

第7章 |

Introduction to Java Programming, 8E

多维数组

学习目标

- 给出使用二维数组表示数据的例子 (7.1节)。
- 声明二维数组变量、创建数组，以及使用行下标和列下标访问二维数组中的数组元素 (7.2节)。
- 编程实现常用的二维数组的操作 (显示数组、对所有元素求和、找出最小元素和最大元素，以及随意打乱数组) (7.3节)。
- 给方法传递二维数组 (7.4节)。
- 使用二维数组编写多选题评分程序 (7.5节)。
- 使用二维数组解决距离最近的点对问题 (7.6节)。
- 使用二维数组检测一种九宫格的解决方案 (7.7节)。
- 使用多维数组 (7.8节)。

7.1 引言

第6章中介绍过一维数组如何存储线性的元素集合。可以使用二维数组存储矩阵或表格。例如，使用二维数组就可以存储下面这个描述城市之间距离的表格。

距离表 (以公里为单位)

	芝加哥	波士顿	纽约	亚特兰大	迈阿密	达拉斯	休斯敦
芝加哥	0	983	787	714	1375	967	1087
波士顿	983	0	214	1102	1763	1723	1842
纽约	787	214	0	888	1549	1548	1627
亚特兰大	714	1102	888	0	661	781	810
迈阿密	1375	1763	1549	661	0	1426	1187
达拉斯	967	1723	1548	781	1426	0	239
休斯敦	1087	1842	1627	810	1187	239	0

7.2 二维数组的基础知识

该如何声明一个二维数组变量呢？如何创建一个二维数组？如何访问二维数组中的元素？本节将解决这些问题。

7.2.1 声明二维数组变量并创建二维数组

下面是声明二维数组的语法：

数据类型[][] 数组名；

或者

数据类型 数组名[][]； // 允许这种方式，但并不推荐使用它

作为例子，下面演示如何声明int型的二维数组变量matrix：

int[][] matrix；

或者



`int matrix[][];` //允许这种方式,但并不推荐使用它

可以使用这个语法创建 5×5 的`int`型二维数组,并将它赋值给`matrix`:

`matrix = new int[5][5];`

二维数组中使用两个下标,一个表示行,另一个表示列。同一维数组一样,每个下标索引值都是`int`型的,从0开始,如图7-1a所示。

如图7-1b所示,要将7赋值给第2行第1列的特定元素,可以使用下面的语句:

`matrix[2][1] = 7;`

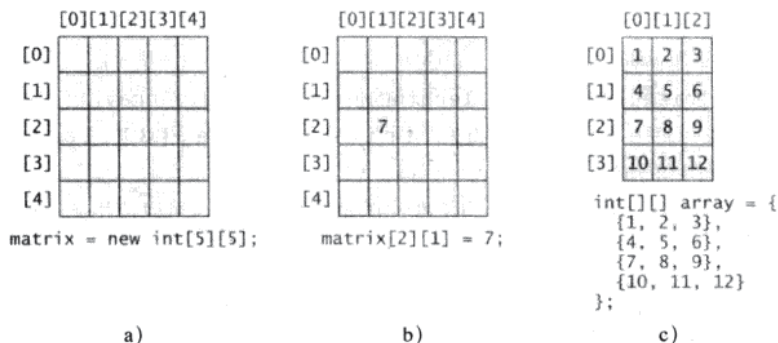
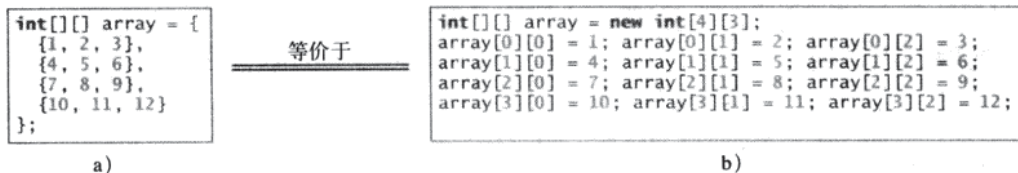


图7-1 二维数组的每个下标索引值都是从0开始的`int`值

警告 使用`matrix[2,1]`访问第2行第1列的元素是一种常见错误。在Java中,每个下标必须放在一对方括号中。

也可以使用数组初始化来声明、创建和初始化一个二维数组。例如:图a中的代码创建一个具有特定初值的数组,如图7-1c所示。它和图b中的代码是等价的。



7.2.2 获取二维数组的长度

二维数组实际上是一个数组,它的每个元素都是一个一维数组。数组`x`的长度是数组中元素的个数,可以用`x.length`获取该值。元素`x[0]`, `x[1]`, ..., `x[x.length-1]`也是数组。可以使用`x[0].length`, `x[1].length`, ..., `x[x.length-1].length`获取它们的长度。

例如:假设`x = new int[3][4]`,那么`x[0]`、`x[1]`和`x[2]`都是一维数组,每个数组都包含4个元素,如图7-2所示。`x.length`为3, `x[0].length`、`x[1].length`和`x[2].length`都是4。

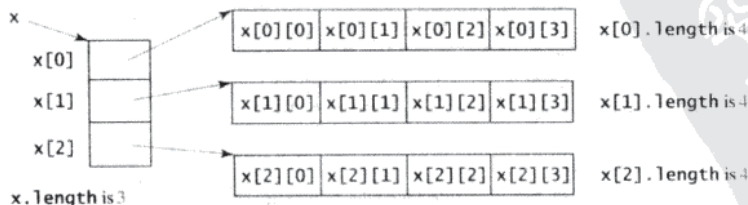


图7-2 二维数组是一个一维数组,它的每个元素是另一个一维数组

7.2.3 锯齿数组

二维数组中的每一行本身就是一个数组，因此，各行的长度就可以不同。这样的数组称为锯齿数组 (ragged array)。下面就是一个创建锯齿数组的例子：

```
int[][] triangleArray = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
```

