## 计算网络信号

知识点广搜Q数组

时间限制: 1s 空间限制: 256MB 限定语言: 不限

#### 题目描述:

网络信号经过传递会逐层衰减,且遇到阻隔物无法直接穿透,在此情况下需要计算某个位置的网络信号值。注意:网络信号可以绕过阻隔物array[m][n]的二维数组代表网格地图,array[i][j]=0代表i行j列是空旷位置;array[i][j]=x(x为正整数)代表i行j列是信号源,信号强度是x;array[i][j]=-1代表i行j列是阻隔物。信号源只有1个,阻隔物可能有0个或多个

网络信号衰减是上下左右相邻的网格衰减1

现要求输出对应位置的网络信号值

### 输入描述:

输入为三行,

第一行为m n, 代表输入是一个m\*n的数组

第二行是一串m\*n个用空格分隔的整数。每连续n个数代表一行,再往后n个代表下一行,以此 类推。对应的值代表对应的网格是空旷位置,还是信号源,还是阻隔物

第三行是i j,代表需要计算array[i][j]的网络信号值,注意:此处i和j均从0开始,即第一行i为0

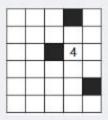
#### 例如:

65

000-10000000-14000000000-100000

1 4

代表如下地图



需要输出第2行第1列的网络信号值,如下图,值为2

		1		1
	1	2	3	2
	0		4	3
Г	1	2	3	2
		1	2	
-			1	

#### 输出描述:

输出对应位置的网络信号值,如果网络信号未覆盖到,也输出0。

一个网格如果可以途经不同的传播衰减路径传达,取较大的值作为其信号值。

#### 补充说明:

- 1、m不一定等于n, m<100, n<100, 网络信号值小于1000
- 2、信号源只有1个,阻隔物可能有0个或多个
- 3、输入的m, n与第二行的数组是合法的, 无需处理数量对不上的异常情况
- 4、要求输出信号值的位置,不会是阻隔物

```
示例1
输入:
65
000-100000000-14000000000-100000
21
输出:
0
示例2
输入:
65
000-100000000-14000000000-100000
14
输出:
```

#### 解题思路:

因为信号源是向着目标网格发射的,所以以目标网格的横纵坐标作为边界。 如果到达了目标网格,或者网格是阻隔物,或者网格信号值为0,则return, 因为阻隔物和信号值为0无法向外扩散信号。 同时需要注意的是,有的网格信号会受到边邻两个网格的信号影响,我们需要取其中最大值作为 本格信号值。

# public class Main{

```
public static int i;
public static int j;
public static int[][] signals; //网格地图二维数组
public static int signalX; //信号源横坐标
public static int signalY; //信号源纵坐标
public static void main(String[] args) {

Scanner sc = new Scanner(System.in);
int m = sc.nextInt();
int n = sc.nextInt();
```

```
signals = new int[m][n];
    for( int i=0; i<m; i++){
         for( int j=0; j<n; j++){
             signals[i][j] = sc.nextInt();
             if(signals[i][j]!=0&& signals[i][j]!=-1){ //信号
                  signalX = i;
                  signalY = j;
             }
         }
    }
    i = sc.nextInt();
    j = sc.nextInt();
    handle(signalX, signalY, signals[signalX][signalY]);
    System.out.println(signals[i][j]);
}
                      网格地图横坐标
 * @param x
                      网格地图纵坐标
 * @param y
 * @param signal
                    信号值
public static void handle(int x, int y, int signal){
    if(x == i && y == j){ //达到所求位置
         return;
    }
    if(signal == 0 || signal == -1){ //无信号进行传播
         return;
    }
    if(x < i){
         if(signals[x+1][y]!=-1){ //下一个位置如果是阻隔则无需进行操作
             signals[x+1][y] = Math.max(signals[x+1][y], signal - 1);
         handle( x+1, y, signals[x+1][y]);
    }
    if(y < j){
         if(signals[x][y+1]!=-1){ //下一个位置如果是阻隔则无需进行操作
```

```
signals[x][y+1] = Math.max(signals[x][y+1], signal - 1);
              }
              handle( x, y+1, signals[x][y+1]);
         }
         if(x > i){
              if(signals[x-1][y]!=-1){ //下一个位置如果是阻隔则无需进行操作
                  signals[x-1][y] = Math.max(signals[x-1][y], signal - 1);
              }
              handle( x-1, y, signals[x-1][y]);
         }
         if(y > j){
              if(signals[x][y-1]!=-1){ //下一个位置如果是阻隔则无需进行操作
                  signals[x][y-1] = Math.max(signals[x][y-1], signal - 1);
              handle( x, y-1, signals[x][y-1]);
         }
    }
}
```