

# 【DFS/BFS】-机器人活动区域

## 题目描述与示例

### 题目

现有一个机器人，可放置于  $M \times N$  的网格 `grid` 中任意位置，每个网格包含一个非负整数编号。当相邻网格的数字编号差值的绝对值小于等于 1 时，机器人可以在网格间移动。**求机器人可活动的最大范围对应的网格点数目。**

说明：

- 1. 网格左上角坐标为  $(0,0)$ ，右下角坐标为  $(m-1, n-1)$
- 2. 机器人只能在相邻网格间上下左右移动

### 输入

第 1 行输入为  $M$  和  $N$ ， $M$  表示网格的行数  $N$  表示网格的列数。之后  $M$  行表示网格数值，每行  $N$  个数值（数值大小用  $k$  表示），数值间用单个空格分隔，行首行尾无多余空格。

$M$ 、 $N$ 、 $k$  均为整数，且  $1 \leq M, N \leq 150$ ， $0 \leq k \leq 50$

### 输出

输出 1 行，包含 1 个数字，表示最大活动区域的网格点数目。

### 示例一

#### 输入

```
1 4 4
2 1 2 5 2
3 2 4 4 5
4 3 5 7 1
5 4 6 2 4
```

#### 输出

```
1 6
```

说明

如图所示，图中绿色区域，相邻网格差值绝对值都小于等于 1，且为最大区域，对应网格点数目为 6。



示例二

输入

```
1 2 3
2 1 3 5
3 4 1 3
```

输出

```
1 1
```

说明

任意两个相邻网格的差值绝对值都大于 1，机器人不能在网格间移动，只能在单个网格内活动。对应网格点数目为 1

解题思路

注意，本题和LC695. 岛屿的最大面积几乎完全一致，均需要计算最大连通块的面积。唯一的区别在于，本题的连通性需要通过两个数值之差的绝对值来进行判断。

因此，在近邻点是否能够进行进一步DFS/BFS判断的时候，其相关的条件语句条件应该修改为

```
1 if (0 <= nxt_x < n and 0 <= nxt_y < m and check_list[nxt_x][nxt_y] == 0 and
2     abs(grid[x][y] - grid[nxt_x][nxt_y]) <= 1):
3     pass
```

## 代码

### 解法一：BFS

#### Python

```
1 # 题目：2024E-机器人的活动区域
2 # 分值：200
3 # 作者：闭着眼睛学数理化
4 # 算法：BFS
5 # 代码看不懂的地方，请直接在群上提问
6
7 from collections import deque
8
9 # 表示四个方向的数组
10 DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1)]
11
12 # 输入行数、列数
13 n, m = map(int, input().split())
14 # 输入地图
15 grid = list()
16 for i in range(n):
17     row = list(map(int, input().split()))
18     grid.append(row)
19
20
21 # 答案变量，用于记录最大活动区域
22 ans = 0
23 # 用于检查的二维矩阵
```

```

24 # 0表示没检查过, 1表示检查过了
25 check_list = [[0] * m for _ in range(n)]
26
27 # 最外层的双重循环, 是用来找BFS的起始搜索位置的
28 for i in range(n):
29     for j in range(m):
30         # 找到一个尚未检查过的点(i,j), 那么可以进行BFS的搜索
31         if check_list[i][j] == 0:
32             # 初始化维护BFS过程的队列
33             q = deque()
34             q.append([i, j])
35             # 将起始点(i,j)标记为已检查过
36             check_list[i][j] = 1
37             # 初始化最大活动区域面积为0
38             area = 0
39             # 进行BFS, 当队列中还有元素时, 持续地进行搜索
40             while len(q) > 0:
41                 # 弹出队头元素, 为当前点
42                 x, y = q.popleft()
43                 # 更新当前BFS最大活动区域面积
44                 area += 1
45                 # 考虑上下左右的四个近邻点
46                 for dx, dy in DIRECTIONS:
47                     nxt_x, nxt_y = x+dx, y+dy
48                     # 若下一个点要加入队列, 应该满足以下三个条件:
49                     # 1. 没有越界
50                     # 2. grid[x][y]和grid[nxt_x][nxt_y]的差值的绝对值小于等于1
51                     # 3. 尚未被检查过
52                     if (0 <= nxt_x < n and 0 <= nxt_y < m and check_list[nxt_x]
[nxt_y] == 0 and
53                         abs(grid[x][y] - grid[nxt_x][nxt_y]) <= 1):
54                         q.append([nxt_x, nxt_y]) # 入队
55                         check_list[nxt_x][nxt_y] = 1 # 标记为已检查过
56
57             # BFS搜索完成, 更新最大活动区域面积
58             ans = max(ans, area)
59
60 print(ans)

