

题目描述

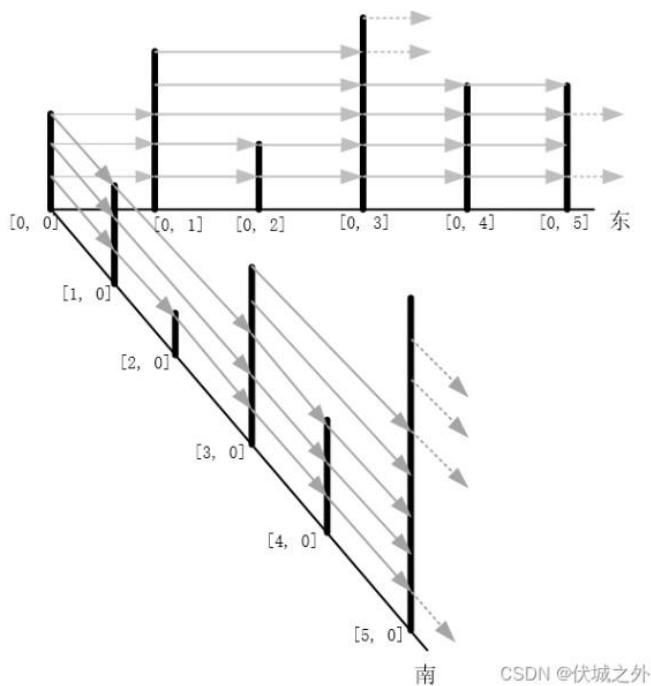
有一个二维的天线矩阵，每根天线可以向其他天线发射信号，也能接收其他天线的信号，为了简化起见，我们约定每根天线只能向东和向南发射信号，换言之，每根天线只能接收东向或南向的信号。

每根天线有自己的高度 $anth$ ，每根天线的高度存储在一个二维数组中，各个天线的位置用 $[r, c]$ 表示， $r$ 代表天线的行位置（从0开始编号）， $c$ 代表天线的列位置（从0开始编号）。

在某一方向（东向或南向），某根天线可以收到多根其他天线的信号（也可能收不到任何其他天线的信号），对任一天线 $X$ 和天线 $Y$ ，天线 $X$ 能接收到天线 $Y$ 的条件是：

- 1. 天线 $X$ 在天线 $Y$ 的东边或南边
- 2. 天线 $X$ 和天线 $Y$ 之间的其他天线的高度都低于天线 $X$ 和天线 $Y$ ，或天线 $X$ 和天线 $Y$ 之间无其他天线，即无遮挡。

如下图所示：



在天线矩阵的第0行上：

- 天线 $[0, 0]$ 接收不到任何其他天线的信号，
- 天线 $[0, 1]$ 可以接收到天线 $[0, 0]$ 的信号，
- 天线 $[0, 2]$ 可以接收到天线 $[0, 1]$ 的信号，
- 天线 $[0, 3]$ 可以接收到天线 $[0, 1]$ 和天线 $[0, 2]$ 的信号，
- 天线 $[0, 4]$ 可以接收到天线 $[0, 3]$ 的信号，
- 天线 $[0, 5]$ 可以接收到天线 $[0, 4]$ 的信号；

在天线的第0列上：

- 天线 $[0, 0]$ 接收不到任何其他天线的信号，
- 天线 $[1, 0]$ 可以接收到天线 $[0, 0]$ 的信号，
- 天线 $[2, 0]$ 可以接收到天线 $[1, 0]$ 的信号，
- 天线 $[3, 0]$ 可以接收到天线 $[1, 0]$ 和天线 $[2, 0]$ 的信号，
- 天线 $[4, 0]$ 可以接收到天线 $[3, 0]$ 的信号，
- 天线 $[5, 0]$ 可以接收到天线 $[3, 0]$ 和天线 $[4, 0]$ 的信号；

给一个 $m$ 行 $n$ 列的矩阵（二维数组），矩阵存储各根天线的高度，求出每根天线可以接收到多少根其他天线的信

在天线矩阵的第0行上：

- 天线[0, 0]接收不到任何其他天线的信号，
- 天线[0, 1]可以接收到天线[0, 0]的信号，
- 天线[0, 2]可以接收到天线[0, 1]的信号，
- 天线[0, 3]可以接收到天线[0, 1]和天线[0, 2]的信号，
- 天线[0, 4]可以接收到天线[0, 3]的信号，
- 天线[0, 5]可以接收到天线[0, 4]的信号；

