

期中复习之一——动态规划&&DFS

求细胞数量 P1451

题目描述

一矩形阵列由数字 0 到 9 组成，数字 1 到 9 代表细胞，细胞的定义为沿细胞数字上下左右若还是细胞数字则为同一细胞，求给定矩形阵列的细胞个数。

输入格式

第一行两个整数代表矩阵大小 n 和 m 。

接下来 n 行，每行一个长度为 m 的只含字符 0 到 9 的字符串，代表这个 $n \times m$ 的矩阵。

输出格式

一行一个整数代表细胞个数。

样例 #1

样例输入 #1

```
1 4 10
2 0234500067
3 1034560500
4 2045600671
5 0000000089
```

样例输出 #1

```
1 4
```

提示

数据规模与约定

对于 100% 的数据，保证 $1 \leq n, m \leq 100$ 。

0	2	3	0	5	0	0	0	6	7
1	0	3	4	5	6	0	5	0	0
2	0	4	5	6	0	0	6	7	1
0	0	0	0	0	0	0	0	8	9

本质：遍历数组中的每一个点，因有以下儿种情况

- ① A 是 0 (没有细菌)
跳过。
- ② A 不是 0，且没有被标记过。
以 A 为中心，去探索他四周 (上、下、左、右) 的细菌，找到其中一个不为 0 的数 B，标记一下，再以 B 为中心，去探索 B 四周的细菌。
这个操作遍历了与 A 连接的所有细菌。
一个细菌群。
完成这个操作后将 cnt++ 记录一个细菌群。
- ③ A 不是 0，但被标记过了。

100%

代码：

```

1  #include <stdio.h>
2
3  int a[105][105]={0};
4  int check[105][105]={0};
5  void dfs(int x,int y);
6  //这个函数的作用就是通过一个点出发，标记他和
7  //他周围所有的"细菌" 作为一个细菌坨
8  const int dx[]={0,0,1,-1};
9  const int dy[]={1,-1,0,0};
10 //这两个数组保存上下左右四个方向的数
11 int main()
12 {
13     int n,m;
14     int i,j;
15     int cnt=0;
16     scanf("%d%d",&n,&m); // 读取n行m列
17
18     for(i=1;i<=n;i++){
19         for(j=1;j<=m;j++){
20             scanf("%1d",&a[i][j]); // 大家多去复习复习scanf的用法，期中考试
21         } //很愿意考读入操作， %1d的意思是每次仅读入
22     } //一位数字保存到a[i][j]中。
23
24     for(i=1;i<=n;i++){
25         for(j=1;j<=m;j++){
26
27             if(!check[i][j] && a[i][j]){
28                 dfs(i,j);

```

```

29         cnt++;
30         //细菌坨的个数++
31     }
32
33     }
34 }
35
36 printf("%d",cnt);
37
38 return 0;
39
40 }
41
42 void dfs(int x,int y){
43     int i;
44     int nx,ny;
45     check[x][y] = 1;
46
47     for(i=0;i<4;i++){           //遍历上下左右四个方向的细菌
48         nx = x + dx[i];
49         ny = y + dy[i];
50
51         if(a[nx][ny] && !check[nx][ny]){           //进入细菌坨的条件：这里有细菌
52                                                     // 即a[nx][ny]!=0 并且他之前没有被标记
53                                                     // 即check[nx][ny] = 0
54             dfs(nx,ny);           // 继续以a[nx][ny]为基准标记他上下左右
55                                     过
56                                     的方块。
57         }
58     }
59     return;
60 }

```

注：DFS的本质是暴力遍历每一种情况，取选择符合要求的路径，实现的方式就是递归。大家好好理解一下递归，我之后还会找很多递归的题(这个还是比较难的)。

滑雪 P1434

题目描述

Michael 喜欢滑雪。这并不奇怪，因为滑雪的确很刺激。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael 想知道在一个区域中最长的滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子：

1	1	2	3	4	5
2	16	17	18	19	6
3	15	24	25	20	7
4	14	23	22	21	8
5	13	12	11	10	9

一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度会减小。在上面的例子中，一条可行的滑坡为 24 - 17 - 16 - 1（从 24 开始，在 1 结束）。当然 25 - 24 - 23 - ... - 3 - 2 - 1 更长。事实上，这是最长的一条。

输入格式

输入的第一行为表示区域的二维数组的行数 R 和列数 C 。下面是 R 行，每行有 C 个数，代表高度（两个数字之间用 1 个空格间隔）。

输出格式

输出区域中最长滑坡的长度。

样例 #1

样例输入 #1

1	5	5			
2	1	2	3	4	5
3	16	17	18	19	6
4	15	24	25	20	7
5	14	23	22	21	8
6	13	12	11	10	9

样例输出 #1

1	25
---	----

提示

对于 100% 的数据， $1 \leq R, C \leq 100$ 。

Handwritten notes on a digital notepad showing a 5x5 grid of numbers and a recursive solution for finding the longest snow path.

Grid:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

思路: 遍历数组中的每一个点, 从该点开始, 算出该点的最长滑雪路径, 并找出最长的滑雪路径

会遇到以下情况:

①

5 1(A) 3

4

A的四周都是比她大的, 此时, A点为起点, 最长滑雪路径为0

②

5 3(A) 1(B)

2(C)

A的路径 = $\max(A\text{的路径}, B\text{的路径}+1, C\text{的路径}+1)$

(B, C的值替比A点的情况)

状态转移方程

代码:

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int a[110][110];
6  int f[110][110];
7  int check[105][105];
8  int max(int a, int b);
9  //这个函数用来求a, b的最大值
10 int dfs(int x, int y);
11 //这个函数的作用是算出从(x, y)开始的最长滑雪路径
12 int R, C;
13 const int dx[] = {0, 0, 1, -1};
14 const int dy[] = {1, -1, 0, 0};
15
16 int main()
17 {
18     int i, j;
19     int ans = 0;
20
21
22     scanf("%d%d", &R, &C);
23     //初始化处理
24     for(i=0; i<=R+1; i++){ //将边界的高度设置为无限高这样就免去了判断是否超出地图限制
25         for(j=0; j<=C+1; j++){
26             a[i][j] = 1e9;
27         }
28     }

```

```

29 //读入每一个点的数据
30 for(i=1;i<=R;i++){
31     for(j=1;j<=C;j++){
32         scanf("%d",&a[i][j]);
33     }
34 }
35
36 for(i=1;i<=R;i++){
37     for(j=1;j<=C;j++){
38         ans = max(ans,dfs(i,j));
39     }
40 }
41 //每一个点都跑一遍dfs，答案取最大的就是题目要求了
42
43 printf("%d",ans);
44
45 return 0;
46 }
47 int max(int a,int b){
48     if(a > b)
49         return a;
50     return b;
51 }
52
53 int dfs(int x,int y){
54     int i=0;
55     int nx,ny;
56     if(f[x][y]) return f[x][y]; //一开始每个点的步数都应该为0，
57                                     //如果当前这个点已知其最大步数说明之前该点
58                                     //已经被计算过就不用再重复计算了
59     f[x][y]=1; //既然这是一个从未走过的点那么现在来到该点至少都会使其
步数为1
60     for(i=0;i<4;i++){ //四个可行的方向
61         nx = x + dx[i];
62         ny = y + dy[i];
63         if(a[nx][ny] < a[x][y]){
64             f[x][y] = max(f[x][y],1+dfs(nx,ny)); //代码核心（状态转移方程）
65         }
66     }
67     return f[x][y]; //返回值给上一个dfs调用
68 }

```