```

## Java

```

1 import java.util.*;
2
3 public class Main {
4     static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

```

```

5
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8
9         int n = scanner.nextInt();
10        int m = scanner.nextInt();
11        int[][] grid = new int[n][m];
12
13        for (int i = 0; i < n; i++) {
14            for (int j = 0; j < m; j++) {
15                grid[i][j] = scanner.nextInt();
16            }
17        }
18
19        int ans = 0;
20        int[][] checkList = new int[n][m];
21
22        for (int i = 0; i < n; i++) {
23            for (int j = 0; j < m; j++) {
24                if (checkList[i][j] == 0) {
25                    Queue<int[]> q = new LinkedList<>();
26                    q.add(new int[]{i, j});
27                    checkList[i][j] = 1;
28                    int area = 0;
29
30                    while (!q.isEmpty()) {
31                        int[] point = q.poll();
32                        int x = point[0];
33                        int y = point[1];
34                        area++;
35
36                        for (int[] dir : DIRECTIONS) {
37                            int nx = x + dir[0];
38                            int ny = y + dir[1];
39
40                            if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
41                                checkList[nx][ny] == 0 && Math.abs(grid[x]
42                                    [y] - grid[nx][ny]) <= 1) {
43                                q.add(new int[]{nx, ny});
44                                checkList[nx][ny] = 1;
45                            }
46                        }
47
48                        ans = Math.max(ans, area);
49                    }
50                }

```

```

51     }
52
53     System.out.println(ans);
54 }
55 }
56

```

## C++

```

1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <cmath>
5  using namespace std;
6
7  int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
8
9  int main() {
10     int n, m;
11     cin >> n >> m;
12
13     vector<vector<int>> grid(n, vector<int>(m));
14     vector<vector<int>> checkList(n, vector<int>(m, 0));
15
16     for (int i = 0; i < n; i++) {
17         for (int j = 0; j < m; j++) {
18             cin >> grid[i][j];
19         }
20     }
21
22     int ans = 0;
23
24     for (int i = 0; i < n; i++) {
25         for (int j = 0; j < m; j++) {
26             if (checkList[i][j] == 0) {
27                 queue<pair<int, int>> q;
28                 q.push({i, j});
29                 checkList[i][j] = 1;
30                 int area = 0;
31
32                 while (!q.empty()) {
33                     pair<int, int> point = q.front();
34                     q.pop();
35                     int x = point.first;
36                     int y = point.second;

```

```

37         area++;
38
39         for (auto dir : DIRECTIONS) {
40             int nx = x + dir[0];
41             int ny = y + dir[1];
42
43             if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
44                 checkList[nx][ny] == 0 && abs(grid[x][y] - grid[nx]
45                 [ny]) <= 1) {
46                 q.push({nx, ny});
47                 checkList[nx][ny] = 1;
48             }
49         }
50
51         ans = max(ans, area);
52     }
53 }
54 }
55
56 cout << ans << endl;
57
58 return 0;
59 }
60

```

## 解法二：DFS

### Python

```

1  # 题目：2024E-机器人的活动区域
2  # 分值：200
3  # 作者：闭着眼睛学数理化
4  # 算法：DFS
5  # 代码看不懂的地方，请直接在群上提问
6
7  # 表示四个方向的数组
8  DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1)]
9
10 # 输入行数、列数
11 n, m = map(int, input().split())
12 # 输入地图
13 grid = list()