在天线的第0列上：

- 天线[0, 0]接收不到任何其他天线的信号，
- 天线[1, 0]可以接收到天线[0, 0]的信号，
- 天线[2, 0]可以接收到天线[1, 0]的信号，
- 天线[3, 0]可以接收到天线[1, 0]和天线[2, 0]的信号，
- 天线[4, 0]可以接收到天线[3, 0]的信号，
- 天线[5, 0]可以接收到天线[3, 0]和天线[4, 0]的信号；

给一个m行n列的矩阵（二维数组），矩阵存储各根天线的高度，求出每根天线可以接收到多少根其他天线的信号，结果输出到m行n列的矩阵（二维矩阵）中。

输入描述

输入为1个m行n列的矩阵（二维矩阵）anth[m][n]，矩阵存储各根天线的高度，高度值anth[r][c]为大于0的整数。

第一行为输入矩阵的行数和列数，如：

m n

第二行为输入矩阵的元素值，按行输入，如：

anth[0][0] anth[0][1] ... anth[0][n-1] anth[1][0] anth[1][1] ... anth[1][n-1] ... anth[m-1][0] ... anth[m-1][n-1]

输出描述

输出一个m行n列的矩阵（二维数组）ret[m][n]，矩阵存储每根天线能收到多少根其他天线的信号，根数为ret[r][c]。

第一行为输出矩阵的行数和列数，如：

m n

第二行为输出矩阵的元素值，按行输出，如：

ret[0][0] ret[0][1] ... ret[0][n-1] ret[1][0] ret[1][1] ... ret[1][n-1] ... ret[m-1][0] ... ret[m-1][n-1]

备注

- $1 \leq m \leq 500$
- $1 \leq n \leq 500$
- $0 < anth[r][c] < 10^5$

用例

输入	1 6 2 4 1 5 3 3
输出	1 6 0 1 1 2 1 1
说明	输入为1行6列的天线矩阵的高度值 [2 4 1 5 3 3] 输出为1行6列的结果矩阵 [0 1 1 2 1 1]
输入	2 6 2 5 4 3 2 8 9 7 5 10 10 3
输出	2 6 0 1 1 1 1 4 1 2 2 4 2 2
说明	输入为2行6列的天线矩阵高度值 [2 5 4 3 2 8] [9 7 5 10 10 3] 输出为2行6列的结果矩阵 [0 1 1 1 1 4] [1 2 2 4 2 2]

题目解析

首先，本题我们需要从输入的一维数组中解析出二维天线矩阵，JS的实现略微麻烦，具体逻辑请看源码。

下面我们以用例1为例子，来解析本题，如下图是用例1的二维天线矩阵anth

--	--	--	--	--	--	--	--



伏城之外 已关注



1



2



0



专栏目录

已订阅

用例

输入	1 6 2 4 1 5 3 3
输出	1 6 0 1 1 2 1 1
说明	输入为1行6列的天线矩阵的高度值 [2 4 1 5 3 3] 输出为1行6列的结果矩阵 [0 1 1 2 1 1]

输入	2 6 2 5 4 3 2 8 9 7 5 10 10 3
输出	2 6 0 1 1 1 1 4 1 2 2 4 2 2
说明	输入为2行6列的天线矩阵高度值 [2 5 4 3 2 8] [9 7 5 10 10 3] 输出为2行6列的结果矩阵 [0 1 1 1 1 4] [1 2 2 4 2 2]

题目解析

首先，本题我们需要从输入的一维数组中解析出二维天线矩阵，JS的实现略微麻烦，具体逻辑请看源码。  
下面我们以用例1为例子，来解析本题，如下图是用例1的二维天线矩阵anth

	2	4	1	5	3	3

CSDN @伏城之外

我们从anth[0][0]开始发射，本题说了，天线信号发射只能向东或者向南发射，即如下图所示

	2	4	1	5	3	3

CSDN @伏城之外

由于用例1只有一层，因此只需要考虑向东发射信号。  
而东边的天线是否能收到信号，也有前提条件，即“发射天线”与“接收天线”之间的天线的高度都低于“发射天线”与“接收天线”，或者“发射天线”与“接收天线”之间没有其他天线  
因此，上面用例1中anth[0][0]可以发射到anth[0][1]，因为它们之间没有其他天线

