【华为OD机考 统一考试机试C卷】 欢乐的周末（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

小华和小为是很要好的朋友，他们约定周末一起吃饭。
通过手机交流，他们在地图上选择了多个聚餐地点（由于自然地形等原因，部分聚餐地点不可达），求小华和小为都能到达的聚餐地点有多少个?

输入描述

第一行输入m和n，m代表地图的长度，n代表地图的宽度。

第二行开始具体输入地图信息，地图信息包含:

0 为通畅的道路

1 为障碍物（且仅1为障碍物）

2 为小华或者小为，地图中必定有且仅有2个 （非障碍物）

3 为被选中的聚餐地点（非障碍物）

输出描述

可以被两方都到达的聚餐地点数量，行末无空格。

用例

输入	4 4 2 1 0 3 0 1 2 1 0 3 0 0 0 0 0 0
输出	2
说明	第一行输入地图的长宽为3和4。 第二行开始为具体的地图，其中：3代表小华和小明选择的聚餐地点；2代表小华或者小明（确保有2个）；0代表可以通行的位置；1代表不可以通行的位置。 此时两者能都能到达的聚餐位置有2处。

输入	4 4 2 1 2 3 0 1 0 0 0 1 0 0 0 1 0 0
输出	0
说明	第一行输入地图的长宽为4和4。 第二行开始为具体的地图，其中：3代表小华和小明选择的聚餐地点；2代表小华或者小明（确保有2个）；0代表可以通行的位置；1代表不可以通行的位置。 由于图中小华和小为之间有个阻隔，此时，没有两人都能到达的聚餐地址，故而返回0。

备注：

地图的长宽为m和n，其中：

4 <= m <= 100

4 <= n <= 100

聚餐的地点数量为 k，则

1< k <= 100

C++

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

using namespace std;

// 定义四个方向的偏移量（上、下、左、右）
const int dirs[4][2] = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};

// 深度优先搜索函数
bool dfs(int currX, int currY, int targetX, int targetY, vector<vector<int>>& map, vector<vector<vector<bool>>>& visited, int person) {
    // 如果当前位置就是目标位置，返回true
    if (currX == targetX && currY == targetY) {
        return true;
    }

    // 遍历四个方向
    for (int i = 0; i < 4; ++i) {
        int nextX = currX + dirs[i][0], nextY = currY + dirs[i][1];
        // 如果下一个位置超出地图范围，或者是障碍物，或者已经访问过，跳过
        if (nextX < 0 || nextY < 0 || nextX >= map.size() || nextY >= map[0].size() || map[nextX][nextY] == 1 || visited[nextX][nextY][person]) {
            continue;
        }

        // 标记下一个位置为已访问
        visited[nextX][nextY][person] = true;
        // 递归搜索下一个位置
        if (dfs(nextX, nextY, targetX, targetY, map, visited, person)) {
            return true;
        }
    }

    return false;
}

```

```cpp
int main() {
    // 输入初始化
    int m, n;
    cin >> m >> n;
    vector<vector<int>> map(m, vector<int>(n));
    // 使用三维数组visited来记录每个人访问过的位置
    vector<vector<vector<bool>>> visited(m, vector<vector<bool>>(n, vector<bool>(2)));
    vector<pair<int, int>> persons, targets;

    // 读取地图信息，并记录小华和小为的位置以及聚餐地点
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> map[i][j];
            if (map[i][j] == 2) {
                persons.emplace_back(i, j);
            } else if (map[i][j] == 3) {
                targets.emplace_back(i, j);
            }
        }
    }

    // 获取小华和小为的位置
    auto xiaohua = persons[0];
    auto xiaowei = persons[1];
    int res = 0;

    // 遍历所有聚餐地点
    for (const auto& target : targets) {
        // 重置visited数组
        visited = vector<vector<vector<bool>>>(m, vector<vector<bool>>(n, vector<bool>(2)));
        // 判断小华是否能到达目标位置
        if (dfs(xiaohua.first, xiaohua.second, target.first, target.second, map, visited, 0)) {
            // 重置visited数组
            visited = vector<vector<vector<bool>>>(m, vector<vector<bool>>(n, vector<bool>(2)));
            // 判断小为是否能到达目标位置
            if (dfs(xiaowei.first, xiaowei.second, target.first, target.second, map, visited, 1)) {
                // 如果两个人都能到达目标位置，结果加1
                res++;
            }
        }
```

