【华为OD机考 统一考试机试C卷】图像物体的边界（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

给定一个二维数组M行N列，二维数组里的数字代表图片的像素，为了简化问题，仅包含像素1和5两种像素，每种像素代表一个物体，2个物体相邻的格子为边界，求像素1代表的物体的边界个数。

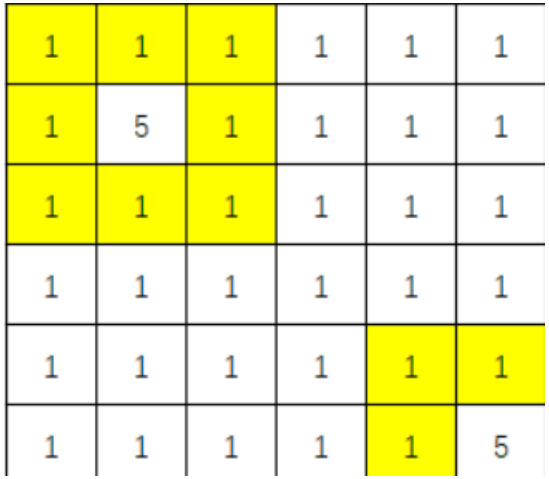
像素1代表的物体的边界指与像素5相邻的像素1的格子，边界相邻的属于同一个边界，相邻需要考虑8个方向（上，下，左，右，左上，左下，右上，右下）。

其他约束

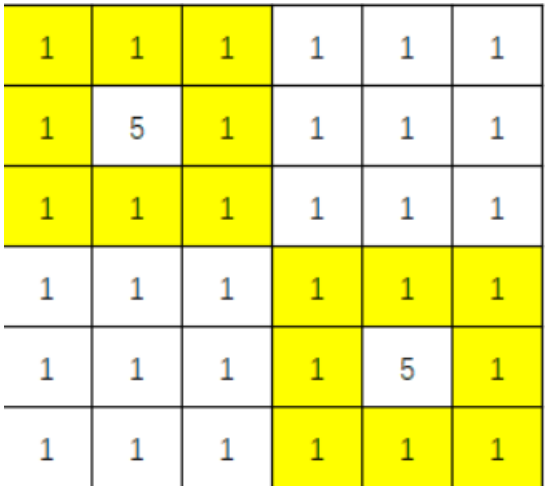
地图规格约束为：

0<M<100
0<N<100

1）如下图，与像素5的格子相邻的像素1的格子（0,0）、（0,1）、（0,2）、（1,0）、（1,2）、（2,0）、（2,1）、（2,2）、（4,4）、（4,5）、（5,4）为边界，另（0,0）、（0,1）、（0,2）、（1,0）、（1,2）、（2,0）、（2,1）、（2,2）相邻，为1个边界，（4,4）、（4,5）、（5,4）相邻，为1个边界，所以下图边界个数为2。



2）如下图，与像素5的格子相邻的像素1的格子（0,0）、（0,1）、（0,2）、（1,0）、（1,2）、（2,0）、（2,1）、（2,2）、（3,3）、（3,4）、（3,5）、（4,3）、（4,5）、（5,3）、（5,4）、（5,5）为边界，另这些边界相邻，所以下图边界个数为1。



注：（2,2）、（3,3）相邻。

输入描述

第一行，行数M，列数N

第二行开始，是M行N列的像素的二维数组，仅包含像素1和5

输出描述

像素1代表的物体的边界个数。

如果没有边界输出0（比如只存在像素1，或者只存在像素5）。

用例

输入	6 6 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5
输出	2
说明	参考题目描述部分

输入	6 6 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1
输出	1
说明	参考题目描述部分

解题思路

1. **定义方向数组**：设置两个数组 `dx` 和 `dy`，分别表示在二维数组中八个方向上的横纵坐标变化（上、下、左、右和四个对角方向）。
2. **深度优先搜索 (DFS) 函数**：定义 `dfs` 函数，用于深度优先搜索。这个函数会递归地在二维数组中移动，查找满足特定条件的边界单元。
    - **标记访问**：首先，将当前位置标记为已访问，防止重复搜索。