	2	4	1	5	3	3

CSDN @伏城之外

但是anth[0][0]不能发射到anth[0][2]，因为它们之间的天线大于等于了它们

	2	4	1	5	3	3

CSDN @伏城之外

其实这一步，不需要走到anth[0][2]，因为anth[0][1] >= anth[0][0]，因此anth[0][0]必然会被anth[0][1]遮挡，导致无法继续向东发射。  
因此，对于anth[0][0]作为发射点的所有情况已经讨论完了，它只有一个接收点，那就是anth[0][1]。

接下来继续讨论anth[0][1]作为发射点

	2	4	1	5	3	3

CSDN @伏城之外

首先，相邻的anth[0][2]肯定能接收到信号。  
并且由于anth[0][2]小于anth[0][1]，因此无法完全将anth[0][1]发射的信号遮挡，

	2	4	1	5	3	3

CSDN @伏城之外

其实这一步，不需要走到 $\text{anth}[0][2]$ ，因为 $\text{anth}[0][1] \geq \text{anth}[0][0]$ ，因此 $\text{anth}[0][0]$ 必然会被 $\text{anth}[0][1]$ 遮挡，导致无法继续向东发射。

因此，对于 $\text{anth}[0][0]$ 作为发射点的所有情况已经讨论完了，它只有一个接收点，那就是 $\text{anth}[0][1]$ 。

接下来继续讨论 $\text{anth}[0][1]$ 作为发射点

	2	4	1	5	3	3	
							CSDN @伏城之外

首先，相邻的 $\text{anth}[0][2]$ 肯定能接收到信号。

并且由于 $\text{anth}[0][2] < \text{anth}[0][1]$ ，因此无法完全将 $\text{anth}[0][1]$ 发射的信号遮挡，

	2	4	1	5	3	3	
							CSDN @伏城之外

因此 $\text{anth}[0][3]$ 可以接收到 $\text{anth}[0][1]$ 的信号？

注意：这里我打了一个问号，因为题目要求，如果“发射天线”和“接收天线”之间有其他天线，那么其他天线的高度必须低于“发射天线”和“接收天线”。

上面打问号的原因是：我们只判断了中间天线  $\text{anth}[0][2] < \text{anth}[0][1]$  发射天线，并没有判断中间天线  $\text{anth}[0][2]$  也小于  $\text{anth}[0][3]$  接收天线。

- 而，这里中间天线  $\text{anth}[0][2]$  确实是小于  $\text{anth}[0][3]$  接收天线的，因此 $\text{anth}[0][3]$ 可以接收到 $\text{anth}[0][1]$ 的信号。

如果，我们采用双重for，外层遍历发射天线，内层遍历接收天线，则还需一个for遍历求得发射天线和接收天线之间的：所有中间天线中最高的高度h，如果这个高度h大于等于发射天线，或者接收天线，则发射天线和接收天线之间无法进行通信。

这其实已经是三重for了，再加上每个天线都会接收来自东向和南向这两个方向的信号，因此需要进行两次三重for。

这里的优化，我们可以利用单调递减栈。

首先定义一个单调递减栈stack，然后开始遍历天线 $\text{anth}[i][j]$ （比如先处理东向，即按行从左到右遍历）：

1、如果stack为空，则直接将天线 $\text{anth}[i][j]$ 加入stack

2、如果stack不为空，则获取栈顶天线top

2.1、如果 $\text{anth}[i][j] > \text{top}$ ，则将stack栈顶的top弹出，然后 $\text{anth}[i][j]$ 对应的 $\text{ret}[i][j]++$ ，表示 $\text{anth}[i][j]$ 天线新增接收一个信号，而由于stack栈是递减栈，因此 $\text{anth}[i][j]$ 还可以继续接收新栈顶天线的信号

2.2、如果 $\text{anth}[i][j] == \text{top}$ ，则将stack栈顶的top弹出，然后 $\text{anth}[i][j]$ 新增接收一个信号， $\text{ret}[i][j]++$ 。（注意，由于stack是严格递减栈，因此如果栈顶元素和 $\text{anth}[i][j]$ 等高，则必然只有一个，且stack弹栈后的新栈顶必然大于 $\text{anth}[i][j]$ ，此时其实可以直接结束）

