实验舱
青少年编程
走近科学 走进名校

挑战信息学奥林匹克

C++程序设计（303）
二维数组（3）

例1：图像模糊处理

描述
给定n行m列的图像各像素点的灰度值，要求用如下方法对其进行模糊化处理：
1. 四周最外侧的像素点灰度值不变；
2. 中间各像素点新灰度值为该像素点及其上下左右相邻四个像素点原灰度值的平均（舍入到最接近的整数）。
输入
第一行包含两个整数n和m，表示图像包含像素点的行数和列数。1 <= n <= 100，1 <= m <= 100。
接下来n行，每行m个整数，表示图像的每个像素点灰度。相邻两个整数之间用单个空格隔开，每个元素均在0~255之间。
输出
n行，每行m个整数，为模糊处理后的图像。相邻两个整数之间用单个空格隔开。

样例输入
4 5
100 0 100 0 50
50 100 200 0 0
50 50 100 100 200
100 100 50 50 100
样例输出
100 0 100 0 50
50 80 100 60 0
50 80 100 90 200
100 100 50 50 100

问题分析

a数组

100	0	100	0	50
50	100	200	0	0
50	50	100	100	200
100	100	50	50	100

b数组

100	0	100	0	50
50	80	100	60	0
50	80	100	90	200
100	100	50	50	100

求平均值：

```
s = (double)(a[i-1][j] + a[i][j+1] + a[i+1][j] + a[i][j-1] + a[i][j]) / 5;
```

求最接近的整数：

```
b[i][j] = floor(s + 0.5);
```

模糊计算

1. 定义2个数组a、b
2. a数组读取数据，并赋值b数组
3. 枚举a数组，计算模糊值存放到b数组
4. 输出b数组

```
for ( int i = 2; i <= n - 1; i++ )
{
    for ( int j = 2; j <= m - 1; j++ )
    {
        double s=
          (double)(a[i-1][j]+a[i][j+1]+a[i+1][j]+a[i][j-1]+a[i][j])/5;
        b[i][j] = floor(s + 0.5);
    }
}
```

例2：计算鞍点

描述
给定一个5*5的矩阵，每行只有一个最大值，每列只有一个最小值，寻找这个矩阵的鞍点。
鞍点指的是矩阵中的一个元素，它是所在行的最大值，并且是所在列的最小值。
例如：在下面的例子中（第4行第1列的元素就是鞍点，值为8 ）。
11 3 5 6 9
12 4 7 8 10
10 5 6 9 11
8 6 4 7 2
15 10 11 20 25
输入
输入包含一个5行5列的矩阵
输出
如果存在鞍点，输出鞍点所在的行、列及其值，如果不存在，输出"not found"

样例输入
11 3 5 6 9
12 4 7 8 10
10 5 6 9 11
8 6 4 7 2
15 10 11 20 25
样例输出
4 1 8

算法分析

- 枚举所有的行
  1. 查找第h行的最大值，记录最大值所在列j；
  2. 枚举j列的所有的数据，检查是否最小值。

```
int j = 1;
for ( int i = 2; i <= n; i++ )
{
    if ( a[h][i] > a[h][j] )  j = i;
}
```

11	3	5	6	9
12	4	7	8	10
10	5	6	9	11
8	6	4	7	2
15	10	11	20	25

算法分析

- 枚举所有的行
  1. 查找第h行的最大值，记录最大值所在列j；
  2. 枚举j列的所有的数据，检查是否最小值。

```
int bj = 1;
for ( int i = 1; i <= n; i++ )
    if ( a[i][j] < a[h][j] )
    {
        bj = 0;
        break;
    }
```

11	3	5	6	9
12	4	7	8	10
10	5	6	9	11
8	6	4	7	2
15	10	11	20	25

查找鞍点

```
for ( int h = 1; h <= n; h++ )
{

    int j = 1;
    for ( int i = 2; i <= n; i++ )
    {
        if ( a[h][i] > a[h][j] )  j = i;
    }

    int bj = 1;
    for ( int i = 1; i <= n; i++ )
        if ( a[i][j] < a[h][j] )
        {
            bj = 0;
            break;
        }

    if ( bj == 1 )
    {
        cout << h << " " << j << " " << a[h][j] << endl;
        return 0;
    }
}
cout << "not found" << endl;
```

