

【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。

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

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。

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

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	<div>6 6</div> <div>1 1 1 1 1 1</div> <div>1 5 1 1 1 1</div> <div>1 1 1 1 1 1</div> <div>1 1 1 1 1 1</div> <div>1 1 1 1 1 1</div> <div>1 1 1 1 1 5</div>
输出	2
说明	参考题目描述部分

输入	<div>6 6</div> <div>1 1 1 1 1 1</div> <div>1 5 1 1 1 1</div> <div>1 1 1 1 1 1</div> <div>1 1 1 1 1 1</div> <div>1 1 1 1 5 1</div> <div>1 1 1 1 1 1</div>
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++

```
1  #include <iostream>
2  #include <vector>
3
4  // 定义移动方向数组，表示八个方向上的横纵坐标变化
5  int dx[8] = {-1, -1, -1, 0, 1, 1, 1, 0};
6  int dy[8] = {-1, 0, 1, 1, 1, 0, -1, -1};
7  // 定义一个二维数组，用于记录某个位置是否被访问过
8  std::vector<std::vector<int>> visited;
9
10 // 声明深度优先搜索函数和判断边界的函数
11 void dfs(int x, int y, std::vector<std::vector<int>>& mp, int n, int m);
12 bool isBorder(int x, int y, std::vector<std::vector<int>>& mp, int n, int m);
13
14 int main() {
15     // 使用cin读取输入
16     int n, m;
17     std::cin >> n >> m;
18     // 初始化地图数组mp和访问记录数组visited
19     std::vector<std::vector<int>> mp(n, std::vector<int>(m));
20     visited.resize(n, std::vector<int>(m, 0));
21
22     // 循环读取地图信息
23     for (int i = 0; i < n; i++) {
24         for (int j = 0; j < m; j++) {
```

```
25         std::cin >> mp[i][j];
26     }
27 }
28
29 // 初始化计数器, 用于记录边界的数量
30 int count = 0;
31 // 遍历地图的每一个位置
32 for (int i = 0; i < n; i++) {
33     for (int j = 0; j < m; j++) {
34         // 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
35         if (mp[i][j] == 1 && isBorder(i, j, mp, n, m) && visited[i][j] == 0) {
36             dfs(i, j, mp, n, m);
37             count++; // 每完成一次深度优先搜索, 边界数量加1
38         }
39     }
40 }
41
42 // 输出边界的数量
43 std::cout << count << std::endl;
44 return 0;
45 }
46
47 // 深度优先搜索函数
48 void dfs(int x, int y, std::vector<std::vector<int>>& mp, int n, int m) {
49     // 标记当前位置为已访问
50     visited[x][y] = 1;
51     // 遍历八个方向
52     for (int i = 0; i < 8; i++) {
53         // 计算移动后的新坐标
54         int nx = x + dx[i];
55         int ny = y + dy[i];
56         // 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
57         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 1 && isBorder(nx, ny, mp, n, m) && visited[nx][ny] == 0) {
58             // 递归进行深度优先搜索
59             dfs(nx, ny, mp, n, m);
60         }
61     }
62 }
63
64 // 判断一个位置是否是边界的函数
65
```

```
66 bool isBorder(int x, int y, std::vector<std::vector<int>>& mp, int n, int m) {
67     // 遍历八个方向
68     for (int i = 0; i < 8; i++) {
69         // 计算移动后的新坐标
70         int nx = x + dx[i];
71         int ny = y + dy[i];
72         // 如果新坐标在地图范围内, 且值为5, 则当前位置是边界
73         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 5) {
74             return true;
75         }
76     }
77     // 如果所有方向都不满足边界条件, 则返回false
78     return false;
}
```

javaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({
3      input: process.stdin,
4      output: process.stdout
5  });
6
7  // 定义移动方向数组, 表示八个方向上的横纵坐标变化
8  const dx = [-1, -1, -1, 0, 1, 1, 1, 0];
9  const dy = [-1, 0, 1, 1, 1, 0, -1, -1];
10
11 // 深度优先搜索函数
12 function dfs(x, y, mp, n, m, visited) {
13     // 标记当前位置为已访问
14     visited[x][y] = true;
15     // 遍历八个方向
16     for (let i = 0; i < 8; i++) {
17         // 计算移动后的新坐标
18         const nx = x + dx[i];
19         const ny = y + dy[i];
20         // 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
21         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] === 1 && isBorder(nx, ny, mp, n, m) && !visited[nx][ny]) {
22             // 递归进行深度优先搜索
23             dfs(nx, ny, mp, n, m, visited);
24         }
25     }
26 }
```