2.3、如果 $\text{anth}[i][j] < \text{top}$ ，则表示 $\text{anth}[i][j]$ 已经无法接收到top之前的信号了，因为已经被top完全阻挡了。因此 $\text{anth}[i][j]$ 只能栈顶天线的信号， $\text{ret}[i][j]++$ ，而无法继续接收前面天线的信号。

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 })
13 if (lines.length === 2) {
14   const [m, n] = lines[0].split(" ").map(Number);
15   const arr = lines[1].split(" ").map(Number);
16   console.log(getResult(arr, m, n));
17   lines.length = 0;
18 }
19 });
20
21 function getResult(arr, m, n) {
22   const anth = new Array(m).fill(0).map(() => new Array(n));
23
24   for (let i = 0; i < m * n; i++) {
25     const r = Math.floor(i / n);
26     const c = i % n;
27     anth[r][c] = arr[i];
28   }
29 }
```

伏城之外 已关注

1 2 0 专栏目录 已订阅

```

1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 });
13
14 if (lines.length === 2) {
15   const [m, n] = lines[0].split(" ").map(Number);
16   const arr = lines[1].split(" ").map(Number);
17   console.log(getResult(arr, m, n));
18   lines.length = 0;
19 }
20
21 function getResult(arr, m, n) {
22   const anth = new Array(m).fill(0).map(() => new Array(n));
23
24   for (let i = 0; i < m * n; i++) {
25     const r = Math.floor(i / n);
26     const c = i % n;
27     anth[r][c] = arr[i];
28   }
29
30   const ret = new Array(m).fill(0).map(() => new Array(n).fill(0));
31
32   // 先处理水平方向，即先进行每行的东向发射处理
33   for (let i = 0; i < m; i++) {
34     const stack = [];
35     for (let j = 0; j < n; j++) {
36       // 如果栈顶天线比anth[i][j]，则anth[i][j]必然能接收到栈顶天线的信号，并且还能继续接收栈顶前一个天线的信号（递减栈）
37       while (stack.length && anth[i][j] > stack.at(-1)) {
38         ret[i][j]++;
39         stack.pop();
40       }
41
42       // 走到此步，如果stack还有值，那么由于是递减栈，因此此时栈顶天线高度必然 >= anth[i][j]
43       if (stack.length) {
44         // 如果栈顶天线高度 == anth[i][j]，那么此时anth[i][j]可以接收栈顶天线的信号，比如5 3 2 3，最后一个3可以接收到前面
45         if (anth[i][j] == stack.at(-1)) {
46           ret[i][j]++;
47           stack.pop(); // 维护严格递减栈
48         }
49         // 此情况必然是：anth[i][j] < stack.at(-1)，那么此时anth[i][j]可以接收栈顶天线的信号，比如6 5 2 3，最后一个3可
50         else {
51           ret[i][j]++;
52         }
53       }
54
55       stack.push(anth[i][j]);
56     }
57   }
58
59   // 再处理垂直方向，即每列的南向发射处理，和上面同理
60   for (let j = 0; j < n; j++) {
61     const stack = [];
62     for (let i = 0; i < m; i++) {
63       // 如果栈顶天线比anth[i][j]，则anth[i][j]必然能接收到栈顶天线的信号，并且还能继续接收栈顶前一个天线的信号（递减栈）
64       while (stack.length && anth[i][j] > stack.at(-1)) {
65         ret[i][j]++;
66         stack.pop();
67       }
68
69       // 走到此步，如果stack还有值，那么由于是递减栈，因此此时栈顶天线高度必然 >= anth[i][j]
70       if (stack.length) {
71         // 如果栈顶天线高度 == anth[i][j]，那么此时anth[i][j]可以接收栈顶天线的信号，比如5 3 2 3，最后一个3可以接收到前面
72         if (anth[i][j] == stack.at(-1)) {
73           ret[i][j]++;
74           stack.pop(); // 维护严格递减栈
75         }
76         // 此情况必然是：anth[i][j] < stack.at(-1)，那么此时anth[i][j]可以接收栈顶天线的信号，比如6 5 2 3，最后一个3可
77         else {
78           ret[i][j]++;
79         }
80       }
81
82       stack.push(anth[i][j]);
83     }
84   }
85
86   return `${m} ${n}\n${ret.toString().split(",").join(" ")}`;
87 }

```

提取公共代码，优化代码结构

提取公共代码，优化代码结构