- **遍历方向**：然后，遍历八个方向。对于每个方向，计算新坐标。
- **边界条件检查**：如果新坐标在地图范围内，且对应的值为 1，且是边界（通过调用 `isBorder` 函数判断），且未被访问过，则递归地调用 `dfs` 函数。

3. **边界判断函数（`isBorder`）**：这个函数用于判断给定的坐标是否是边界。
   - **遍历方向**：遍历八个方向，对于每个方向，计算新坐标。
   - **判断边界**：如果新坐标在地图范围内，且对应的值为 5，则认为当前位置是边界。
4. **遍历地图**：遍历地图的每个单元。
   - 对于每个单元，检查是否满足以下条件：值为 1，是边界（通过 `isBorder` 判断），且未被访问过。
   - 如果满足条件，则从该单元开始执行深度优先搜索，并将 `count` 加 1。

C++

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

// 定义移动方向数组，表示八个方向上的横纵坐标变化
int dx[8] = {-1, -1, -1, 0, 1, 1, 1, 0};
int dy[8] = {-1, 0, 1, 1, 1, 0, -1, -1};
// 定义一个二维数组，用于记录某个位置是否被访问过
std::vector<std::vector<int>> visited;

// 声明深度优先搜索函数和判断边界的函数
void dfs(int x, int y, std::vector<std::vector<int>>& mp, int n, int m);
bool isBorder(int x, int y, std::vector<std::vector<int>>& mp, int n, int m);

int main() {
    // 使用cin读取输入
    int n, m;
    std::cin >> n >> m;
    // 初始化地图数组mp和访问记录数组visited
    std::vector<std::vector<int>> mp(n, std::vector<int>(m));
    visited.resize(n, std::vector<int>(m, 0));

    // 循环读取地图信息
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
```

```
            std::cin >> mp[i][j];
        }
    }

    // 初始化计数器，用于记录边界的数量
    int count = 0;
    // 遍历地图的每一个位置
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            // 如果当前位置是1，且是边界，且未被访问过，则进行深度优先搜索
            if (mp[i][j] == 1 && isBorder(i, j, mp, n, m) && visited[i][j] == 0) {
                dfs(i, j, mp, n, m);
                count++; // 每完成一次深度优先搜索，边界数量加1
            }
        }
    }

    // 输出边界的数量
    std::cout << count << std::endl;
    return 0;
}

// 深度优先搜索函数
void dfs(int x, int y, std::vector<std::vector<int>>& mp, int n, int m) {
    // 标记当前位置为已访问
    visited[x][y] = 1;
    // 遍历八个方向
    for (int i = 0; i < 8; i++) {
        // 计算移动后的新坐标
        int nx = x + dx[i];
        int ny = y + dy[i];
        // 检查新坐标是否在地图范围内，是否为1，是否是边界，是否未被访问过
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 1 && isBorder(nx, ny, mp, n, m) && visited[nx][ny] == 0) {
            // 递归进行深度优先搜索
            dfs(nx, ny, mp, n, m);
        }
    }
}