从上图可以看到，`triangleArray[0].length`的值为5，`triangleArray[1].length`的值为4，`triangleArray[2].length`的值为3，`triangleArray[3].length`的值为2，`triangleArray[4].length`的值为1。

如果事先不知道锯齿数组的值，但知道它的长度，正如前面讲到的，可以使用如下所示的语法创建锯齿数组：

```
int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];
```

现在可以给数组赋值，例如：

```
triangleArray[0][3] = 50;
triangleArray[4][0] = 45;
```

注意 使用语法`new int[5][]`创建数组时，必须指定第一个下标。语法`new int[][]`是错误的。

7.3 处理二维数组

假设如下创建数组`matrix`：

```
int[][] matrix = new int[10][10];
```

下面是一些处理二维数组的例子：

- 1) (使用输入值初始化数组) 下面的循环使用用户输入值初始化数组：

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
    matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        matrix[row][column] = input.nextInt();
    }
}
```

- 2) (使用随机值初始化数组) 下面的循环使用0到99之间的随机值初始化数组：

```
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        matrix[row][column] = (int)(Math.random() * 100);
    }
}
```

- 3) (打印数组) 为打印一个二维数组，必须使用如下所示的循环打印数组中的每个元素：

```
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        System.out.print(matrix[row][column] + " ");
    }
}
```

```

    }
    System.out.println();
}

```

4) (求所有元素的和) 使用名为total的变量存储和。将total初始化为0。利用类似下面的循环, 把数组中的每一个元素都加到total上:

```

int total = 0;
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        total += matrix[row][column];
    }
}

```

5) (对数组按列求和) 对于每一列, 使用名为total的变量存储它的和。利用类似下面的循环, 将该列中的每个元素加到total上:

```

for (int column = 0; column < matrix[0].length; column++) {
    int total = 0;
    for (int row = 0; row < matrix.length; row++)
        total += matrix[row][column];
    System.out.println("Sum for column " + column + " is " +
        total);
}

```

6) (哪一行的和最大?) 使用变量maxRow和indexOfMaxRow分别跟踪和的最大值以及该行的索引值。计算每一行的和, 如果计算出的新行的和更大, 就更新maxRow和indexOfMaxRow。

```

int maxRow = 0;
int indexOfMaxRow = 0;

// Get sum of the first row in maxRow
for (int column = 0; column < matrix[0].length; column++) {
    maxRow += matrix[0][column];
}

for (int row = 1; row < matrix.length; row++) {
    int totalOfThisRow = 0;
    for (int column = 0; column < matrix[row].length; column++)
        totalOfThisRow += matrix[row][column];

    if (totalOfThisRow > maxRow) {
        maxRow = totalOfThisRow;
        indexOfMaxRow = row;
    }
}

```

```

System.out.println("Row " + indexOfMaxRow
    + " has the maximum sum of " + maxRow);

```

7) (随意打乱) 在6.2.6节中已经介绍了如何打乱一维数组的元素, 那么如何打乱二维数组中的所有元素呢? 为了实现这个功能, 对每个元素matrix[i][j], 随机产生下标i1和j1, 然后互换matrix[i][j]和matrix[i1][j1], 如下所示:

```

for (int i = 0; i < matrix.length; i++) {
    for (int j = 0; j < matrix[i].length; j++) {
        int i1 = (int)(Math.random() * matrix.length);
        int j1 = (int)(Math.random() * matrix[i].length);

        // Swap matrix[i][j] with matrix[i1][j1]
        int temp = matrix[i][j];
        matrix[i][j] = matrix[i1][j1];
        matrix[i1][j1] = temp;
    }
}

```

7.4 给方法传递二维数组

可以像传递一维数组一样，给方法传递二维数组。程序清单7-1给出一个返回矩阵中所有元素之和的方法。

程序清单7-1 PassTwoDimensionalArray.java

```

1 import java.util.Scanner;
2
3 public class PassTwoDimensionalArray {
4     public static void main(String[] args) {
5         // Create a Scanner
6         Scanner input = new Scanner(System.in);
7
8         // Enter array values
9         int[][] m = new int[3][4];
10        System.out.println("Enter " + m.length + " rows and "
11            + m[0].length + " columns: ");
12        for (int i = 0; i < m.length; i++)
13            for (int j = 0; j < m[i].length; j++)
14                m[i][j] = input.nextInt();
15
16        // Display result
17        System.out.println("\nSum of all elements is " + sum(m));
18    }
19
20    public static int sum(int[][] m) {
21        int total = 0;
22        for (int row = 0; row < m.length; row++) {
23            for (int column = 0; column < m[row].length; column++) {
24                total += m[row][column];
25            }
26        }
27
28        return total;
29    }
30 }

```

Enter 3 rows and 4 columns:

1 2 3 4

5 6 7 8

9 10 11 12

Sum of all elements is 78



方法sum（第20行）有一个二维数组参数。可以使用m.length获取行数（第22行），而使用m[row].column得到特定行的列数（第23行）。

7.5 问题：多选题测验评分

这里的问题是编写一个程序，对多选题测验进行打分。假设这里有8个学生和10道题目，学生的答案存储在一个二维数组中。每一行记录一名学生对所有题目的答案，如下面数组所示：

学生给出的题目答案：

0 1 2 3 4 5 6 7 8 9

Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

正确答案存储在一个一维数组中：

题目的正确答案：
 0 1 2 3 4 5 6 7 8 9
 Key D B D C C D A E A D

程序给测验打分并显示结果。它将每个学生的答案与正确答案进行比较，统计正确答案的个数，并将其显示出来。程序清单7-2给出该程序。

程序清单7-2 GradeExam.java