```
1 /* JavaScript Node ACM模式 控制台输入获取 */
2 const readline = require("readline");
3
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout,
7 });
8
9 const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const [m, n] = lines[0].split(" ").map(Number);
15     const arr = lines[1].split(" ").map(Number);
16     console.log(getResult(arr, m, n));
17     lines.length = 0;
18   }
19 });
20
21 function getResult(arr, m, n) {
22   const anth = new Array(m).fill(0).map(() => new Array(n));
23
24   for (let i = 0; i < m * n; i++) {
25     const r = Math.floor(i / n);
26     const c = i % n;
27     anth[r][c] = arr[i];
28   }
29
30   const ret = new Array(m).fill(0).map(() => new Array(n).fill(0));
31
32   // 先处理水平方向，即先进行每行的东向发射处理
33   for (let i = 0; i < m; i++) {
34     const stack = [];
35     for (let j = 0; j < n; j++) {
36       common(stack, anth, ret, i, j);
37     }
38   }
39
40   // 再处理垂直方向，即每列的南向发射处理，和上面同理
41   for (let j = 0; j < n; j++) {
42     const stack = [];
43     for (let i = 0; i < m; i++) {
44       common(stack, anth, ret, i, j);
45     }
46   }
47
48   return `${m} ${n}\n${ret.toString().split(",").join(" ")}`;
49 }
50
51 function common(stack, anth, ret, i, j) {
52   // 如果栈顶天线比anth[i][j]，则anth[i][j]必然能接收到栈顶天线的信号，并且还能继续接收栈顶前面一个天线的信号（递减栈，栈顶元素
53   while (stack.length && anth[i][j] > stack.at(-1)) {
54     ret[i][j]++;
55     stack.pop();
56   }
57
58   // 走到此处，如果stack还有值，那么由于是递减栈，因此此时栈顶天线高度必然 >= anth[i][j]
59   if (stack.length) {
60     // 如果栈顶天线高度 == anth[i][j]，那么此时anth[i][j]可以接收栈顶天线的信号，比如5 3 2 3，最后一个3可以接收到前面等高的
61     if (anth[i][j] == stack.at(-1)) {
62       ret[i][j]++;
63       stack.pop(); // 维护严格递减栈
64     }
65     // 此情况必然是：anth[i][j] < stack.at(-1)，那么此时anth[i][j]可以接收栈顶天线的信号，比如6 5 2 3，最后一个3可以接收
66     else {
67       ret[i][j]++;
68     }
69   }
70   stack.push(anth[i][j]);
71 }
```

Java算法源码

```
1 import java.util.LinkedList;
2 import java.util.Scanner;
3 import java.util.StringJoiner;
4
5 public class Main {
6   public static void main(String[] args) {
7     Scanner sc = new Scanner(System.in);
8
9     int m = sc.nextInt();
10    int n = sc.nextInt();
11
12    int[][] anth = new int[m][n];
13
14    for (int i = 0; i < m; i++) {
15      for (int j = 0; j < n; j++) {
16        anth[i][j] = sc.nextInt();
17      }
18    }
19  }
```

伏城之外 已关注

1 2 0 专栏目录 已订阅



