += dfs(i - 1, j);
    count += dfs(i + 1, j);
    count += dfs(i, j - 1);
    count += dfs(i, j + 1);


```

```
    return count;
}

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

    server.resize(n);
    visited.resize(n);

    for (int i = 0; i < n; i++) {
        server[i].resize(m);
        visited[i].resize(m, false);
        for (int j = 0; j < m; j++) {
            cin >> server[i][j];
        }
    }

    int ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            ans = max(ans, dfs(i, j));
        }
    }

    cout << ans << endl;

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {
    static int n, m;
    static int[][] server; // 定义一个矩阵，用于存储服务器的状态
    static boolean[][] visited; // 记录每个位置是否已经被访问过

    // 定义一个深度优先搜索函数，用于搜索当前位置的连通块大小
```

```
    public static int dfs(int i, int j) {
        // 如果当前位置超出矩阵边界或者当前位置没有服务器，则返回0
        if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] == 0) {
            return 0;
        }

        // 如果当前位置已经被访问过，则返回0
        if (visited[i][j]) {
            return 0;
        }

        // 标记当前位置为已访问
        visited[i][j] = true;

        // 分别向上、下、左、右四个方向递归搜索，并累加连通块大小
        int count = 1; // 当前位置有服务器，将count初始化为1
        count += dfs(i - 1, j); // 上
        count += dfs(i + 1, j); // 下
        count += dfs(i, j - 1); // 左
        count += dfs(i, j + 1); // 右

        return count;
    }

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

        // 读入矩阵的行数和列数
        n = sc.nextInt();
        m = sc.nextInt();
        sc.nextLine();

        server = new int[n][m];
        visited = new boolean[n][m];

        // 读入矩阵中每个位置的状态（0表示没有服务器，1表示有服务器）
        for (int i = 0; i < n; i++) {
            String line = sc.nextLine();
            String[] nums = line.split(" ");
            for (int j = 0; j < m; j++) {
```

```
                server[i][j] = Integer.parseInt(nums[j]);
            }
        }

        int ans = 0; // 定义一个变量，用于存储最大连通块大小

        // 枚举矩阵中的每个位置，以该位置为起点进行深度优先搜索，并更新最大连通块大小
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                ans = Math.max(ans, dfs(i, j));
            }
        }

        // 输出最大连通块大小
        System.out.println(ans);
    }
}
```

javaScript

```
const readline = require('readline');

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

let n, m;
let server = [];
let visited = [];

function dfs(i, j) {
  if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] === 0) {
    return 0;
  }

  if (visited[i][j]) {
    return 0;
  }

```

```
  visited[i][j] = true;

  let count = 1;
  count += dfs(i - 1, j);
  count += dfs(i + 1, j);
  count += dfs(i, j - 1);
  count += dfs(i, j + 1);

  return count;
}

rl.on('line', (line) => {
  if (!n) {
    [n, m] = line.split(' ').map(Number);
    visited = new Array(n).fill(false).map(() => new Array(m).fill(false));
  } else {
    server.push(line.split(' ').map(Number));
    if (server.length === n) {
      let ans = 0;
      for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
          ans = Math.max(ans, dfs(i, j));
        }
      }
      console.log(ans);
      rl.close();
    }
  }
});
```

python

```
import sys

def dfs(i, j):
    if i < 0 or i >= n or j < 0 or j >= m or server[i][j] == 0:
        return 0

    if visited[i][j]:
        return 0
```

```
    visited[i][j] = True

    count = 1
    count += dfs(i - 1, j)
    count += dfs(i + 1, j)
    count += dfs(i, j - 1)
    count += dfs(i, j + 1)

    return count

n, m = map(int, input().split())

server = []
visited = []

for i in range(n):
    line = input().split()
    server.append([int(x) for x in line])
    visited.append([False] * m)

ans = 0

for i in range(n):
    for j in range(m):
        ans = max(ans, dfs(i, j))

print(ans)
```

C语言

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_SIZE 100

int n, m;
int server[MAX_SIZE][MAX_SIZE];
bool visited[MAX_SIZE][MAX_SIZE];
```

```
// 深度优先搜索(DFS)函数，用于计算从(i, j)位置开始能够形成的局域网大小
int dfs(int i, int j) {
    // 如果坐标越界或者当前位置没有服务器，返回0
    if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] == 0) {
        return 0;
    }

    // 如果当前位置已经访问过，也返回0
    if (visited[i][j]) {
        return 0;
    }

    // 标记当前位置为已访问
    visited[i][j] = true;

    // 从当前位置开始，局域网的大小至少为1
    int count = 1;

    // 递归地搜索上下左右四个方向，并累加局域网的大小
    count += dfs(i - 1, j);
    count += dfs(i + 1, j);
    count += dfs(i, j - 1);
    count += dfs(i, j + 1);

    // 返回局域网的总大小
    return count;
}

int main() {
    // 读取矩阵的行数和列数
    scanf("%d %d", &n, &m);

    // 读取服务器矩阵
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &server[i][j]);
            visited[i][j] = false; // 初始化访问状态为未访问
        }
    }
```

```
    int ans = 0; // 存储最大局域网的大小

    // 遍历每个位置，使用DFS寻找最大的局域网
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            // 只有在当前位置有服务器且未访问过时才进行DFS搜索
            if (server[i][j] == 1 && !visited[i][j]) {
                int currentNetworkSize = dfs(i, j);
                if (currentNetworkSize > ans) {
                    ans = currentNetworkSize; // 更新最大局域网的大小
                }
            }
        }
    }

    // 输出最大局域网包含的服务器个数
    printf("%d\n", ans);

    return 0;
}
```

完整用例

用例1

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

用例2

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

用例3

```
4 4
1 0 0 1
0 1 1 0
0 1 1 0
1 0 0 1
```

用例4

```
5 5
1 0 0 1 0
0 1 1 0 1
0 1 1 0 1
1 0 0 1 0
0 1 1 0 1
```

用例5

```
3 4
1 0 1 1
0 1 1 0
1 0 0 1
```

用例6

```
10 10
1 0 0 1 0 0 0 0 0 0
0 1 1 0 1 0 1 1 1 0
0 1 1 0 1 0 1 1 1 0
1 0 0 1 0 0 0 0 0 0
0 1 1 0 1 0 1 1 1 0
0 1 1 0 1 0 1 1 1 0
1 0 0 1 0 0 0 0 0 0
0 1 1 0 1 0 1 1 1 0
0 1 1 0 1 0 1 1 1 0
1 0 0 1 0 0 0 0 0 0
```

用例7

```
6 8
1 0 0 1 0 0 0 0
0 1 1 0 1 0 1 1
0 1 1 0 1 0 1 1
1 0 0 1 0 0 0 0
0 1 1 0 1 0 1 1
1 1 1 0 1 0 1 1
```

用例8

```
7 7
1 0 0 1 0 0 0
0 1 1 0 1 0 1
0 1 1 0 1 0 1
1 0 0 1 0 0 0
0 1 1 0 1 0 1
1 1 1 0 1 0 1
1 1 1 0 1 0 1
```

用例9

```
4 5
1 0 0 1 0
0 1 1 0 1
0 1 1 0 1
1 0 0 1 0
```

用例10

```
5 4
1 0 1 1
0 1 1 0
1 0 0 1
0 1 1 0
1 0 0 1
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师