周 末 爬 山

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。

补充说明：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 展开

示例1
输入：5 4 1
0 1 2 0
1 0 0 0
1 0 1 2
1 3 1 0
0 0 0 9
输出：2 2
说明：输出解读：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：5 4 3
0 0 0 0
0 0 0 0
0 9 0 0
0 0 0 0
0 0 0 9
输出：0 0
说明：输出解读：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。

```
import java.util.Scanner;
import java.util.LinkedList;
import java.util.Queue;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] strs = sc.nextLine().split(" ");
        int m = Integer.parseInt(strs[0]);
        int n = Integer.parseInt(strs[1]);
        int k = Integer.parseInt(strs[2]);

        int[][] map = new int[m][n];

        for (int i = 0; i < m; i++) {
            String[] peaks = sc.nextLine().split(" ");
            for (int j = 0; j < n; j++) {
                map[i][j] = Integer.parseInt(peaks[j]);
            }
        }

        climb(map, k);
```

```
}

static class    Point {
        int x;
        int y;
        int step;
        public Point(int x, int y, int step) {
                this.x = x;
                this.y = y;
                this.step = step;
        }
}

public static void climb(int[][] map, int k) {
        int m = map.length;
        int n = map[0].length;

        int[][] direction = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

        LinkedList<Point> queue = new LinkedList<>();
        queue.offer(new Point(0, 0, 0));

        boolean[][] visited = new boolean[m][n];
        visited[0][0] = true;

        int maxPeak = 0;
        int minSteps = 0;

        while (!queue.isEmpty()) {
                Point current = queue.poll();
                int x = current.x;
                int y = current.y;
                int steps = current.step;

                if (x == m - 1 && y == n - 1) {
                        break;
                }

                if (map[x][y] > maxPeak) {
                        maxPeak = map[x][y];
                        minSteps = steps;
                } else if (map[x][y] == maxPeak) {
                        minSteps = Math.min(steps, minSteps);
```

```
                }


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

                    if (nx >= 0 && nx < m && ny >= 0 && ny < n && !visited[nx][ny]) {
                        int diff = Math.abs(map[x][y] - map[nx][ny]);
                        if (diff <= k) {
                            queue.offer(new Point(nx, ny, steps + 1));
                            visited[nx][ny] = true;
                        }
                    }
                }
            }

            System.out.println(maxPeak + " " + minSteps);

        }
    }
```