```
1 import java.util.LinkedList;
2 import java.util.Scanner;
3 import java.util.StringJoiner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int m = sc.nextInt();
10        int n = sc.nextInt();
11
12        int[][] anth = new int[m][n];
13
14        for (int i = 0; i < m; i++) {
15            for (int j = 0; j < n; j++) {
16                anth[i][j] = sc.nextInt();
17            }
18        }
19
20        System.out.println(getResult(anth, m, n));
21    }
22
23    public static String getResult(int[][] anth, int m, int n) {
24        int[][] ret = new int[m][n];
25
26        // 先处理南向发射信号
27        for (int j = 0; j < n; j++) {
28            LinkedList<Integer> stack = new LinkedList<>();
29            for (int i = 0; i < m; i++) {
30                // 如果栈顶天线比anth[i][j], 则anth[i][j]必然能接收到栈顶天线的信号, 并且还能继续接收栈顶前面一个天线的信号 (递减栈)
31                while (stack.size() > 0 && anth[i][j] > stack.getLast()) {
32                    ret[i][j] += 1;
33                    stack.removeLast();
34                }
35
36                // 走到此步, 如果stack还有值, 那么由于是递减栈, 因此此时栈顶天线高度必然
37                if (stack.size() > 0) {
38                    // 如果栈顶天线高度 == anth[i][j], 那么此时anth[i][j]可以接收栈顶天线的信号,
39                    // 比如5 3 2 3, 最后一个3可以接收到前面等高3的信号, 但是无法继续接收前面5的信号, 因此这里anth[i][j]结束处理
40                    if (anth[i][j] == stack.getLast()) {
41                        ret[i][j] += 1;
42                        stack.removeLast(); // 维护严格递减栈
43                    }
44                    // 此情况必然是: anth[i][j] < stack.at(-1), 那么此时anth[i][j]可以接收栈顶天线的信号,
45                    // 比如6 5 2 3, 最后一个3可以接收到前面5的信号, 但是无法继续接收前面6的信号, 因此这里anth[i][j]结束处理
46                    else {
47                        ret[i][j] += 1;
48                    }
49                }
50
51                stack.add(anth[i][j]);
52            }
53        }
54
55        StringJoiner sj = new StringJoiner(" ");
56
57        // 再处理东向发射信号, 和上面同理
58        for (int i = 0; i < m; i++) {
59            LinkedList<Integer> stack = new LinkedList<>();
60            for (int j = 0; j < n; j++) {
61                // 如果栈顶天线比anth[i][j], 则anth[i][j]必然能接收到栈顶天线的信号, 并且还能继续接收栈顶前面一个天线的信号 (递减栈)
62                while (stack.size() > 0 && anth[i][j] > stack.getLast()) {
63                    ret[i][j] += 1;
64                    stack.removeLast();
65                }
66
67                // 走到此步, 如果stack还有值, 那么由于是递减栈, 因此此时栈顶天线高度必然
68                if (stack.size() > 0) {
69                    // 如果栈顶天线高度 == anth[i][j], 那么此时anth[i][j]可以接收栈顶天线的信号,
70                    // 比如5 3 2 3, 最后一个3可以接收到前面等高3的信号, 但是无法继续接收前面5的信号, 因此这里anth[i][j]结束处理
71                    if (anth[i][j] == stack.getLast()) {
72                        ret[i][j] += 1;
73                        stack.removeLast(); // 维护严格递减栈
74                    }
75                    // 此情况必然是: anth[i][j] < stack.at(-1), 那么此时anth[i][j]可以接收栈顶天线的信号,
76                    // 比如6 5 2 3, 最后一个3可以接收到前面5的信号, 但是无法继续接收前面6的信号, 因此这里anth[i][j]结束处理
77                    else {
78                        ret[i][j] += 1;
79                    }
80                }
81
82                stack.add(anth[i][j]);
83                sj.add(ret[i][j] + "");
84            }
85        }
86
87        return m + " " + n + "\n" + sj.toString();
88    }
89 }
```

提取公共代码，优化代码结构