```

1 public class GradeExam {
2     /** Main method */
3     public static void main(String[] args) {
4         // Students' answers to the questions
5         char[][] answers = {
6             {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
7             {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
8             {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
9             {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
10            {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
11            {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
12            {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
13            {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'};
14
15            // Key to the questions
16            char[] keys = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};
17
18            // Grade all answers
19            for (int i = 0; i < answers.length; i++) {
20                // Grade one student
21                int correctCount = 0;
22                for (int j = 0; j < answers[i].length; j++) {
23                    if (answers[i][j] == keys[j])
24                        correctCount++;
25                }
26
27                System.out.println("Student " + i + "'s correct count is " +
28                    correctCount);
29            }
30        }
31    }

```

```

Student 0's correct count is 7
Student 1's correct count is 6
Student 2's correct count is 5
Student 3's correct count is 4
Student 4's correct count is 8
Student 5's correct count is 7
Student 6's correct count is 7
Student 7's correct count is 7

```



第5~13行的语句声明、创建和初始化一个二维字符数组，并将它的引用赋值给char[][]型变量answers。

第16行的语句声明、创建和初始化一个char值构成的数组，并将其引用赋值给char[]型变量keys。

数组answers的每一行存储一个学生的答案，将它与数组keys中的正确答案比较之后进行打分。给一个学生打完分数后就立刻将结果显示出来。

7.6 问题：找出距离最近的点对

GPS导航系统正在变得越来越流行。这个系统使用图形和几何算法计算距离，并且绘制出一个线路。本节介绍一个找出最接近点对的几何问题。

假设有一个集合的点，找出最接近的点对问题就是找到两个点，它们到彼此的距离最近。例如：在图7-3中，点(1,1)和(2,0.5)是彼此之间距离最近的一对点。解决这个问题的方法有好几种。一种直观的方法就是计算所有点对之间的距离，并且找出最短的距离，它的实现如程序清单7-3所示。

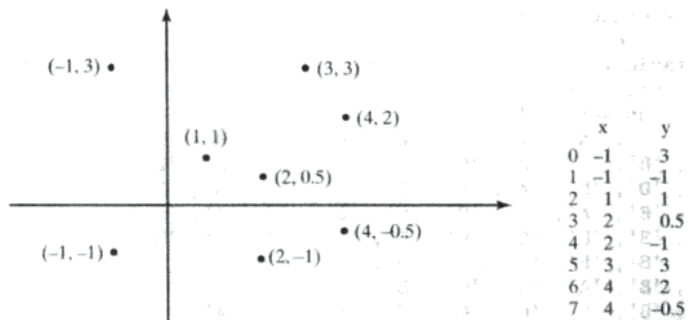


图7-3 使用二维数组表示点

程序清单7-3 FindNearestPoints.java

```

1 import java.util.Scanner;
2
3 public class FindNearestPoints {
4     public static void main(String[] args) {
5         Scanner input = new Scanner(System.in);
6         System.out.print("Enter the number of points: ");
7         int numberOfPoints = input.nextInt();
8
9         // Create an array to store points
10        double[][] points = new double[numberOfPoints][2];
11        System.out.print("Enter " + numberOfPoints + " points: ");
12        for (int i = 0; i < points.length; i++) {
13            points[i][0] = input.nextDouble();
14            points[i][1] = input.nextDouble();
15        }
16
17        // p1 and p2 are the indices in the points array
18        int p1 = 0, p2 = 1; // Initial two points
19        double shortestDistance = distance(points[p1][0], points[p1][1],
20            points[p2][0], points[p2][1]); // Initialize shortestDistance
21
22        // Compute distance for every two points
23        for (int i = 0; i < points.length; i++) {
24            for (int j = i + 1; j < points.length; j++) {
25                double distance = distance(points[i][0], points[i][1],
26                    points[j][0], points[j][1]); // Find distance
27
28                if (shortestDistance > distance) {
29                    p1 = i; // Update p1
30                    p2 = j; // Update p2
31                    shortestDistance = distance; // Update shortestDistance
32                }
33            }
34        }
35
36        // Display result
37        System.out.println("The closest two points are " +

```

```

38     "(" + points[p1][0] + ", " + points[p1][1] + ") and (" +
39     points[p2][0] + ", " + points[p2][1] + ")");
40 }
41
42 /** Compute the distance between two points (x1, y1) and (x2, y2)*/
43 public static double distance(
44     double x1, double y1, double x2, double y2) {
45     return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
46 }
47 }

```

Enter the number of points: 8
 Enter 8 points: -1 3 -1 -1 1 1 2 0.5 2 -1 3 3 4 2 4 -0.5
 The closest two points are (1, 1) and (2, 0.5)



程序提示用户输入点的个数（第6~7行）。从控制台读取多个点，并将它们存储在一个名为`points`的二维数组中（第12~15行）。程序使用变量`shortestDistance`（第19行）来存储两个距离最近的点，而这两个点在`points`数组中的下标都存储在`p1`和`p2`中（第18行）。

对每一个索引值为 i 的点，程序会对所有的 $j > i$ 计算`points[i]`和`points[j]`之间的距离（第23~34行）。只要找到比当前最短距离更短的距离，就更新变量`shortestDistance`以及`p1`和`p2`（第28~32行）。两个点 (x_1, y_1) 和 (x_2, y_2) 之间的距离可以使用公式 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ 计算（第43~46行）。程序假设平面上至少有两个点。可以简单地修改程序，处理平面没有点或只有一个点的情况。

注意 也可能会有不止一对具有相同最小距离的点对。程序找到这样的一对点。可以在编程练习题7.8中修改这个程序，找出所有距离最短的点对。

提示 从键盘输入所有的点是很繁琐的。可以将输入存储在一个名为`FindNearestPoints.txt`的文件中，并使用下面的命令编译和运行这个程序：