```
    }

    // 输出可以被两人都到达的聚餐地点数量
    cout << res << endl;

    return 0;
}
```

JavaScript

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

// 定义四个方向的偏移量（上、下、左、右）
const dirs = [[-1, 0], [1, 0], [0, 1], [0, -1]];

// 深度优先搜索函数
function dfs(currX, currY, targetX, targetY, map, visited, person) {
    // 如果当前位置就是目标位置，返回true
    if (currX === targetX && currY === targetY) {
        return true;
    }

    // 遍历四个方向
    for (const dir of dirs) {
        const nextX = currX + dir[0], nextY = currY + dir[1];
        // 如果下一个位置超出地图范围，或者是障碍物，或者已经访问过，跳过
        if (nextX < 0 || nextY < 0 || nextX >= map.length || nextY >= map[0].length || map[nextX][nextY] === 1 || visited[nextX][nextY][person]) {
            continue;
        }

        // 标记下一个位置为已访问
        visited[nextX][nextY][person] = true;
        // 递归搜索下一个位置
        if (dfs(nextX, nextY, targetX, targetY, map, visited, person)) {
            return true;
        }
    }

    return false;
```

```
}

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

const input = [];

rl.on('line', (line) => {
    input.push(line);
}).on('close', () => {
    // 输入初始化
    const [m, n] = input.shift().split(' ').map(Number);
    const map = input.splice(0, m).map(row => row.split(' ').map(Number));
    const visited = Array.from({ length: m }, () => Array.from({ length: n }, () => Array(2).fill(false)));
    const persons = [];
    const targets = [];

    // 读取地图信息，并记录小华和小为的位置以及聚餐地点
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            if (map[i][j] === 2) {
                persons.push([i, j]);
            } else if (map[i][j] === 3) {
                targets.push([i, j]);
            }
        }
    }

    // 获取小华和小为的位置
    const xiaohua = persons[0];
    const xiaowei = persons[1];
    let res = 0;

    // 遍历所有聚餐地点
    for (const target of targets) {
        // 重置visited数组
        visited.forEach(row => row.forEach(cell => cell.fill(false)));
        // 判断小华是否能到达目标位置
```

```
        if (dfs(xiaohua[0], xiaohua[1], target[0], target[1], map, visited, 0)) {
            // 重置visited数组
            visited.forEach(row => row.forEach(cell => cell.fill(false)));
            // 判断小为是否能到达目标位置
            if (dfs(xiaowei[0], xiaowei[1], target[0], target[1], map, visited, 1)) {
                // 如果两个人都能到达目标位置，结果加1
                res++;
            }
        }
    }

    // 输出可以被两人都到达的聚餐地点数量
    console.log(res);
});
```

Java

```
import java.util.Scanner;
import java.util.ArrayList;
import java.util.List;

public class Main {
    // 定义四个方向的偏移量（上、下、左、右）
    private static int[][] dirs = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};

    // 深度优先搜索函数
    private static boolean dfs(int currX, int currY, int targetX, int targetY, int[][] map, boolean[][][] visited, int person) {
        // 如果当前位置就是目标位置，返回true
        if (currX == targetX && currY == targetY) {
            return true;
        }

        // 遍历四个方向
        for (int[] dir : dirs) {
            int nextX = currX + dir[0], nextY = currY + dir[1];
            // 如果下一个位置超出地图范围，或者是障碍物，或者已经访问过，跳过
            if (nextX < 0 || nextY < 0 || nextX >= map.length || nextY >= map[0].length || map[nextX][nextY] == 1 || visited[nextX][nextY][person]) {
                continue;
            }
```

```
            // 标记下一个位置为已访问
            visited[nextX][nextY][person] = true;
            // 递归搜索下一个位置
            if (dfs(nextX, nextY, targetX, targetY, map, visited, person)) {
                return true;
            }
        }

        return false;
    }

    public static void main(String[] args) {
        // 输入初始化
        Scanner scanner = new Scanner(System.in);
        int m = scanner.nextInt();
        int n = scanner.nextInt();
        int[][] map = new int[m][n];
        // 使用三维数组visited来记录每个人访问过的位置
        boolean[][][] visited = new boolean[m][n][2];
        List<int[]> persons = new ArrayList<>();
        List<int[]> targets = new ArrayList<>();
        // 读取地图信息，并记录小华和小为的位置以及聚餐地点
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                map[i][j] = scanner.nextInt();
                if (map[i][j] == 2) {
                    persons.add(new int[]{i, j});
                } else if (map[i][j] == 3) {
                    targets.add(new int[]{i, j});
                }
            }
        }

        // 获取小华和小为的位置
        int[] xiaohua = persons.get(0);
        int[] xiaowei = persons.get(1);
        int res = 0;
        // 遍历所有聚餐地点
        for (int[] target : targets) {
```

```
            // 重置visited数组
            visited = new boolean[m][n][2];
            // 判断小华是否能到达目标位置
            if (dfs(xiaohua[0], xiaohua[1], target[0], target[1], map, visited, 0)) {
                // 重置visited数组
                visited = new boolean[m][n][2];
                // 判断小为是否能到达目标位置
                if (dfs(xiaowei[0], xiaowei[1], target[0], target[1], map, visited, 1)) {
                    // 如果两个人都能到达目标位置，结果加1
                    res++;
                }
            }
        }
        // 输出可以被两人都到达的聚餐地点数量
        System.out.println(res);

        scanner.close();
    }
}
```