```
1 import java.util.LinkedList;
2 import java.util.Scanner;
3 import java.util.StringJoiner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         int m = sc.nextInt();
10        int n = sc.nextInt();
11
12        int[][] anth = new int[m][n];
13
14        for (int i = 0; i < m; i++) {
15            for (int j = 0; j < n; j++) {
16                anth[i][j] = sc.nextInt();
17            }
18        }
19
20        System.out.println(getResult(anth, m, n));
21    }
22
23    public static String getResult(int[][] anth, int m, int n) {
24        int[][] ret = new int[m][n];
25
26        // 先处理南向发射信号
27        for (int j = 0; j < n; j++) {
28            LinkedList<Integer> stack = new LinkedList<>();
29            for (int i = 0; i < m; i++) {
30                common(stack, anth, ret, i, j);
31            }
32        }
33
34        StringJoiner sj = new StringJoiner(" ");
35
36        // 再处理东向发射信号，和上面同理
37        for (int i = 0; i < m; i++) {
38            LinkedList<Integer> stack = new LinkedList<>();
39            for (int j = 0; j < n; j++) {
40                common(stack, anth, ret, i, j);
41                sj.add(ret[i][j] + "");
42            }
43        }
44
45        return m + " " + n + "\n" + sj.toString();
46    }
47
48    public static void common(LinkedList<Integer> stack, int[][] anth, int[][] ret, int i, int j) {
49        // 如果栈顶天线高度 == anth[i][j]，则anth[i][j]必然能接收到栈顶天线的信号，并且还能继续接收栈顶前面一个天线的信号（递减栈，栈
50        while (stack.size() > 0 && anth[i][j] > stack.getLast()) {
51            ret[i][j] += 1;
52            stack.removeLast();
53        }
54
55        // 走到此步，如果stack还有值，那么由于是递减栈，因此此时栈顶天线高度必然
56        if (stack.size() > 0) {
57            // 如果栈顶天线高度 == anth[i][j]，那么此时anth[i][j]可以接收栈顶天线的信号，
58            // 比如5 3 2 3，最后一个3可以接收到前面等高的3的信号，但是无法继续接收前面5的信号，因此这里anth[i][j]结束处理
59            if (anth[i][j] == stack.getLast()) {
60                ret[i][j] += 1;
61                stack.removeLast(); // 维护严格递减栈
62            }
63            // 此情况必然是：anth[i][j] < stack.at(-1)，那么此时anth[i][j]可以接收栈顶天线的信号，
64            // 比如6 5 2 3，最后一个3可以接收到前面5的信号，但是无法继续接收前面6的信号，因此这里anth[i][j]结束处理
65            else {
66                ret[i][j] += 1;
67            }
68        }
69
70        stack.add(anth[i][j]);
71    }
72 }
```

Python算法源码

```
1 # 输入读取
2 m, n = map(int, input().split())
3 arr = list(map(int, input().split()))
4
5
6 # 算法源码
7 def getResult(m, n, arr):
8     anth = [[0 for j in range(n)] for i in range(m)]
9
10    for i in range(m * n):
11        r = int(i / n)
12        c = i % n
13        anth[r][c] = arr[i]
14
15    ret = [[0 for i in range(n)] for i in range(m)]
```

伏城之外 已关注

1 2 0 专栏目录 已订阅



```

1 # 输入获取
2 m, n = map(int, input().split())
3 arr = list(map(int, input().split()))
4
5
6 # 算法源码
7 def getResult(m, n, arr):
8     anth = [[0 for j in range(n)] for i in range(m)]
9
10    for i in range(m * n):
11        r = int(i / n)
12        c = i % n
13        anth[r][c] = arr[i]
14
15    ret = [[0 for j in range(n)] for i in range(m)]
16
17    # 先处理东向发射信号
18    for i in range(m):
19        stack = []
20        for j in range(n):
21            # 如果栈顶天线比anth[i][j], 则anth[i][j]必然能接收到栈顶天线的信号, 并且还能继续接收栈顶前面一个天线的信号 (递减栈, 栈顶
22            while len(stack) > 0 and anth[i][j] > stack[-1]:
23                ret[i][j] += 1
24                stack.pop()
25            # 走到此步, 如果stack还有值, 那么由于是递减栈, 因此此时栈顶天线高度必然
26            if len(stack) > 0:
27                # 如果栈顶天线高度 == anth[i][j], 那么此时anth[i][j]可以接收栈顶天线的信号, 比如5 3 2 3, 最后一个3可以接收
28                if anth[i][j] == stack[-1]:
29                    ret[i][j] += 1
30                    stack.pop() # 维护严格递减栈
31            # 此情况必然是: anth[i][j] < stack.at(-1), 那么此时anth[i][j]可以接收栈顶天线的信号, 比如6 5 2 3, 最后一个3可以接收
32            else:
33                ret[i][j] += 1
34                stack.append(anth[i][j])
35
36    # 再处理南向发射信号
37    for j in range(n):
38        stack = []
39        for i in range(m):
40            # 如果栈顶天线比anth[i][j], 则anth[i][j]必然能接收到栈顶天线的信号, 并且还能继续接收栈顶前面一个天线的信号 (递减栈, 栈顶
41            while len(stack) > 0 and anth[i][j] > stack[-1]:
42                ret[i][j] += 1
43                stack.pop()
44            # 走到此步, 如果stack还有值, 那么由于是递减栈, 因此此时栈顶天线高度必然
45            if len(stack) > 0:
46                # 如果栈顶天线高度 == anth[i][j], 那么此时anth[i][j]可以接收栈顶天线的信号, 比如5 3 2 3, 最后一个3可以接收
47                if anth[i][j] == stack[-1]:
48                    ret[i][j] += 1
49                    stack.pop() # 维护严格递减栈
50            # 此情况必然是: anth[i][j] < stack.at(-1), 那么此时anth[i][j]可以接收栈顶天线的信号, 比如6 5 2 3, 最后一个3可以接收
51            else:
52                ret[i][j] += 1
53                stack.append(anth[i][j])
54
55    res = " ".join([" ".join(map(str, i)) for i in ret])
56
57    print(f'{m} {n}\n{res}')
58
59 # 算法调用
60 getResult(m, n, arr)

```

提取公共代码, 优化代码结构