```
java FindNearestPoints < FindNearestPoints.txt
```

7.7 问题：九宫格

本书教授如何为各种不同难度的问题来编程，使用简单、短小和刺激的例子来介绍程序设计以及解决问题的技术，并且使用有趣的和有挑战性的例子来激发学生的兴趣。本节介绍一个每天都会出现在报纸上的很有趣的问题。这是一个数字放置的难题，通常称为九宫格（Sudoku）。它是一个非常有挑战性的问题。为了使之能被编程新手接受，本节给出九宫格问题的简化版本的一个解决方案，它可以验证该解决方案是否正确。解决九宫格问题的完整方案放在补充材料VII.A中。

九宫格是一个 9×9 的网格，它被分为更小的 3×3 的盒子（也称为区域或者块），如图7-4a所示。将从1到9的数字植入一些称为固定方格（fixed cell）的格子里。该程序的目标是将从1到9的数字植入那些称为自由方格（free cell）的格子，以便达到能够使得每行每列以及每个 3×3 的盒子都包含从1到9的数字，如图7-4b所示。

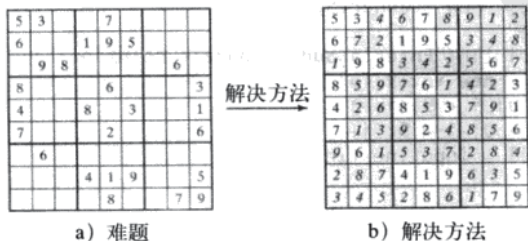


图7-4 图a中的难题在图b中解决

为了方便起见,使用值0表示自由方格,如图7-5a所示。很自然就会使用二维数组表示网格,如图7-5b所示。

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

a)

```
int[][] grid =
{{5, 3, 0, 0, 7, 0, 0, 0, 0},
 {6, 0, 0, 1, 9, 5, 0, 0, 0},
 {0, 9, 8, 0, 0, 0, 0, 6, 0},
 {8, 0, 0, 0, 6, 0, 0, 0, 3},
 {4, 0, 0, 8, 0, 3, 0, 0, 1},
 {7, 0, 0, 0, 2, 0, 0, 0, 6},
 {0, 6, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 4, 1, 9, 0, 0, 5},
 {0, 0, 0, 0, 8, 0, 0, 7, 9}};
```

b)

图7-5 使用一个二维数组表示网格

为了找到该难题的解决方案,必须用1到9之间合适的数字替换网格中的每个0。对于图7-4b中的解决方案,网格应该如图7-6所示。

```
A solution grid is
{{5, 3, 4, 6, 7, 8, 9, 1, 2},
 {6, 7, 2, 1, 9, 5, 3, 4, 8},
 {1, 9, 8, 3, 4, 2, 5, 6, 7},
 {8, 5, 9, 7, 6, 1, 4, 2, 3},
 {4, 2, 6, 8, 5, 3, 7, 9, 1},
 {7, 1, 3, 9, 2, 4, 8, 5, 6},
 {9, 6, 1, 5, 3, 7, 2, 8, 4},
 {2, 8, 7, 4, 1, 9, 6, 3, 5},
 {3, 4, 5, 2, 8, 6, 1, 7, 9}};
```

图7-6 解决方案存储在网格grid中

九宫格问题的简化版本用来检测解决方案的有效性。程序清单7-4中的程序提示用户输入一个解决方案,然后报告它是否有效。

程序清单7-4 CheckSudokuSolution.java

```
1 import java.util.Scanner;
2
3 public class CheckSudokuSolution {
4     public static void main(String[] args) {
5         // Read a Sudoku solution
6         int[][] grid = readASolution();
7
8         System.out.println(isValid(grid) ? "Valid solution" :
9             "Invalid solution");
10    }
11
12    /** Read a Sudoku solution from the console */
13    public static int[][] readASolution() {
14        // Create a Scanner
15        Scanner input = new Scanner(System.in);
16
17        System.out.println("Enter a Sudoku puzzle solution:");
18        int[][] grid = new int[9][9];
19        for (int i = 0; i < 9; i++)
20            for (int j = 0; j < 9; j++)
21                grid[i][j] = input.nextInt();
22
23        return grid;
24    }
25
26    /** Check whether a solution is valid */
27    public static boolean isValid(int[][] grid) {
```

