【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 × 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 ≤ M, N ≤ 150` ，`0 ≤ k ≤ 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

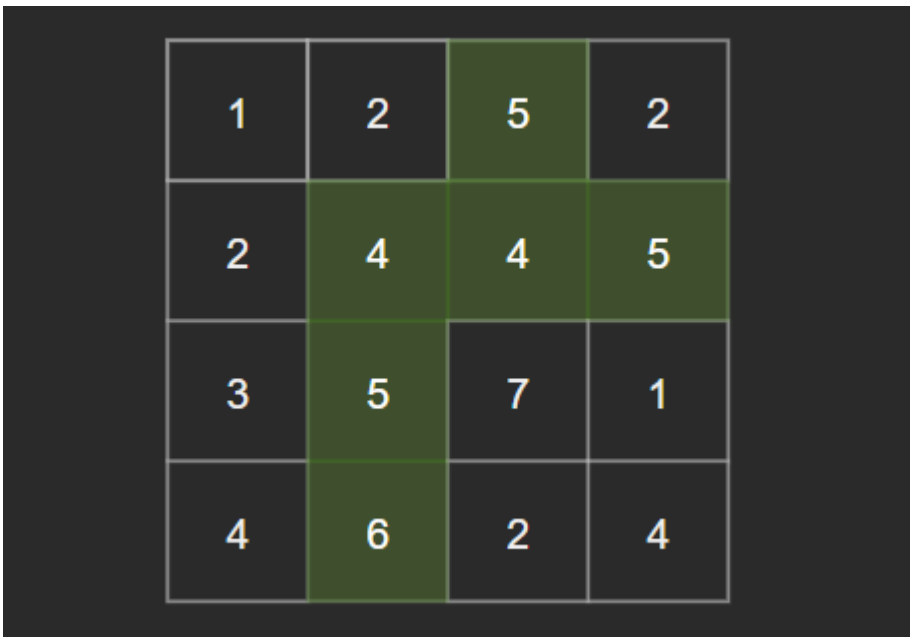
```
1 4 4
2 1 2 5 2
3 2 4 4 5
4 3 5 7 1
5 4 6 2 4
```

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

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

代码

解法一：BFS

Python

```python
# 题目：2024E-机器人的活动区域
# 分值：200
# 作者：闭着眼睛学数理化
# 算法：BFS
# 代码看不懂的地方，请直接在群上提问

from collections import deque

# 表示四个方向的数组
DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1)]

# 输入行数、列数
n, m = map(int, input().split())
# 输入地图
grid = list()
for i in range(n):
    row = list(map(int, input().split()))
    grid.append(row)


# 答案变量，用于记录最大活动区域
ans = 0
# 用于检查的二维矩阵
```

```python
# 0表示没检查过，1表示检查过了
check_list = [[0] * m for _ in range(n)]

# 最外层的双重循环，是用来找BFS的起始搜索位置的
for i in range(n):
    for j in range(m):
        # 找到一个尚未检查过的点(i,j)，那么可以进行BFS的搜索
        if check_list[i][j] == 0:
            # 初始化维护BFS过程的队列
            q = deque()
            q.append([i, j])
            # 将起始点(i,j)标记为已检查过
            check_list[i][j] = 1
            # 初始化最大活动区域面积为0
            area = 0
            # 进行BFS，当队列中还有元素时，持续地进行搜索
            while len(q) > 0:
                # 弹出队头元素，为当前点
                x, y = q.popleft()
                # 更新当前BFS最大活动区域面积
                area += 1
                # 考虑上下左右的四个近邻点
                for dx, dy in DIRECTIONS:
                    nxt_x, nxt_y = x+dx, y+dy
                    # 若下一个点要加入队列，应该满足以下三个条件：
                    # 1.没有越界
                    # 2.grid[x][y]和grid[nxt_x][nxt_y]的差值的绝对值小于等于1
                    # 3.尚未被检查过
                    if (0 <= nxt_x < n and 0 <= nxt_y < m and check_list[nxt_x][nxt_y] == 0 and
                            abs(grid[x][y] - grid[nxt_x][nxt_y]) <= 1):
                            q.append([nxt_x, nxt_y])        # 入队
                            check_list[nxt_x][nxt_y] = 1    # 标记为已检查过

            # BFS搜索完成，更新最大活动区域面积
            ans = max(ans, area)

print(ans)
```

