

【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 ， $0 < n, m \leq 100$

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

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

输出

```
1 | 3
```

[0][0]、[1][0]、[1][1]三台服务器相互连接，可以组成局域网

C++

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int n, m;
6  vector<vector<int>> server;
7  vector<vector<bool>> visited;
8
9  int dfs(int i, int j) {
10     if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] == 0) {
11         return 0;
12     }
13
14     if (visited[i][j]) {
15         return 0;
16     }
17
18     visited[i][j] = true;
19
20     int count = 1;
21     count += dfs(i - 1, j);
22     count += dfs(i + 1, j);
23     count += dfs(i, j - 1);
24     count += dfs(i, j + 1);
25
26 }
```

```
27     return count;
28 }
29
30 int main() {
31     cin >> n >> m;
32
33     server.resize(n);
34     visited.resize(n);
35
36     for (int i = 0; i < n; i++) {
37         server[i].resize(m);
38         visited[i].resize(m, false);
39         for (int j = 0; j < m; j++) {
40             cin >> server[i][j];
41         }
42     }
43
44     int ans = 0;
45
46     for (int i = 0; i < n; i++) {
47         for (int j = 0; j < m; j++) {
48             ans = max(ans, dfs(i, j));
49         }
50     }
51
52     cout << ans << endl;
53
54     return 0;
55 }
```

Java

```
1 import java.util.Scanner;
2
3 public class Main {
4     static int n, m;
5     static int[][] server; // 定义一个矩阵, 用于存储服务器的状态
6     static boolean[][] visited; // 记录每个位置是否已经被访问过
7
8     // 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
9 }
```

```
9 public static int dfs(int i, int j) {
10     // 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
11     if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] == 0) {
12         return 0;
13     }
14
15     // 如果当前位置已经被访问过, 则返回0
16     if (visited[i][j]) {
17         return 0;
18     }
19
20     // 标记当前位置为已访问
21     visited[i][j] = true;
22
23     // 分别向上、下、左、右四个方向递归搜索, 并累加连通块大小
24     int count = 1; // 当前位置有服务器, 将count初始化为1
25     count += dfs(i - 1, j); // 上
26     count += dfs(i + 1, j); // 下
27     count += dfs(i, j - 1); // 左
28     count += dfs(i, j + 1); // 右
29
30     return count;
31 }
32
33 public static void main(String[] args) {
34     Scanner sc = new Scanner(System.in);
35
36     // 读入矩阵的行数和列数
37     n = sc.nextInt();
38     m = sc.nextInt();
39     sc.nextLine();
40
41     server = new int[n][m];
42     visited = new boolean[n][m];
43
44     // 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)
45     for (int i = 0; i < n; i++) {
46         String line = sc.nextLine();
47         String[] nums = line.split(" ");
48         for (int j = 0; j < m; j++) {
49             --
```

```

50         server[i][j] = Integer.parseInt(nums[j]);
51     }
52 }
53
54 int ans = 0; // 定义一个变量, 用于存储最大连通块大小
55
56 // 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
57 for (int i = 0; i < n; i++) {
58     for (int j = 0; j < m; j++) {
59         ans = Math.max(ans, dfs(i, j));
60     }
61 }
62
63 // 输出最大连通块大小
64 System.out.println(ans);
65 }

```

javaScript

```

1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let n, m;
9  let server = [];
10 let visited = [];
11
12 function dfs(i, j) {
13     if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] === 0) {
14         return 0;
15     }
16
17     if (visited[i][j]) {
18         return 0;
19     }
20
21

```

```
21 visited[i][j] = true;
22
23 let count = 1;
24 count += dfs(i - 1, j);
25 count += dfs(i + 1, j);
26 count += dfs(i, j - 1);
27 count += dfs(i, j + 1);
28
29 return count;
30 }
31
32 rl.on('line', (line) => {
33   if (!n) {
34     [n, m] = line.split(' ').map(Number);
35     visited = new Array(n).fill(false).map(() => new Array(m).fill(false));
36   } else {
37     server.push(line.split(' ').map(Number));
38     if (server.length === n) {
39       let ans = 0;
40       for (let i = 0; i < n; i++) {
41         for (let j = 0; j < m; j++) {
42           ans = Math.max(ans, dfs(i, j));
43         }
44       }
45       console.log(ans);
46       rl.close();
47     }
48   }
49 });
```

python

```
1 import sys
2
3 def dfs(i, j):
4     if i < 0 or i >= n or j < 0 or j >= m or server[i][j] == 0:
5         return 0
6
7     if visited[i][j]:
8         return 0
9
10     visited[i][j] = True
11     count = 1
12     count += dfs(i - 1, j)
13     count += dfs(i + 1, j)
14     count += dfs(i, j - 1)
15     count += dfs(i, j + 1)
16     return count
```