```
24     }
25   }
26 }
27
28 // 判断一个位置是否是边界的函数
29 function isBorder(x, y, mp, n, m) {
30   // 遍历八个方向
31   for (let i = 0; i < 8; i++) {
32     // 计算移动后的新坐标
33     const nx = x + dx[i];
34     const ny = y + dy[i];
35     // 如果新坐标在地图范围内, 且值为5, 则当前位置是边界
36     if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] === 5) {
37       return true;
38     }
39   }
40   // 如果所有方向都不满足边界条件, 则返回false
41   return false;
42 }
43
44 let lines = [];
45 let lineCount = 0;
46 let n, m, mp, visited, count = 0;
47
48 rl.on('line', (line) => {
49   lines.push(line);
50   lineCount++;
51   if (lineCount === 1) {
52     // 读取 n 和 m 的值
53     [n, m] = lines[0].split(' ').map(Number);
54   } else if (lineCount <= n + 1) {
55     // 读取地图数据
56     if (lineCount === 2) {
57       mp = new Array(n);
58       visited = new Array(n);
59     }
60     mp[lineCount - 2] = lines[lineCount - 1].split(' ').map(Number);
61     visited[lineCount - 2] = new Array(m).fill(false);
62   }
63 }
64
--
```



```
65 if (lineCount === n + 1) {
66     // 处理输入完成后的逻辑
67     // 遍历地图的每一个位置
68     for (let i = 0; i < n; i++) {
69         for (let j = 0; j < m; j++) {
70             // 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
71             if (mp[i][j] === 1 && isBorder(i, j, mp, n, m) && !visited[i][j]) {
72                 dfs(i, j, mp, n, m, visited);
73                 count++; // 每完成一次深度优先搜索, 边界数量加1
74             }
75         }
76     }
77     // 输出边界的数量
78     console.log(count);
79     // 关闭 readline 接口
80     rl.close();
81 }
});
```

java

```
1 import java.util.Scanner;
2
3 public class Main {
4     // 定义移动方向数组, 表示八个方向上的横纵坐标变化
5     static int[] dx = {-1, -1, -1, 0, 1, 1, 1, 0};
6     static int[] dy = {-1, 0, 1, 1, 1, 0, -1, -1};
7     // 定义一个二维数组, 用于记录某个位置是否被访问过
8     static int[][] visited;
9
10    public static void main(String[] args) {
11        // 使用Scanner类读取输入
12        Scanner scanner = new Scanner(System.in);
13        // 读取行数n和列数m
14        int n = scanner.nextInt();
15        int m = scanner.nextInt();
16        // 初始化地图数组mp和访问记录数组visited
17        int[][] mp = new int[n][m];
18        visited = new int[n][m];
19
20    }
```

```
20 // 循环读取地图信息
21 for (int i = 0; i < n; i++) {
22     for (int j = 0; j < m; j++) {
23         mp[i][j] = scanner.nextInt();
24     }
25 }
26
27 // 初始化计数器, 用于记录边界的数量
28 int count = 0;
29 // 遍历地图的每一个位置
30 for (int i = 0; i < n; i++) {
31     for (int j = 0; j < m; j++) {
32         // 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
33         if (mp[i][j] == 1 && isBorder(i, j, mp, n, m) && visited[i][j] == 0) {
34             dfs(i, j, mp, n, m);
35             count++; // 每完成一次深度优先搜索, 边界数量加1
36         }
37     }
38 }
39
40 // 输出边界的数量
41 System.out.println(count);
42 }
43
44 // 深度优先搜索函数
45 static void dfs(int x, int y, int[][] mp, int n, int m) {
46     // 标记当前位置为已访问
47     visited[x][y] = 1;
48     // 遍历八个方向
49     for (int i = 0; i < 8; i++) {
50         // 计算移动后的新坐标
51         int nx = x + dx[i];
52         int ny = y + dy[i];
53         // 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
54         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 1 && isBorder(nx, ny, mp, n, m) && visited[nx][ny] == 0) {
55             // 递归进行深度优先搜索
56             dfs(nx, ny, mp, n, m);
57         }
58     }
59 }
60 }
```

```

61
62 // 判断一个位置是否是边界的函数
63 static boolean isBorder(int x, int y, int[][] mp, int n, int m) {
64     // 遍历八个方向
65     for (int i = 0; i < 8; i++) {
66         // 计算移动后的新坐标
67         int nx = x + dx[i];
68         int ny = y + dy[i];
69         // 如果新坐标在地图范围内, 且值为5, 则当前位置是边界
70         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 5) {
71             return true;
72         }
73     }
74     // 如果所有方向都不满足边界条件, 则返回false
75     return false;
76 }
}

```

python