查表法

问题分析：

11	3	5	6	9
12	4	7	8	10
10	5	6	9	11
8	6	4	7	2
15	10	11	20	25

11	12	11	8	25
第1行	第2行	第3行	第4行	第5行

8	3	4	6	2
第1列	第2列	第3列	第4列	第5列

H[i] == a[i][j]

L[j] == a[i][j]

建表

```
for( int i = 1; i <= 5; i++ )
{
    for( int j = 1; j <= 5; j++ )
    {
        cin >> a[i][j];
        if( a[i][j] > H[i]) H[i] = a[i][j];
        if( a[i][j] < L[j]) L[j] = a[i][j];
    }
}
```

查表

```
for( int i = 1; i <= 5; i++ )
    for( int j = 1; j <= 5; j++ )
        if((a[i][j] == H[i])&&(a[i][j] == L[j]))
        {
            cout << i << " " << j << " " << a[i][j];
            return 0;
        }
```

例3：扫雷游戏地雷数计算

描述

扫雷游戏是一款十分经典的单机小游戏。它的精髓在于，通过已翻开格子所提示的周围格地雷数，来判断未翻开格子里是否是地雷。

现在给出n行m列的雷区中的地雷分布，要求计算出每个非地雷格的周围格地雷数。

注：每个格子周围格有八个：上、下、左、右、左上、右上、左下、右下。

输入

第一行包含两个整数n和m，分别表示雷区的行数和列数。1 <= n <= 100, 1 <= m <= 100。

接下来n行，每行m个字符，‘*’表示相应格子中是地雷，‘？’表示相应格子中无地雷。字符之间无任何分隔符。

输出

n行，每行m个字符，描述整个雷区。若相应格中是地雷，则用‘*’表示，否则用相应的周围格地雷数表示。字符之间无任何分隔符。

样例输入

3 3

*??

???

?*?

样例输出

*10

221

1*1

问题分析

- “*”代表雷
- a数组存放雷区
- b数组计算

```
char A[102][102];
int B[102][102]={0}
```

*	?	?
?	?	?
?	*	?

*	1	0
2	2	1
1	*	1

算法一

1. 读取字符到a数组
2. 枚举a数组，如果是”*”，b数组相应位置周边8个方向+1
3. 输出时，枚举a数组，如果是”*”，输出”*”，如果是”?”，输出b数组相应位置的数字。

*	?	?
?	?	?
?	*	?

*	1	0
2	2	1
1	*	1

```
for(i=1;i<=n;i++)
    for(j=1;j<=m;j++){
        cin >> A[i][j];
        if (A[i][j] == '*'){
            B[i-1][j-1]++;
            B[i][j-1]++;
            B[i+1][j-1]++;
            B[i+1][j]++;
            B[i+1][j+1]++;
            B[i][j+1]++;
            B[i-1][j+1]++;
            B[i-1][j]++;
        }
    }
```

算法二

1. 读取字符到a数组；
2. 枚举a数组，如果是”*”，直接输出”*”；
3. 如果是”?”，检查周边8个方向是否有地雷并进行累加；
4. 输出累加的数字。

*	?	?
?	?	?
?	*	?

	i-1, j-1	i-1, j	i-1, j+1	
	i, j-1	i, j	i, j+1	
	i+1, j-1	i+1, j	i+1, j+1	

```
int X[8] = {-1,-1,-1,0,1,1,1,0};
int Y[8] = {-1,0,1,1,1,0,-1,-1};
```

计算地雷数

```
for ( int i = 1; i <= n; i++ )
{
    for ( int j = 1; j <= m; j++ )
    {
        if ( A[i][j] == '*' ) cout << '*';
        else
        {
            s = 0;
            for ( int k = 0; k < 8; k++ )
            {
                r = i + X[k];
                c = j + Y[k];
                if ( A[r][c] == '*' ) s++;
            }
            cout << s;
        }
    }
    cout << endl;
}
```

例4：细菌的繁殖与扩散

描述

在边长为9的正方形培养皿中，正中心位置有m个细菌。假设细菌的寿命仅一天，但每天可繁殖10个后代，而且这10个后代，有两个分布在原来的单元格中，其余的均匀分布在其四周相邻的八个单元格中。求经过n(1≤n≤4)天后，细菌在培养皿中的分布情况。