```
9
10     visited[i][j] = True
11
12     count = 1
13     count += dfs(i - 1, j)
14     count += dfs(i + 1, j)
15     count += dfs(i, j - 1)
16     count += dfs(i, j + 1)
17
18     return count
19
20 n, m = map(int, input().split())
21
22 server = []
23 visited = []
24
25 for i in range(n):
26     line = input().split()
27     server.append([int(x) for x in line])
28     visited.append([False] * m)
29
30 ans = 0
31
32 for i in range(n):
33     for j in range(m):
34         ans = max(ans, dfs(i, j))
35
36 print(ans)
37
```

C语言

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 #define MAX_SIZE 100
5
6 int n, m;
7 int server[MAX_SIZE][MAX_SIZE];
8 bool visited[MAX_SIZE][MAX_SIZE];
9
```



```
9
10 // 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
11 int dfs(int i, int j) {
12     // 如果坐标越界或者当前位置没有服务器, 返回0
13     if (i < 0 || i >= n || j < 0 || j >= m || server[i][j] == 0) {
14         return 0;
15     }
16
17     // 如果当前位置已经访问过, 也返回0
18     if (visited[i][j]) {
19         return 0;
20     }
21
22     // 标记当前位置为已访问
23     visited[i][j] = true;
24
25     // 从当前位置开始, 局域网的大小至少为1
26     int count = 1;
27
28     // 递归地搜索上下左右四个方向, 并累加局域网的大小
29     count += dfs(i - 1, j);
30     count += dfs(i + 1, j);
31     count += dfs(i, j - 1);
32     count += dfs(i, j + 1);
33
34     // 返回局域网的总大小
35     return count;
36 }
37
38 int main() {
39     // 读取矩阵的行数和列数
40     scanf("%d %d", &n, &m);
41
42     // 读取服务器矩阵
43     for (int i = 0; i < n; i++) {
44         for (int j = 0; j < m; j++) {
45             scanf("%d", &server[i][j]);
46             visited[i][j] = false; // 初始化访问状态为未访问
47         }
48     }
49
50 }
```

```
50
51     int ans = 0; // 存储最大局域网的大小
52
53     // 遍历每个位置, 使用DFS寻找最大的局域网
54     for (int i = 0; i < n; i++) {
55         for (int j = 0; j < m; j++) {
56             // 只有在当前位置有服务器且未访问过时才进行DFS搜索
57             if (server[i][j] == 1 && !visited[i][j]) {
58                 int currentNetworkSize = dfs(i, j);
59                 if (currentNetworkSize > ans) {
60                     ans = currentNetworkSize; // 更新最大局域网的大小
61                 }
62             }
63         }
64     }
65
66     // 输出最大局域网包含的服务器个数
67     printf("%d\n", ans);
68
69     return 0;
}
```

完整用例

用例1

1	2	2
2	1	0
3	1	1

用例2

1	3	3	
2	1	0	1
3	0	1	0
4	1	0	1

用例3

1	4 4
2	1 0 0 1
3	0 1 1 0
4	0 1 1 0
5	1 0 0 1

用例4

1	5 5
2	1 0 0 1 0
3	0 1 1 0 1
4	0 1 1 0 1
5	1 0 0 1 0
6	0 1 1 0 1

用例5

1	3 4
2	1 0 1 1
3	0 1 1 0
4	1 0 0 1

用例6

1	10 10
2	1 0 0 1 0 0 0 0 0 0
3	0 1 1 0 1 0 1 1 1 0
4	0 1 1 0 1 0 1 1 1 0
5	1 0 0 1 0 0 0 0 0 0
6	0 1 1 0 1 0 1 1 1 0
7	0 1 1 0 1 0 1 1 1 0
8	1 0 0 1 0 0 0 0 0 0
9	0 1 1 0 1 0 1 1 1 0
10	0 1 1 0 1 0 1 1 1 0
11	1 0 0 1 0 0 0 0 0 0

用例7

1	6	8
2	1	0 0 1 0 0 0 0
3	0	1 1 0 1 0 1 1
4	0	1 1 0 1 0 1 1
5	1	0 0 1 0 0 0 0
6	0	1 1 0 1 0 1 1
7	1	1 1 0 1 0 1 1

用例8

1	7	7
2	1	0 0 1 0 0 0
3	0	1 1 0 1 0 1
4	0	1 1 0 1 0 1
5	1	0 0 1 0 0 0
6	0	1 1 0 1 0 1
7	1	1 1 0 1 0 1
8	1	1 1 0 1 0 1

用例9

1	4	5
2	1	0 0 1 0
3	0	1 1 0 1
4	0	1 1 0 1
5	1	0 0 1 0

用例10

1	5	4
2	1	0 1 1
3	0	1 1 0
4	1	0 0 1
5	0	1 1 0
6	1	0 0 1

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

题目描述

输入描述

输出描述

用例1

C++

Java

javaScript

python

C语言

完整用例

用例1

用例2

用例3

用例4

用例5

用例6

用例7

用例8

用例9

用例10

机考真题 华为OD



CSDN @算法大师