```

28 // Check whether each row has numbers 1 to 9
29 for (int i = 0; i < 9; i++)
30     if (!is1To9(grid[i])) // If grid[i] does not contain 1 to 9
31         return false;
32
33 // Check whether each column has numbers 1 to 9
34 for (int j = 0; j < 9; j++) {
35     // Obtain a column in the one-dimensional array
36     int[] column = new int[9];
37     for (int i = 0; i < 9; i++) {
38         column[i] = grid[i][j];
39     }
40
41     if (!is1To9(column)) // If column does not contain 1 to 9
42         return false;
43 }
44
45 // Check whether each 3-by-3 box has numbers 1 to 9
46 for (int i = 0; i < 3; i++) {
47     for (int j = 0; j < 3; j++) {
48         // The starting element in a small 3-by-3 box
49         int k = 0;
50         int[] list = new int[9]; // Get all numbers in the box to list
51         for (int row = i * 3; row < i * 3 + 3; row++)
52             for (int column = j * 3; column < j * 3 + 3; column++)
53                 list[k++] = grid[row][column];
54
55         if (!is1To9(list)) // If list does not contain 1 to 9
56             return false;
57     }
58 }
59
60 return true; // The fixed cells are valid
61 }
62
63 /** Check whether the one-dimensional array contains 1 to 9 */
64 public static boolean is1To9(int[] list) {
65     // Make a copy of the array
66     int[] temp = new int[list.length];
67     System.arraycopy(list, 0, temp, 0, list.length);
68
69     // Sort the array
70     java.util.Arrays.sort(temp);
71
72     // Check whether the list contains 1, 2, 3, ..., 9
73     for (int i = 0; i < 9; i++)
74         if (temp[i] != i + 1)
75             return false;
76
77     return true; // The list contains exactly 1 to 9
78 }
79 }

```

Enter a Sudoku puzzle solution:

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

Valid solution



PDG

程序调用`readASolution()`方法(第6行)来读取一个九宫格的解决方案,并且返回一个表示九宫格网格的二维数组。

`isValid(grid)`方法(第27~61行)检查每行是否都包含从1到9的数字(第29~31行)。`grid`是一个二维数组。`grid[i]`是第*i*行的一维数组。如果`grid[i]`行的确包含从1到9的数,那么调用`is1To9(grid[i])`方法就会返回`true`(第30行)。

为了检测`grid`中每一列是否都包含从1到9的数字,将一列放到一个一维数组中(第36~39行),然后调用`is1To9`方法来检测它是否包括从1到9的数字(第41行)。

为了检测`grid`中每个小的 3×3 的盒子中是否包含从1到9的数字,将一个盒子放到一个一维数组中(第49~53行),然后调用`is1To9`方法来检测它是否包含从1到9的数字(第55行)。

如何将所有的方格都放置在同一个盒子里呢?首先,定位 3×3 盒子的起始方格。它们的位置在 $(3i, 3j)$ (其中 $i=0, 1, 2$; $j=0, 1, 2$),如图7-7所示。

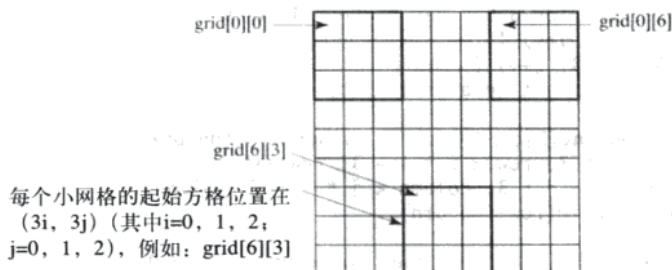


图7-7 在 3×3 盒子中的第一个方格的位置决定了盒子中其他格子位置

随着这种观测的深入,可以很容易地确定盒子中的所有元素。假设`grid[r][c]`是 3×3 盒子的起始方格。这个盒子中的元素可以使用嵌套循环来遍历,如下所示:

```
// Get all cells in a 3-by-3 box starting at grid[r][c]
for (int row = r; row < r + 3; row++)
    for (int column = c; column < c + 3; column++)
        // grid[row][column] is in the box
```

所有小盒子中的数字都集中在一个一维数组`list`中(第53行),然后调用`is1To9(list)`检测`list`是否包含从1到9的数字(第55行)。

`is1To9(list)`方法(第64~78行)检测数组`list`是否的确包含从1到9的数字。它首先将`list`复制给一个新数组`temp`,然后对`temp`排序。注意:如果对`list`排序,那么`grid`中的内容将改变。在对`temp`排序后,如果`temp`的确包含从1到9的数字,那么`temp`中的数字就应该是1,2,...,9。第73~75行的循环会检测它是否是这种情况。

从控制台输入81个数字是很繁琐的。测试这个程序时,可以将输入存储在一个名为`CheckSudokuSolution.txt`的文件中,然后使用下面的命令运行这个程序:

```
java CheckSudokuSoluton < CheckSudokuSoluton.txt
```

7.8 多维数组

在7.7节中,我们使用二维数组表示矩阵或者表格。有时,可能还需要表示*n*维的数据结构。在Java中,可以创建*n*维数组,其中*n*是任意整数。

可以对二维数组变量的声明以及二维数组的创建方法进行推广,用于声明*n*维数组变量和创建 $n \geq 3$ 的*n*维数组。例如,下述语法声明一个三维数组变量`scores`,创建一个数组并将它的引用赋值给`scores`:

```
double[][][] scores = new double[10][24][2];
```

多维数组实际上是一个数组，它的每个元素都是另一个数组。三维数组是由二维数组构成的数组，每个二维数组又是一维数组的数组。例如，假设`x=new int[2][2][5]`，则`x[0]`和`x[1]`是二维数组，`x[0][0]`、`x[0][1]`、`x[1][0]`和`x[1][1]`都是一维数组，而且它们都含5个元素。`x.length`的值为2，`x[0].length`和`x[1].length`的值为2，`x[0][0].length`、`x[0][1].length`、`x[1][0].length`和`x[1][1].length`的值为5。

7.8.1 问题：每日温度和湿度

假设气象站每天每小时都会报告温度和湿度，并且将过去十天的数据都存储在一个名为`weather.txt`的文本文件中。文件中的每一行包含四个数字，分别表明日期、小时、温度和湿度。这个文件的内容如图a所示：

```
1 1 76.4 0.92
1 2 77.7 0.93
...
10 23 97.7 0.71
10 24 98.7 0.74
```

a)

