【BFS】-周末爬山

题目描述与示例

题目描述：

周末小明准备去爬山锻炼，`0`代表平地，山的高度使用`1`到`9`来表示，小明每次爬山或下山高度只能相差`k`及`k`以内，每次只能上下左右一个方向上移动一格，小明从左上角`(0,0)`位置出发

输入描述

第一行输入`m n k`(空格分隔)

代表`m*n`的二维山地图，`k`为小明每次爬山或下山高度差的最大值，然后接下来输入山地图，一共`m`行`n`列，均以空格分隔。

取值范围：

`0 < m ≤ 500`

`0 < n ≤ 500`

`0 < k < 5`

输出描述

请问小明能爬到的最高峰多高，到该最高峰的**最短步数**，输出以空格分隔。

同高度的山峰输出较短步数。

如果没有可以爬的山峰，则高度和步数都返回`0`。

备注：所有用例输入均为正确格式，且在取值范围内，考生不需要考虑不合法的输入格式。

示例1

输入

```
1 5 4 1
2 0 1 2 0
3 1 0 0 0
4 1 0 1 2
5 1 3 1 0
6 0 0 0 9
```

输入

```
1 2 2
```

说明

根据山地图可知，能爬到的最高峰在 `(0,2)` 位置，高度为 `2` ，最短路径为 `(0,0)->(0,1)->(0,2)` ，最短步数为 `2` 。

示例2

输入

```
1 5 4 3
2 0 0 0 0
3 0 0 0 0
4 0 9 0 0
5 0 0 0 0
6 0 0 0 9
```

输入

```
1 0 0
```

说明

根据山地图可知，每次爬山距离 `3` ，无法爬到山峰上，步数为 `0` 。

解题思路

又是二维网格状无向图的搜索问题。

题目包含两个设问：找到最高峰和到达最高峰的最短路径。

找到最高峰的问题可以用DFS也可以用BFS解决，但是最短路径问题只能用BFS解决（回溯可以解决但是会超时）。

综上，可以用一次BFS就同时完成最高峰和最短路径的搜索。

代码

Python

```python
# 题目：【BFS】2023C-周末爬山
# 分值：200
# 作者：许老师-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
from math import inf


# 四个方向数组
DIERECTIONS = [(0,1), (1,0), (-1,0), (0,-1)]

# 输入长、宽、最大高度差
m, n, k = map(int, input().split())
# 初始化地图
grid = list()
for _ in range(m):
    grid.append(list(map(int, input().split())))

# 初始化BFS搜索层数，表示最短路径大小
level = 0
# 初始化全局的最高山峰高度，为起点的高度
highest_hill = grid[0][0]
# 初始化全局的到达最高山峰的最短路径
shortest_route = 0

# 初始化检查数组
check_list = [[0] * n for _ in range(m)]
# 起点设置为已检查过
check_list[0][0] = 1
# 初始化维护BFS的队列
```

```
q = deque()
# 起始位置只包含左上角的(0, 0)
q.append((0, 0))

# 进行BFS
while q:
    # 搜索层数+1
    level += 1
    qSize = len(q)
    for _ in range(qSize):
        x, y = q.popleft()
        # 四个近邻点
        for dx, dy in DIERECTIONS:
            nx, ny = x+dx, y+dy
            # 三个条件：
            # 1.未越界；2.未检查；3.当前点和近邻点之间高度差小于k
            if 0 <= nx < m and 0 <= ny < n and check_list[nx][ny] == 0 and abs(grid[nx][ny]-grid[x][y]) <= k:
                q.append((nx, ny))
                check_list[nx][ny] = 1
                # 如果该近邻点的高度大于全局最高山峰的高度
                # 则更新highest_hill和shortest_route
                if grid[nx][ny] > highest_hill:
                    highest_hill = grid[nx][ny]
                    shortest_route = level

print(highest_hill, shortest_route) if highest_hill > grid[0][0] else print(0, 0)
```

Java

```
import java.util.*;

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int m = scanner.nextInt();
        int n = scanner.nextInt();
        int k = scanner.nextInt();

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

```
                grid[i][j] = scanner.nextInt();
            }
        }

        int level = 0;
        int highestHill = grid[0][0];
        int shortestRoute = 0;

        int[][] checkList = new int[m][n];
        checkList[0][0] = 1;
        Queue<int[]> queue = new LinkedList<>();
        queue.add(new int[]{0, 0});

        while (!queue.isEmpty()) {
            level++;
            int qSize = queue.size();
            for (int i = 0; i < qSize; ++i) {
                int[] point = queue.poll();
                int x = point[0];
                int y = point[1];
                for (int[] dir : DIRECTIONS) {
                    int nx = x + dir[0];
                    int ny = y + dir[1];
                    if (0 <= nx && nx < m && 0 <= ny && ny < n && checkList[nx][ny] == 0 && Math.abs(grid[nx][ny] - grid[x][y]) <= k) {
                        queue.add(new int[]{nx, ny});
                        checkList[nx][ny] = 1;
                        if (grid[nx][ny] > highestHill) {
                            highestHill = grid[nx][ny];
                            shortestRoute = level;
                        }
                    }
                }
            }
        }
        if (highestHill > grid[0][0]){
            System.out.println(highestHill + " " + shortestRoute);
        }
        else{
            System.out.println("0 0");
        }
    }
}

```

C++

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

using namespace std;

vector<pair<int, int>> DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}};

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

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

    int level = 0;
    int highestHill = grid[0][0];
    int shortestRoute = 0;

    vector<vector<int>> checkList(m, vector<int>(n));
    checkList[0][0] = 1;
    queue<pair<int, int>> q;
    q.push({0, 0});

    while (!q.empty()) {
        level++;
        int qSize = q.size();
        for (int i = 0; i < qSize; ++i) {
            int x = q.front().first;
            int y = q.front().second;
            q.pop();
            for (auto& dir : DIRECTIONS) {
                int nx = x + dir.first;
                int ny = y + dir.second;
                if (0 <= nx && nx < m && 0 <= ny && ny < n && checkList[nx][ny] == 0 && abs(grid[nx][ny] - grid[x][y]) <= k) {
                    q.push({nx, ny});
                    checkList[nx][ny] = 1;
                    if (grid[nx][ny] > highestHill) {
                        highestHill = grid[nx][ny];
                        shortestRoute = level;
```

```
                    }
                }
            }
        }
    }
    if (highestHill > grid[0][0]){
            cout << highestHill << " " << shortestRoute << endl;
        }
    else{
        cout << "0 0" << endl;
    }
    return 0;
}
```

时空复杂度

时间复杂度：`O(NM)`。

空间复杂度：`O(NM)`。