输入

输入为两个整数，第一个整数m表示中心位置细菌的个数（2 ≤ m ≤ 30），第二个整数n表示经过的天数（1 ≤ n ≤ 4）。

输出

输出九行九列整数矩阵，每行的整数之间用空格分隔。整个矩阵代表n天后细菌在培养皿上的分布情况。

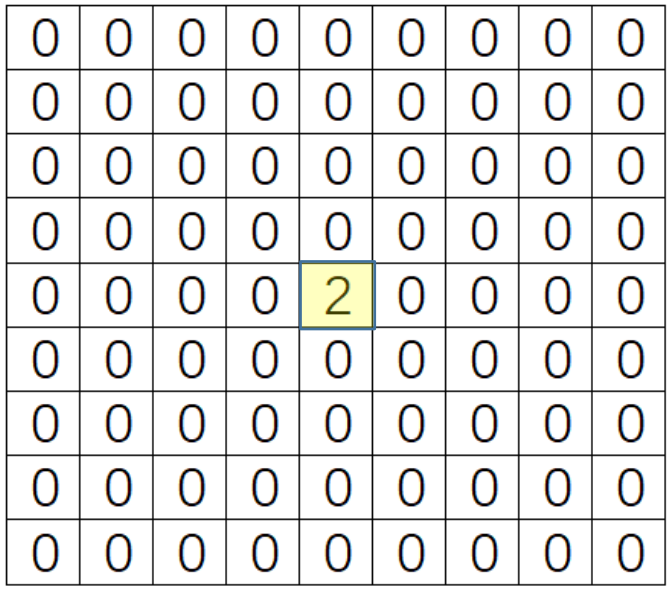
样例输入

2 1

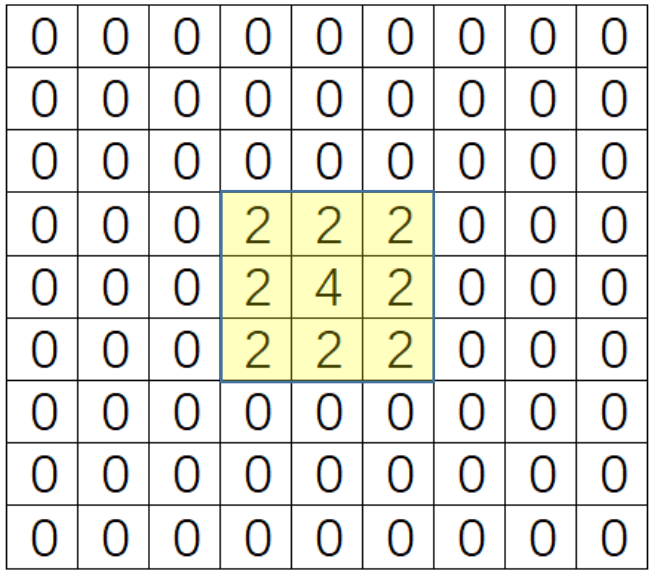
样例输出

0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 2 2 2 0 0 0
0 0 0 2 4 2 0 0 0
0 0 0 2 2 2 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0

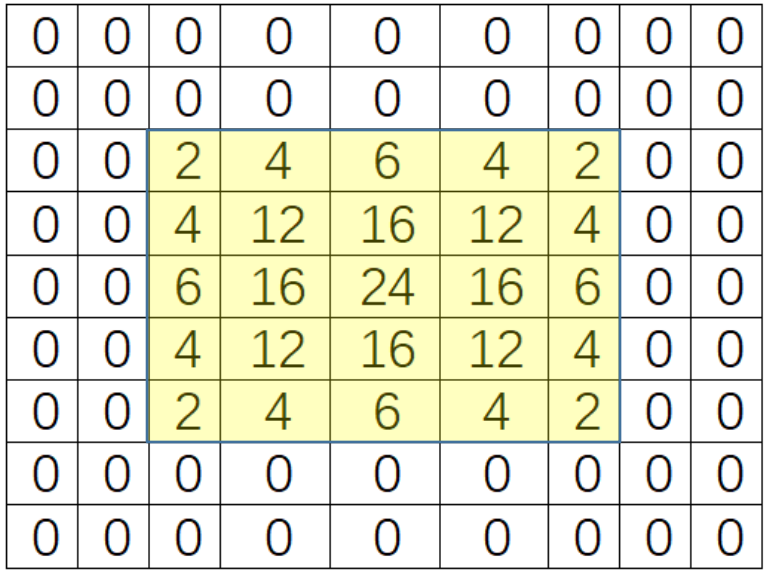
问题分析



第0天



第1天



第2天

问题分析

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	2	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