```

1 # 定义移动方向数组, 表示八个方向上的横纵坐标变化
2 dx = [-1, -1, -1, 0, 1, 1, 1, 0]
3 dy = [-1, 0, 1, 1, 1, 0, -1, -1]
4
5 # 深度优先搜索函数
6 def dfs(x, y, mp, n, m, visited):
7     # 标记当前位置为已访问
8     visited[x][y] = True
9     # 遍历八个方向
10    for i in range(8):
11        # 计算移动后的新坐标
12        nx, ny = x + dx[i], y + dy[i]
13        # 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
14        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]:
15            # 递归进行深度优先搜索
16            dfs(nx, ny, mp, n, m, visited)
17
18 # 判断一个位置是否是边界的函数
19 def is_border(x, y, mp, n, m):
20     # 遍历八个方向
21

```

```

21     for i in range(8):
22         # 计算移动后的新坐标
23         nx, ny = x + dx[i], y + dy[i]
24         # 如果新坐标在地图范围内, 且值为5, 则当前位置是边界
25         if 0 <= nx < n and 0 <= ny < m and mp[nx][ny] == 5:
26             return True
27         # 如果所有方向都不满足边界条件, 则返回False
28     return False
29
30 # 读取输入
31 n, m = map(int, input().split())
32 mp = [list(map(int, input().split())) for _ in range(n)]
33 visited = [[False] * m for _ in range(n)]
34
35 # 初始化计数器, 用于记录边界的数量
36 count = 0
37 # 遍历地图的每一个位置
38 for i in range(n):
39     for j in range(m):
40         # 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
41         if mp[i][j] == 1 and is_border(i, j, mp, n, m) and not visited[i][j]:
42             dfs(i, j, mp, n, m, visited)
43             count += 1 # 每完成一次深度优先搜索, 边界数量加1
44
45 # 输出边界的数量
46 print(count)

```

C语言

```

1  #include <stdio.h>
2  #include <stdbool.h>
3
4  // 定义移动方向数组, 表示八个方向上的横纵坐标变化
5  int dx[] = {-1, -1, -1, 0, 1, 1, 1, 0};
6  int dy[] = {-1, 0, 1, 1, 1, 0, -1, -1};
7
8  // 函数声明
9  void dfs(int x, int y, int mp[][10], int n, int m, bool visited[][10]);
10 bool is_border(int x, int y, int mp[][10], int n, int m);
11
12

```

```
12 // 深度优先搜索函数
13 void dfs(int x, int y, int mp[][10], int n, int m, bool visited[][10]) {
14     // 标记当前位置为已访问
15     visited[x][y] = true;
16     // 遍历八个方向
17     for (int i = 0; i < 8; i++) {
18         // 计算移动后的新坐标
19         int nx = x + dx[i];
20         int ny = y + dy[i];
21         // 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
22         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 1 && is_border(nx, ny, mp, n, m) && !visited[nx][ny]) {
23             // 递归进行深度优先搜索
24             dfs(nx, ny, mp, n, m, visited);
25         }
26     }
27 }
28
29 // 判断一个位置是否是边界的函数
30 bool is_border(int x, int y, int mp[][10], int n, int m) {
31     // 遍历八个方向
32     for (int i = 0; i < 8; i++) {
33         // 计算移动后的新坐标
34         int nx = x + dx[i];
35         int ny = y + dy[i];
36         // 如果新坐标在地图范围内, 且值为5, 则当前位置是边界
37         if (nx >= 0 && nx < n && ny >= 0 && ny < m && mp[nx][ny] == 5) {
38             return true;
39         }
40     }
41     // 如果所有方向都不满足边界条件, 则返回false
42     return false;
43 }
44
45 int main() {
46     int n, m;
47     scanf("%d %d", &n, &m);
48     int mp[10][10];
49     bool visited[10][10] = {false};
50
51     // 读取地图信息
52     --
```

```
53     for (int i = 0; i < n; i++) {
54         for (int j = 0; j < m; j++) {
55             scanf("%d", &mp[i][j]);
56         }
57     }
58
59     // 初始化计数器, 用于记录边界的数量
60     int count = 0;
61     // 遍历地图的每一个位置
62     for (int i = 0; i < n; i++) {
63         for (int j = 0; j < m; j++) {
64             // 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
65             if (mp[i][j] == 1 && is_border(i, j, mp, n, m) && !visited[i][j]) {
66                 dfs(i, j, mp, n, m, visited);
67                 count++; // 每完成一次深度优先搜索, 边界数量加1
68             }
69         }
70     }
71
72     // 输出边界的数量
73     printf("%d\n", count);
74     return 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](#)

[解题思路](#)

[C++](#)

[javaScript](#)

[java](#)

[python](#)

[C语言](#)