Python

```
# 定义四个方向的偏移量（上、下、左、右）
dirs = [[-1, 0], [1, 0], [0, 1], [0, -1]]

# 深度优先搜索函数
def dfs(currX, currY, targetX, targetY, map, visited, person):
    # 如果当前位置就是目标位置，返回True
    if currX == targetX and currY == targetY:
        return True

    # 遍历四个方向
    for dir in dirs:
        nextX, nextY = currX + dir[0], currY + dir[1]
        # 如果下一个位置超出地图范围，或者是障碍物，或者已经访问过，跳过
        if nextX < 0 or nextY < 0 or nextX >= len(map) or nextY >= len(map[0]) or map[nextX][nextY] == 1 or visited[nextX][nextY][person]:
            continue

        # 标记下一个位置为已访问
```

```
        visited[nextX][nextY][person] = True
        # 递归搜索下一个位置
        if dfs(nextX, nextY, targetX, targetY, map, visited, person):
            return True

    return False

# 输入初始化
m, n = map(int, input().split())
map = [list(map(int, input().split())) for _ in range(m)]
# 使用三维数组visited来记录每个人访问过的位置
visited = [[[False, False] for _ in range(n)] for _ in range(m)]
persons = []
targets = []

# 读取地图信息，并记录小华和小为的位置以及聚餐地点
for i in range(m):
    for j in range(n):
        if map[i][j] == 2:
            persons.append([i, j])
        elif map[i][j] == 3:
            targets.append([i, j])

# 获取小华和小为的位置
xiaohua = persons[0]
xiaowei = persons[1]
res = 0

# 遍历所有聚餐地点
for target in targets:
    # 重置visited数组
    visited = [[[False, False] for _ in range(n)] for _ in range(m)]
    # 判断小华是否能到达目标位置
    if dfs(xiaohua[0], xiaohua[1], target[0], target[1], map, visited, 0):
        # 重置visited数组
        visited = [[[False, False] for _ in range(n)] for _ in range(m)]
        # 判断小为是否能到达目标位置
        if dfs(xiaowei[0], xiaowei[1], target[0], target[1], map, visited, 1):
            # 如果两个人都能到达目标位置，结果加1
            res += 1

```

```
# 输出可以被两人都到达的聚餐地点数量
print(res)
```

C语言

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

#define MAX_SIZE 100 // 定义地图的最大尺寸

// 定义四个方向的偏移量（上、下、左、右）
const int dirs[4][2] = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}};

// 深度优先搜索函数
bool dfs(int currX, int currY, int targetX, int targetY, int m, int n, int map[MAX_SIZE][MAX_SIZE], bool visited[MAX_SIZE][MAX_SIZE], int person) {
    // 如果当前位置就是目标位置，返回true
    if (currX == targetX && currY == targetY) {
        return true;
    }

    // 遍历四个方向
    for (int i = 0; i < 4; ++i) {
        int nextX = currX + dirs[i][0], nextY = currY + dirs[i][1];
        // 如果下一个位置超出地图范围，或者是障碍物，或者已经访问过，跳过
        if (nextX < 0 || nextY < 0 || nextX >= m || nextY >= n || map[nextX][nextY] == 1 || visited[nextX][nextY]) {
            continue;
        }

        // 标记下一个位置为已访问
        visited[nextX][nextY] = true;
        // 递归搜索下一个位置
        if (dfs(nextX, nextY, targetX, targetY, m, n, map, visited, person)) {
            return true;
        }
    }

    return false;
}
```

```
int main() {
    int m, n;
    scanf("%d %d", &m, &n); // 读取地图的长和宽
    int map[MAX_SIZE][MAX_SIZE]; // 地图数组
    bool visited[MAX_SIZE][MAX_SIZE]; // 访问标记数组
    int persons[2][2], targets[MAX_SIZE][2]; // 人物和目标位置
    int personsCount = 0, targetsCount = 0;

    // 读取地图信息，并记录小华和小为的位置以及聚餐地点
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            scanf("%d", &map[i][j]);
            if (map[i][j] == 2) {
                persons[personsCount][0] = i;
                persons[personsCount][1] = j;
                personsCount++;
            } else if (map[i][j] == 3) {
                targets[targetsCount][0] = i;
                targets[targetsCount][1] = j;
                targetsCount++;
            }
        }
    }

    int res = 0; // 可以被两人都到达的聚餐地点数量

    // 遍历所有聚餐地点
    for (int i = 0; i < targetsCount; ++i) {
        bool canReach = true; // 标记是否都可以到达
        for (int p = 0; p < 2; ++p) { // 对于小华和小为
            // 重置visited数组
            memset(visited, 0, sizeof(visited));
            // 判断是否能到达目标位置
            if (!dfs(persons[p][0], persons[p][1], targets[i][0], targets[i][1], m, n, map, visited, p)) {
                canReach = false;
                break;
            }
        }
        if (canReach) {
```

```
            res++;
        }
    }

    printf("%d\n", res); // 输出结果

    return 0;
}
```

完整用例

用例1

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

用例2

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

用例3

3 3
2 1 0
0 1 3
0 0 2

用例4

3 3
2 0 0

1 1 0
0 0 2

用例5

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

用例6

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

用例7

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

用例8

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

用例9

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

用例10

6 6
2 1 0 0 0 3
0 1 2 1 0 0
0 3 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
3 0 0 0 0 0

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
- C++
- JavaScript
- Java
- 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