算法分析

1. 初始化数组：数组a和b全部清零，数组a中心赋初值；
2. 枚举数组a，查找大于0的a[i][j]，进行计算：
   ① b[i][j] += 2*a[i][j];
   ② 枚举a[i][j]四周8个方向，b[x][y] += a[i][j];
3. 计算结束后，将b数组的数据导入a数组，b数组清零，计算下一天的细菌数
4. 输出b数组

计算繁殖数

```
memset(b, 0, sizeof(b));
for ( int i = 1; i <= 9; i++ )
    for ( int j = 1; j <= 9; j++ )
    {
        if ( a[i][j] > 0 )
        {
            b[i][j] += 2 * a[i][j];
            for ( int k = 0; k < 8; k++ )
            {
                r = i + X[k];
                c = j + Y[k];
                b[r][c] += a[i][j];
            }
        }
    }
```

例5：肿瘤面积

描述
在一个正方形的灰度图片上，肿瘤是一块矩形的区域，肿瘤的边缘所在的像素点在图片中用0表示。其它肿瘤内和肿瘤外的点都用255表示。现在要求你编写一个程序，计算肿瘤内部的像素点的个数（不包括肿瘤边缘上的点）。已知肿瘤的边缘平行于图像的边缘。
输入
只有一个测试样例。第一行有一个整数n，表示正方形图像的边长。其后n行每行有n个整数，取值为0或255。整数之间用一个空格隔开。已知n不大于1000。
输出
输出一行，该行包含一个整数，为要求的肿瘤内的像素点的个数。

样例输入
5
255 255 255 255 255
255 0 0 0 255
255 0 255 0 255
255 0 0 0 255
255 255 255 255 255
样例输出
1

数学方法

- 查找左上角0的坐标（xl, yl）
- 查找右下角0的坐标（xr, yr）
- 计算面积：(xr-xl-1)*(yr-yl-1)
- xl是a[i][j]等于0时，i的最小值；
- yl是a[i][j]等于0时，j的最小值；
- xr是a[i][j]等于0时，i的最大值；
- yr是a[i][j]等于0时，j的最大值；

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	0	0	0	0	0	0	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	0	0	0	0	0	0	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

该算法不需要使用数组

算法提示

1. 二重循环for(i, j), 读取数据x;
2. 如果x等于0：
   a. xl < i, 替换xl;
   b. yl < j, 替换yl;
   c. xr > i, 替换xr;
   d. yr > j, 替换yr;
3. 计算(xr-xl-1)*(yr-yl-1)并输出

```
for ( int i = 0; i < n; i++ )
{
    for ( int j = 0; j <n; j++ )
    {
        cin >> x;
        if ( x == 0 )
        {
            if ( i <= xl && j <= yl )
            {
                xl = i; yl = j;
            }
            if ( i >= xr && j >= yr )
            {
                xr = i; yr = j;
            }
        }
    }
}
```

数据块方法

- 枚举数组找到第一个0的下标位置（x, y）
- 处理方法一
  1. 分别枚举（x, y）起点的行和列，求出0的个数
  2. 数学计算
- 处理方法二
  1. x+1，y+1，进入到肿瘤内部
  2. 枚举肿瘤内部计数

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	0	0	0	0	0	0	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	255	255	255	255	255	0	255	255
255	255	255	0	0	0	0	0	0	0	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

最好的草

.	#	#	.	.
.	#	#	.	.
.	.	.	#	#
.	.	.	#	.
.	.	#	#	.

稀疏矩阵

0	0	0	0	5
0	0	4	0	0
1	0	0	0	1

```
1 5 5
2 3 4
3 1 1
3 5 1
```