// 判断一个位置是否是边界的函数

```

```
bool isBorder(int x, int y, std::vector<std::vector<int>>& mp, int n, int m) {
    // 遍历八个方向
    for (int i = 0; i < 8; i++) {
        // 计算移动后的新坐标
        int nx = x + dx[i];
        int ny = y + dy[i];
        // 如果新坐标在地图范围内，且值为5，则当前位置是边界
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 5) {
            return true;
        }
    }
    // 如果所有方向都不满足边界条件，则返回false
    return false;
}
```

javaScript

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

// 定义移动方向数组，表示八个方向上的横纵坐标变化
const dx = [-1, -1, -1, 0, 1, 1, 1, 0];
const dy = [-1, 0, 1, 1, 1, 0, -1, -1];

// 深度优先搜索函数
function dfs(x, y, mp, n, m, visited) {
  // 标记当前位置为已访问
  visited[x][y] = true;
  // 遍历八个方向
  for (let i = 0; i < 8; i++) {
    // 计算移动后的新坐标
    const nx = x + dx[i];
    const ny = y + dy[i];
    // 检查新坐标是否在地图范围内，是否为1，是否是边界，是否未被访问过
    if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] === 1 && isBorder(nx, ny, mp, n, m) && !visited[nx][ny]) {
      // 递归进行深度优先搜索
      dfs(nx, ny, mp, n, m, visited);
```

```
    }
  }
}

// 判断一个位置是否是边界的函数
function isBorder(x, y, mp, n, m) {
  // 遍历八个方向
  for (let i = 0; i < 8; i++) {
    // 计算移动后的新坐标
    const nx = x + dx[i];
    const ny = y + dy[i];
    // 如果新坐标在地图范围内，且值为5，则当前位置是边界
    if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] === 5) {
      return true;
    }
  }
  // 如果所有方向都不满足边界条件，则返回false
  return false;
}

let lines = [];
let lineCount = 0;
let n, m, mp, visited, count = 0;

rl.on('line', (line) => {
  lines.push(line);
  lineCount++;
  if (lineCount === 1) {
    // 读取 n 和 m 的值
    [n, m] = lines[0].split(' ').map(Number);
  } else if (lineCount <= n + 1) {
    // 读取地图数据
    if (lineCount === 2) {
      mp = new Array(n);
      visited = new Array(n);
    }
    mp[lineCount - 2] = lines[lineCount - 1].split(' ').map(Number);
    visited[lineCount - 2] = new Array(m).fill(false);
  }
```

```
  if (lineCount === n + 1) {
    // 处理输入完成后的逻辑
    // 遍历地图的每一个位置
    for (let i = 0; i < n; i++) {
      for (let j = 0; j < m; j++) {
        // 如果当前位置是1，且是边界，且未被访问过，则进行深度优先搜索
        if (mp[i][j] === 1 && isBorder(i, j, mp, n, m) && !visited[i][j]) {
          dfs(i, j, mp, n, m, visited);
          count++; // 每完成一次深度优先搜索，边界数量加1
        }
      }
    }
    // 输出边界的数量
    console.log(count);
    // 关闭 readline 接口
    rl.close();
  }
});
```

java

```
import java.util.Scanner;

public class Main {
    // 定义移动方向数组，表示八个方向上的横纵坐标变化
    static int[] dx = {-1, -1, -1, 0, 1, 1, 1, 0};
    static int[] dy = {-1, 0, 1, 1, 1, 0, -1, -1};
    // 定义一个二维数组，用于记录某个位置是否被访问过
    static int[][] visited;

    public static void main(String[] args) {
        // 使用Scanner类读取输入
        Scanner scanner = new Scanner(System.in);
        // 读取行数n和列数m
        int n = scanner.nextInt();
        int m = scanner.nextInt();
        // 初始化地图数组mp和访问记录数组visited
        int[][] mp = new int[n][m];
        visited = new int[n][m];

```

```
        // 循环读取地图信息
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                mp[i][j] = scanner.nextInt();
            }
        }

        // 初始化计数器，用于记录边界的数量
        int count = 0;
        // 遍历地图的每一个位置
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                // 如果当前位置是1，且是边界，且未被访问过，则进行深度优先搜索
                if (mp[i][j] == 1 && isBorder(i, j, mp, n, m) && visited[i][j] == 0) {
                    dfs(i, j, mp, n, m);
                    count++; // 每完成一次深度优先搜索，边界数量加1
                }
            }
        }

        // 输出边界的数量
        System.out.println(count);
    }

    // 深度优先搜索函数
    static void dfs(int x, int y, int[][] mp, int n, int m) {
        // 标记当前位置为已访问
        visited[x][y] = 1;
        // 遍历八个方向
        for (int i = 0; i < 8; i++) {
            // 计算移动后的新坐标
            int nx = x + dx[i];
            int ny = y + dy[i];
            // 检查新坐标是否在地图范围内，是否为1，是否是边界，是否未被访问过
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 1 && isBorder(nx, ny, mp, n, m) && visited[nx][ny] == 0) {
                // 递归进行深度优先搜索
                dfs(nx, ny, mp, n, m);
            }
        }
    }
```

```
    // 判断一个位置是否是边界的函数
    static boolean isBorder(int x, int y, int[][] mp, int n, int m) {
        // 遍历八个方向
        for (int i = 0; i < 8; i++) {
            // 计算移动后的新坐标
            int nx = x + dx[i];
            int ny = y + dy[i];
            // 如果新坐标在地图范围内，且值为5，则当前位置是边界
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 5) {
                return true;
            }
        }
        // 如果所有方向都不满足边界条件，则返回false
        return false;
    }
}
```