```
10 24 98.7 0.74
1 2 77.7 0.93
...
10 23 97.7 0.71
1 1 76.4 0.92
```

b)

注意 文件的行不一定要顺序排列的。例如：文件可以如图b所示。

你的任务是编写程序，计算这10天的平均日温度和平均日湿度。可以使用输入重定向来读取文件，并将这些数据存在一个名为`data`的三维数组中。`data`的第一个下标范围从0到9，表示10天；第二个下标范围从0到23，表示24小时；而第三个下标范围从0到1，分别表示温度和湿度。注意：在文件中，天是从1到10编号的，而小时是从1到24编号的。因为数组下标是从0开始的，所以，`data[0][0][0]`存储的是第一天第一个小时的温度，而`data[9][23][1]`存储的是第10天第24小时的湿度。

该程序在程序清单7-5中给出。

程序清单7-5 Weather.java

```
1 import java.util.Scanner;
2
3 public class Weather {
4     public static void main(String[] args) {
5         final int NUMBER_OF_DAYS = 10;
6         final int NUMBER_OF_HOURS = 24;
7         double[][][] data
8             = new double[NUMBER_OF_DAYS][NUMBER_OF_HOURS][2];
9
10        Scanner input = new Scanner(System.in);
11        // Read input using input redirection from a file
12        for (int k = 0; k < NUMBER_OF_DAYS * NUMBER_OF_HOURS; k++) {
13            int day = input.nextInt();
14            int hour = input.nextInt();
15            double temperature = input.nextDouble();
16            double humidity = input.nextDouble();
17            data[day - 1][hour - 1][0] = temperature;
18            data[day - 1][hour - 1][1] = humidity;
19        }
20
21        // Find the average daily temperature and humidity
22        for (int i = 0; i < NUMBER_OF_DAYS; i++) {
23            double dailyTemperatureTotal = 0, dailyHumidityTotal = 0;
24            for (int j = 0; j < NUMBER_OF_HOURS; j++) {
25                dailyTemperatureTotal += data[i][j][0];
26                dailyHumidityTotal += data[i][j][1];
27            }
28
29            // Display result
30            System.out.println("Day " + i + "'s average temperature is ")
```



```

31         + dailyTemperatureTotal / NUMBER_OF_HOURS);
32     System.out.println("Day " + i + "'s average humidity is "
33         + dailyHumidityTotal / NUMBER_OF_HOURS);
34 }
35 }
35 }

```

```

Day 0's average temperature is 77.7708
Day 0's average humidity is 0.929583
Day 1's average temperature is 77.3125
Day 1's average humidity is 0.929583
...
Day 9's average temperature is 79.3542
Day 9's average humidity is 0.9125

```



可以使用下面的命令来运行这个程序：

```
java Weather < Weather.txt
```

在第8行创建存储温度和湿度的三维数组。在第12~19行的循环中将输入读给数组。可以从键盘键入输入，但是这样做不是很便利。为了方便起见，将这些数据存储在文件中，并使用输入重定向从文件中读取数据。在第24~27行的循环中，将一天中每个小时的温度都加到dailyTemperatureTotal中，并将每个小时的湿度都加到dailyHumidityTotal中。第30~33行显示平均日温度和日湿度。

7.8.2 问题：猜生日

程序清单3-3给出一个猜生日的程序。可以通过用三维数组来存储5个集合的数字来简化程序，然后使用循环提示用户回答，如程序清单7-6所示。该程序的运行示例和程序清单3-3所显示的是一样。

程序清单7-6 GuessBirthdayUsingArray.java

```

1 import java.util.Scanner;
2
3 public class GuessBirthdayUsingArray {
4     public static void main(String[] args) {
5         int day = 0; // Day to be determined
6         int answer;
7
8         int[][][] dates = {
9             {{ 1, 3, 5, 7},
10             { 9, 11, 13, 15},
11             {17, 19, 21, 23},
12             {25, 27, 29, 31}},
13             {{ 2, 3, 6, 7},
14             {10, 11, 14, 15},
15             {18, 19, 22, 23},
16             {26, 27, 30, 31}},
17             {{ 4, 5, 6, 7},
18             {12, 13, 14, 15},
19             {20, 21, 22, 23},
20             {28, 29, 30, 31}},
21             {{ 8, 9, 10, 11},
22             {12, 13, 14, 15},
23             {24, 25, 26, 27},
24             {28, 29, 30, 31}},
25             {{16, 17, 18, 19},
26             {20, 21, 22, 23},
27             {24, 25, 26, 27},
28             {28, 29, 30, 31}}};
29
30         // Create a Scanner
31         Scanner input = new Scanner(System.in);
32
33         for (int i = 0; i < 5; i++) {

```




```

34     System.out.println("Is your birthday in Set" + (i + 1) + "?");
35     for (int j = 0; j < 4; j++) {
36         for (int k = 0; k < 4; k++)
37             System.out.printf("%4d", dates[i][j][k]);
38         System.out.println();
39     }
40
41     System.out.print("\nEnter 0 for No and 1 for Yes: ");
42     answer = input.nextInt();
43
44     if (answer == 1)
45         day += dates[i][0][0];
46     }
47
48     System.out.println("Your birth day is " + day);
49 }
50 }

```

在第8~28行创建三维数组**dates**。这个数组存储5个集合的数字。每个集合都是一个 4×4 的二维数组。

循环从第33行开始，显示每个集合的数字，然后提示用户回答这个生日是否在该集合中（第41~42行）。如果该天是在某个集合中，那么这个集合的第一个数字（**dates[i][0][0]**）就被加到变量**day**中（第45行）。

本章小结

- 使用二维数组来存储表格。
- 可以使用以下语法来声明二维数组变量：

元素类型[][]数组变量。

- 可以使用以下语法来创建二维数组变量：

new 元素类型[行的个数][列的个数]。

- 使用下面的语法表示二维数组中的每个元素：

数组变量[行下标][列下标]。

- 可以使用数组初始化语法来创建和初始化二维数组：

元素类型[][]数组变量={ {某行的值}, ..., {某行的值} }。

- 可以使用数组的数组构成多维数组。例如：一个三维数组变量可以声明为“元素类型[][][]数组变量”，并使用“**new**元素类型[size1][size2][size3]”来创建三维数组。

复习题

- 7.1 声明并创建一个 4×5 的整型矩阵。
- 7.2 二维数组的行可以有不同的长度吗？
- 7.3 以下代码的输出是什么？