```

1 # 输入获取
2 m, n = map(int, input().split())
3 arr = list(map(int, input().split()))
4
5
6 def common(stack, anth, ret, i, j):
7     # 如果栈顶天线比anth[i][j], 则anth[i][j]必然能接收到栈顶天线的信号, 并且还能继续接收栈顶前面一个天线的信号 (递减栈, 栈顶
8     while len(stack) > 0 and anth[i][j] > stack[-1]:
9         ret[i][j] += 1
10        stack.pop()
11    # 走到此步, 如果stack还有值, 那么由于是递减栈, 因此此时栈顶天线高度必然
12    if len(stack) > 0:
13        # 如果栈顶天线高度 == anth[i][j], 那么此时anth[i][j]可以接收栈顶天线的信号, 比如5 3 2 3, 最后一个3可以接收到前面
14        if anth[i][j] == stack[-1]:
15            ret[i][j] += 1
16            stack.pop() # 维护严格递减栈
17    # 此情况必然是: anth[i][j] < stack.at(-1), 那么此时anth[i][j]可以接收栈顶天线的信号, 比如6 5 2 3, 最后一个3可以接收
18    else:
19        ret[i][j] += 1
20        stack.append(anth[i][j])
21
22 # 算法源码
23 def getResult(m, n, arr):
24     anth = [[0 for j in range(n)] for i in range(m)]
25
26

```

提取公共代码，优化代码结构

```
1 # 输入获取
2 m, n = map(int, input().split())
3 arr = list(map(int, input().split()))
4
5
6 def common(stack, anth, ret, i, j):
7     # 如果栈顶天线比anth[i][j], 则anth[i][j]必然能接收到栈顶天线的信号, 并且还能继续接收栈顶前面一个天线的信号 (递减栈, 栈顶
8     while len(stack) > 0 and anth[i][j] > stack[-1]:
9         ret[i][j] += 1
10        stack.pop()
11    # 走到此步, 如果stack还有值, 那么由于是递减栈, 因此此时栈顶天线高度必然
12    if len(stack) > 0:
13        # 如果栈顶天线高度 == anth[i][j], 那么此时anth[i][j]可以接收栈顶天线的信号, 比如5 3 2 3, 最后一个3可以接收到前面
14        if anth[i][j] == stack[-1]:
15            ret[i][j] += 1
16            stack.pop() # 维护严格递减栈
17        # 此情况必然是: anth[i][j] < stack.at(-1), 那么此时anth[i][j]可以接收栈顶天线的信号, 比如6 5 2 3, 最后一个3可以
18        else:
19            ret[i][j] += 1
20    stack.append(anth[i][j])
21
22
23 # 算法源码
24 def getResult(m, n, arr):
25     anth = [[0 for j in range(n)] for i in range(m)]
26
27     for i in range(m * n):
28         r = int(i / n)
29         c = i % n
30         anth[r][c] = arr[i]
31
32     ret = [[0 for j in range(n)] for i in range(m)]
33
34     # 先处理东向发射信号
35     for i in range(m):
36         stack = []
37         for j in range(n):
38             common(stack, anth, ret, i, j)
39
40     # 再处理南向发射信号
41     for j in range(n):
42         stack = []
43         for i in range(m):
44             common(stack, anth, ret, i, j)
45
46     res = " ".join([" ".join(map(str, i)) for i in ret])
47
48     print(f'{m} {n}\n{res}')
49
50
51 # 算法调用
52 getResult(m, n, arr)
```