python

```
# 定义移动方向数组，表示八个方向上的横纵坐标变化
dx = [-1, -1, -1, 0, 1, 1, 1, 0]
dy = [-1, 0, 1, 1, 1, 0, -1, -1]

# 深度优先搜索函数
def dfs(x, y, mp, n, m, visited):
    # 标记当前位置为已访问
    visited[x][y] = True
    # 遍历八个方向
    for i in range(8):
        # 计算移动后的新坐标
        nx, ny = x + dx[i], y + dy[i]
        # 检查新坐标是否在地图范围内，是否为1，是否是边界，是否未被访问过
        if 0 <= nx < n and 0 <= ny < m and mp[nx][ny] == 1 and is_border(nx, ny, mp, n, m) and not visited[nx][ny]:
            # 递归进行深度优先搜索
            dfs(nx, ny, mp, n, m, visited)

# 判断一个位置是否是边界的函数
def is_border(x, y, mp, n, m):
    # 遍历八个方向
```

```
    for i in range(8):
        # 计算移动后的新坐标
        nx, ny = x + dx[i], y + dy[i]
        # 如果新坐标在地图范围内，且值为5，则当前位置是边界
        if 0 <= nx < n and 0 <= ny < m and mp[nx][ny] == 5:
            return True
    # 如果所有方向都不满足边界条件，则返回False
    return False

# 读取输入
n, m = map(int, input().split())
mp = [list(map(int, input().split())) for _ in range(n)]
visited = [[False] * m for _ in range(n)]

# 初始化计数器，用于记录边界的数量
count = 0
# 遍历地图的每一个位置
for i in range(n):
    for j in range(m):
        # 如果当前位置是1，且是边界，且未被访问过，则进行深度优先搜索
        if mp[i][j] == 1 and is_border(i, j, mp, n, m) and not visited[i][j]:
            dfs(i, j, mp, n, m, visited)
            count += 1  # 每完成一次深度优先搜索，边界数量加1

# 输出边界的数量
print(count)
```

C语言

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

// 定义移动方向数组，表示八个方向上的横纵坐标变化
int dx[] = {-1, -1, -1, 0, 1, 1, 1, 0};
int dy[] = {-1, 0, 1, 1, 1, 0, -1, -1};

// 函数声明
void dfs(int x, int y, int mp[][10], int n, int m, bool visited[][10]);
bool is_border(int x, int y, int mp[][10], int n, int m);

```

```
// 深度优先搜索函数
void dfs(int x, int y, int mp[][10], int n, int m, bool visited[][10]) {
    // 标记当前位置为已访问
    visited[x][y] = true;
    // 遍历八个方向
    for (int i = 0; i < 8; i++) {
        // 计算移动后的新坐标
        int nx = x + dx[i];
        int ny = y + dy[i];
        // 检查新坐标是否在地图范围内，是否为1，是否是边界，是否未被访问过
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 1 && is_border(nx, ny, mp, n, m) && !visited[nx][ny]) {
            // 递归进行深度优先搜索
            dfs(nx, ny, mp, n, m, visited);
        }
    }
}

// 判断一个位置是否是边界的函数
bool is_border(int x, int y, int mp[][10], int n, int m) {
    // 遍历八个方向
    for (int i = 0; i < 8; i++) {
        // 计算移动后的新坐标
        int nx = x + dx[i];
        int ny = y + dy[i];
        // 如果新坐标在地图范围内，且值为5，则当前位置是边界
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 5) {
            return true;
        }
    }
    // 如果所有方向都不满足边界条件，则返回false
    return false;
}

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    int mp[10][10];
    bool visited[10][10] = {false};

    // 读取地图信息
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            scanf("%d", &mp[i][j]);
        }
    }

    // 初始化计数器，用于记录边界的数量
    int count = 0;
    // 遍历地图的每一个位置
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            // 如果当前位置是1，且是边界，且未被访问过，则进行深度优先搜索
            if (mp[i][j] == 1 && is_border(i, j, mp, n, m) && !visited[i][j]) {
                dfs(i, j, mp, n, m, visited);
                count++;  // 每完成一次深度优先搜索，边界数量加1
            }
        }
    }

    // 输出边界的数量
    printf("%d\n", count);
    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师