```

int[][] array = new int[5][6];
int[] x = {1, 2};
array[0] = x;
System.out.println("array[0][1] is " + array[0][1]);

```

- 7.4 以下语句中，哪些是合法的数组声明？

```

int[][] r = new int[2];
int[] x = new int[];
int[][] y = new int[3][];

```

- 7.5 为什么is1To9方法需要将list复制到temp？如果用下面的代码替换程序清单7-4中第66~70行的代



码,会发生什么?

```
java.util.Arrays.sort(list);
```

7.6 声明并创建一个 $4 \times 6 \times 5$ 的整型矩阵。


编程练习题

*7.1 (求矩阵中所有数的和) 编写一个方法,求整数矩阵中所有整数的和,使用下面的方法头:

```
public static double sumMatrix(int[][] m)
```

编写一个测试程序,读取一个 4×4 的矩阵,然后显示所有元素的和。下面是一个运行示例:

```
Enter a 4-by-4 matrix row by row:
1 2 3 4 --Enter
5 6 7 8 --Enter
9 10 11 12 --Enter
13 14 15 16 --Enter
Sum of the matrix is 136
```




*7.2 (求矩阵主对角线元素的和) 编写一个方法,求 $n \times n$ 的整数矩阵中主对角线上所有整数的和,使用下面的方法头:

```
public static int sumMajorDiagonal(int[][] m)
```

编写一个测试程序,读取一个 4×4 的矩阵,然后显示它的主对角线上的所有元素的和。下面是一个运行示例:

```
Enter a 4-by-4 matrix row by row:
1 2 3 4 --Enter
5 6 7 8 --Enter
9 10 11 12 --Enter
13 14 15 16 --Enter
Sum of the elements in the major diagonal is 34
```



*7.3 (按考分对学生排序) 重写程序清单7-2,以正确答案个数的升序显示学生。

**7.4 (计算每个雇员每周工作的小时数) 假定所有雇员每周工作的小时数存储在一个二维数组中。每行将一个雇员7天的工作时间记录在7列中。例如:下面显示的数组存储了8个雇员的工作时间。编写一个程序,按照总工时降序的方式显示雇员和他们的总工时。

	Su	M	T	W	H	F	Sa
Employee 0	2	4	3	4	5	8	8
Employee 1	7	3	4	3	3	4	4
Employee 2	3	3	4	3	3	2	2
Employee 3	9	3	4	7	3	4	1
Employee 4	3	5	4	3	6	3	8
Employee 5	3	4	4	6	3	4	4
Employee 6	3	7	4	8	3	8	4
Employee 7	6	3	5	9	2	7	9

7.5 (代数方面:两个矩阵相加) 编写两个矩阵相加的方法。方法头如下:

```
public static double[][] addMatrix(double[][] a, double[][] b)
```

为了能够进行相加,两个矩阵必须具有相同的维数,并且元素具有相同或兼容的数据类型。假设 c 表示最终的矩阵,每个元素 c_{ij} 就是 $a_{ij}+b_{ij}$ 。例如,对于两个 3×3 的矩阵 a 和 b , c 就有:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} \end{pmatrix}$$

编写一个测试程序，提示用户输入两个 3×3 的矩阵，然后显示它们的和。下面是一个运行示例：

```

Enter matrix1: 1 2 3 4 5 6 7 8 9 --Enter
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 --Enter
The matrices are added as follows
1.0 2.0 3.0      0.0 2.0 4.0      1.0 4.0 7.0
4.0 5.0 6.0 +    1.0 4.5 2.2 =    5.0 9.5 8.2
7.0 8.0 9.0      1.1 4.3 5.2      8.1 12.3 14.2

```



****7.6**（代数方面：两个矩阵相乘）编写两个矩阵相乘的方法。方法头如下：

```
public static double[][] multiplyMatrix(double[][] a, double[][] b)
```

为了使矩阵 a 能够与矩阵 b 相乘，矩阵 a 的列数必须与矩阵 b 的行数相同，并且两个矩阵的元素要具有相同或兼容的数据类型。假设矩阵 c 是相乘的结果，而 a 的列数是 n ，那么每个元素 c_{ij} 就是 $a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \dots + a_{in} \times b_{nj}$ 。例如，对于两个 3×3 的矩阵 a 和 b ， c 就有：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

这里的 $c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$ 。

编写一个测试程序，提示用户输入两个 3×3 的矩阵，然后显示它们的乘积。下面是一个运行示例：

```

Enter matrix1: 1 2 3 4 5 6 7 8 9 --Enter
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 --Enter
The matrices are multiplied as follows:
1 2 3      0 2.0 4.0      5.3 23.9 24
4 5 6 *    1 4.5 2.2 =    11.6 56.3 58.2
7 8 9      1.1 4.3 5.2      17.9 88.7 92.4

```



***7.7**（距离最近的两个点）程序清单7-3给出找到二维空间中距离最近的两个点的程序。修改该程序，让程序能够找出在三维空间上距离最近的两个点。使用一个二维数组表示这些点。使用下面的点来测试这个程序：