Java

```java
import java.util.*;

public class Main {
    static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
```

```java

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        int m = scanner.nextInt();
        int[][] grid = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        int ans = 0;
        int[][] checkList = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (checkList[i][j] == 0) {
                    Queue<int[]> q = new LinkedList<>();
                    q.add(new int[]{i, j});
                    checkList[i][j] = 1;
                    int area = 0;

                    while (!q.isEmpty()) {
                        int[] point = q.poll();
                        int x = point[0];
                        int y = point[1];
                        area++;

                        for (int[] dir : DIRECTIONS) {
                            int nx = x + dir[0];
                            int ny = y + dir[1];

                            if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
                                    checkList[nx][ny] == 0 && Math.abs(grid[x]
[y] - grid[nx][ny]) <= 1) {
                                q.add(new int[]{nx, ny});
                                checkList[nx][ny] = 1;
                            }
                        }
                    }

                    ans = Math.max(ans, area);
                }
            }
```

```
        }

        System.out.println(ans);
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <cmath>
using namespace std;

int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

int main() {
    int n, m;
    cin >> n >> m;

    vector<vector<int>> grid(n, vector<int>(m));
    vector<vector<int>> checkList(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    int ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (checkList[i][j] == 0) {
                queue<pair<int, int>> q;
                q.push({i, j});
                checkList[i][j] = 1;
                int area = 0;

                while (!q.empty()) {
                    pair<int, int> point = q.front();
                    q.pop();
                    int x = point.first;
                    int y = point.second;
```

```
                    area++;

                    for (auto dir : DIRECTIONS) {
                        int nx = x + dir[0];
                        int ny = y + dir[1];

                        if (nx >= 0 && nx < n && ny >= 0 && ny < m &&
                            checkList[nx][ny] == 0 && abs(grid[x][y] - grid[nx]
[ny]) <= 1) {
                            q.push({nx, ny});
                            checkList[nx][ny] = 1;
                        }
                    }
                }

                ans = max(ans, area);
            }
        }
    }

    cout << ans << endl;

    return 0;
}

```

解法二：DFS

Python

```python
# 题目：2024E-机器人的活动区域
# 分值：200
# 作者：闭着眼睛学数理化
# 算法：DFS
# 代码看不懂的地方，请直接在群上提问

# 表示四个方向的数组
DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1)]

# 输入行数、列数
n, m = map(int, input().split())
# 输入地图
grid = list()
```

```python
for i in range(n):
    row = list(map(int, input().split()))
    grid.append(row)


# 答案变量，用于记录最大活动区域
ans = 0
# 用于检查的二维矩阵
# 0表示没检查过，1表示检查过了
check_list = [[0] * m for _ in range(n)]

# 构建DFS递归函数
def dfs(check_list, x, y):
    # 声明变量area全局变量，表示当前DFS过程中的活动区域
    global area
    # 将点(x, y)标记为已检查过
    check_list[x][y] = 1
    # 更新当前DFS最大活动区域面积
    area += 1
    for dx, dy in DIRECTIONS:
        nxt_x, nxt_y = x + dx, y + dy
        # 若下一个点继续进行dfs，应该满足以下三个条件：
        # 1.没有越界
        # 2.grid[x][y]和grid[nxt_x][nxt_y]的差值的绝对值小于等于1
        # 3.尚未被检查过
        if (0 <= nxt_x < n and 0 <= nxt_y < m and check_list[nxt_x][nxt_y] == 0
 and
            abs(grid[x][y] - grid[nxt_x][nxt_y]) <= 1):
                # 可以进行dfs
                dfs(check_list, nxt_x, nxt_y)


# 最外层的大的双重循环，是用来找DFS的起始搜索位置的
for i in range(n):
    for j in range(m):
        # 找到一个尚未检查过的点(i,j)，那么可以进行DFS的搜索
        if check_list[i][j] == 0:
            # 初始化最大活动区域面积为0
            area = 0
            dfs(check_list, i, j)
            # DFS搜索完成，更新最大活动区域面积
            ans = max(ans, area)

print(ans)
```

Java

```java
import java.util.*;

public class Main {
    static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
    static int n, m;
    static int[][] grid;
    static int[][] checkList;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        n = scanner.nextInt();
        m = scanner.nextInt();
        grid = new int[n][m];
        checkList = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

        int ans = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (checkList[i][j] == 0) {
                    int[] area = {0};
                    dfs(i, j, area);
                    ans = Math.max(ans, area[0]);
                }
            }
        }

        System.out.println(ans);
    }

    static void dfs(int x, int y, int[] area) {
        checkList[x][y] = 1;
        area[0]++;
        for (int[] dir : DIRECTIONS) {
            int nxt_x = x + dir[0];
            int nxt_y = y + dir[1];

            if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
```

```
                checkList[nxt_x][nxt_y] == 0 && Math.abs(grid[x][y] -
grid[nxt_x][nxt_y]) <= 1) {
                    dfs(nxt_x, nxt_y, area);
            }
        }
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};
int n, m;
vector<vector<int>> grid;
vector<vector<int>> checkList;

void dfs(int x, int y, int &area) {
    checkList[x][y] = 1;
    area++;
    for (auto dir : DIRECTIONS) {
        int nxt_x = x + dir[0];
        int nxt_y = y + dir[1];

        if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
            checkList[nxt_x][nxt_y] == 0 && abs(grid[x][y] - grid[nxt_x]
[nxt_y]) <= 1) {
                dfs(nxt_x, nxt_y, area);
        }
    }
}

int main() {
    cin >> n >> m;
    grid.resize(n, vector<int>(m));
    checkList.resize(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
```

```
    }

    int ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (checkList[i][j] == 0) {
                int area = 0;
                dfs(i, j, area);
                ans = max(ans, area);
            }
        }
    }

    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(MN)`。

空间复杂度：`O(MN)`。