```

```

14 for i in range(n):
15     row = list(map(int, input().split()))
16     grid.append(row)
17
18
19 # 答案变量，用于记录最大活动区域
20 ans = 0
21 # 用于检查的二维矩阵
22 # 0表示没检查过，1表示检查过了
23 check_list = [[0] * m for _ in range(n)]
24
25 # 构建DFS递归函数
26 def dfs(check_list, x, y):
27     # 声明变量area全局变量，表示当前DFS过程中的活动区域
28     global area
29     # 将点(x, y)标记为已检查过
30     check_list[x][y] = 1
31     # 更新当前DFS最大活动区域面积
32     area += 1
33     for dx, dy in DIRECTIONS:
34         nxt_x, nxt_y = x + dx, y + dy
35         # 若下一个点继续进行dfs，应该满足以下三个条件：
36         # 1.没有越界
37         # 2.grid[x][y]和grid[nxt_x][nxt_y]的差值的绝对值小于等于1
38         # 3.尚未被检查过
39         if (0 <= nxt_x < n and 0 <= nxt_y < m and check_list[nxt_x][nxt_y] == 0
40             and
41                 abs(grid[x][y] - grid[nxt_x][nxt_y]) <= 1):
42             # 可以进行dfs
43             dfs(check_list, nxt_x, nxt_y)
44
45 # 最外层的大的双重循环，是用来找DFS的起始搜索位置的
46 for i in range(n):
47     for j in range(m):
48         # 找到一个尚未检查过的点(i,j)，那么可以进行DFS的搜索
49         if check_list[i][j] == 0:
50             # 初始化最大活动区域面积为0
51             area = 0
52             dfs(check_list, i, j)
53             # DFS搜索完成，更新最大活动区域面积
54             ans = max(ans, area)
55
56 print(ans)

```



```

1 import java.util.*;
2
3 public class Main {
4     static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
5     static int n, m;
6     static int[][] grid;
7     static int[][] checkList;
8
9     public static void main(String[] args) {
10         Scanner scanner = new Scanner(System.in);
11
12         n = scanner.nextInt();
13         m = scanner.nextInt();
14         grid = new int[n][m];
15         checkList = new int[n][m];
16
17         for (int i = 0; i < n; i++) {
18             for (int j = 0; j < m; j++) {
19                 grid[i][j] = scanner.nextInt();
20             }
21         }
22
23         int ans = 0;
24
25         for (int i = 0; i < n; i++) {
26             for (int j = 0; j < m; j++) {
27                 if (checkList[i][j] == 0) {
28                     int[] area = {0};
29                     dfs(i, j, area);
30                     ans = Math.max(ans, area[0]);
31                 }
32             }
33         }
34
35         System.out.println(ans);
36     }
37
38     static void dfs(int x, int y, int[] area) {
39         checkList[x][y] = 1;
40         area[0]++;
41         for (int[] dir : DIRECTIONS) {
42             int nxt_x = x + dir[0];
43             int nxt_y = y + dir[1];
44
45             if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&

```

```

46         checkList[nxt_x][nxt_y] == 0 && Math.abs(grid[x][y] -
    grid[nxt_x][nxt_y]) <= 1) {
47             dfs(nxt_x, nxt_y, area);
48         }
49     }
50 }
51 }
52

```

## C++

```

1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  using namespace std;
5
6  int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
7  int n, m;
8  vector<vector<int>> grid;
9  vector<vector<int>> checkList;
10
11 void dfs(int x, int y, int &area) {
12     checkList[x][y] = 1;
13     area++;
14     for (auto dir : DIRECTIONS) {
15         int nxt_x = x + dir[0];
16         int nxt_y = y + dir[1];
17
18         if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
19             checkList[nxt_x][nxt_y] == 0 && abs(grid[x][y] - grid[nxt_x]
20             [nxt_y]) <= 1) {
21                 dfs(nxt_x, nxt_y, area);
22             }
23     }
24
25 int main() {
26     cin >> n >> m;
27     grid.resize(n, vector<int>(m));
28     checkList.resize(n, vector<int>(m, 0));
29
30     for (int i = 0; i < n; i++) {
31         for (int j = 0; j < m; j++) {
32             cin >> grid[i][j];
33         }
34     }
35 }

```

```
34     }
35
36     int ans = 0;
37
38     for (int i = 0; i < n; i++) {
39         for (int j = 0; j < m; j++) {
40             if (checkList[i][j] == 0) {
41                 int area = 0;
42                 dfs(i, j, area);
43                 ans = max(ans, area);
44             }
45         }
46     }
47
48     cout << ans << endl;
49
50     return 0;
51 }
52
```

## 时空复杂度

时间复杂度：  $O(MN)$  。

空间复杂度：  $O(MN)$  。