```
double[][] points = {{-1, 0, 3}, {-1, -1, -1}, {4, 1, 1},
{2, 0.5, 9}, {3.5, 2, -1}, {3, 1.5, 3}, {-1.5, 4, 2},
{5.5, 4, -0.5}};
```

计算两个点 (x_1, y_1, z_1) 和 (x_2, y_2, z_2) 之间距离的公式是 $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$ 。

****7.8**（所有最近的点对）修改程序清单7-3，找出所有具有相同最小距离的点对。

*****7.9**（游戏：玩井字游戏）在井字游戏中，两个玩家使用各自的标志（一方用X则另一方就用O），轮流填写 3×3 的网格中的某个空格。当一个玩家在网格的水平方向，垂直方向或者对角线方向上出现了三个相同的X或三个相同的O，表明游戏结束，该玩家获胜。平局（没有赢家）是指当网格中所有的空格都被填满时，没有任何一方的玩家获胜的情况。创建一个玩井字游戏的程序。程序提示两个玩家可以选择X和O作为他们的标志。当输入一个标志时，程序在控制台上重新显示棋盘，然后确定游戏的状态（是获胜、平局还是继续）。下面是一个运行示例：



```

| | | |
| | | |
| | | |

Enter a row (1, 2, or 3) for player X: 1 Enter
Enter a column (1, 2, or 3) for player X: 1 Enter

| | | |
| | X | |
| | | |

Enter a row (1, 2, or 3) for player O: 1 Enter
Enter a column (1, 2, or 3) for player O: 2 Enter

| | | |
| | X | O |
| | | |

```

```

Enter a row (1, 2, or 3) for player X:
...
| X | | |
| O | X | O |
| | | X |

X player won

```

*7.10 (游戏: 井字游戏棋盘) 编写程序, 随机地在井字游戏棋盘上填入0或者1, 打印该棋盘, 然后找出全是0或者全是1的行、列或对角线。使用一个二维数组表示一个井字游戏的棋盘。下面是这个程序的一个运行示例:

```

001
001
111
All 1s on row 2
All 1s on column 2

```

**7.11 (游戏: 九个正面和背面) 一个 3×3 的矩阵中放置了9个硬币, 这些硬币有些面向上, 有些面向下。可以使用 3×3 的矩阵中的0 (正面) 或1 (反面) 表示硬币的状态。下面是一些例子:

```

0 0 0   1 0 1   1 1 0   1 0 1   1 0 0
0 1 0   0 0 1   1 0 0   1 1 0   1 1 1
0 0 0   1 0 0   0 0 1   1 0 0   1 1 0

```

每个状态都可以使用一个二进制数表示。例如, 前面的矩阵对应到数字

```
000010000 101001100 110100001 101110100 100111110
```

总共会有512种可能性。所以, 可以使用十进制数0, 1, 2, 3, ..., 511来表示这个矩阵的所有状态。编写一个程序, 提示用户输入一个在0到511之间的数字, 然后显示用字符H和T表示的对应的矩阵。下面是一个运行示例:

```
Enter a number between 0 and 511: 7 
H H H
H H H
T T T
```



用户输入7，它代表的是000000111。因为0代表H而1代表T，所以输出正确。

****7.12** (财务应用程序：计算税款) 使用数组重写程序清单3-6。针对每个登记者的身份，都有六种税率。每个税率都应用在某个特定范围内的可征税收入。例如：对一个可征税税收为400 000美元的单身登记者来讲，8350美元的税率是10%，8350~33 950之间的税率是15%，33 950~82 250之间的税率是25%，82 250~171 550之间的税率是28%，82 250~372 550之间的税率是33%，而372 950~400 000之间的税率是36%。这六种税率对所有的登记身份都是一样的，可以用下面的数组来表示：

```
double[] rates = {0.10, 0.15, 0.25, 0.28, 0.33, 0.35};
```

所有登记者身份的每个利率括号都可以用一个二维数组表示，如下所示：

```
int[][] brackets = {
    {8350, 33950, 82250, 171550, 372950}, // Single filer
    {16700, 67900, 137050, 20885, 372950}, // Married jointly
    {8350, 33950, 68525, 104425, 186475}, // Married separately
    {11950, 45500, 117450, 190200, 372950} // Head of household
};
```

假设单身身份的登记者的可征税收入是400 000美元，该税收可以如下计算：

```
tax = brackets[0][0] * rates[0] +
      (brackets[0][1] - brackets[0][0]) * rates[1] +
      (brackets[0][2] - brackets[0][1]) * rates[2] +
      (brackets[0][3] - brackets[0][2]) * rates[3] +
      (brackets[0][4] - brackets[0][3]) * rates[4] +
      (400000 - brackets[0][4]) * rates[5]
```

***7.13** (定位最大的元素) 编写下面的方法，返回二维数组中最大元素的位置。

```
public static int[] locateLargest(double[][] a)
```

返回值是包含两个元素的一维数组。这两个元素表示二维数组中最大元素的行下标和列下标。编写一个测试程序，提示用户输入一个二维数组，然后显示这个数组中最大元素的位置。下面是一个运行示例：

```
Enter the number of rows and columns of the array: 3 4 
Enter the array:
23.5 35 2 10 
4.5 3 45 3.5 
35 44 5.5 9.6 
The location of the largest element is at (1, 2)
```



****7.14** (探究矩阵) 编写程序，提示用户输入一个方阵的长度，随机地在矩阵中填入0和1，打印这个矩阵，然后找出整行、整列或者对角线都是0或1的行、列和对角线。下面是这个程序的一个运行示例：

```
Enter the size for the matrix: 4 
0111
0000
0100
1111
All 0s on row 1
All 1s on row 3
No same numbers on a column
No same numbers on the major diagonal
No same numbers on the sub-diagonal
```



*7.15 (几何方面: 在同一行吗?) 假设给出一个点的集合, 编写程序, 检测是否所有的点都在同一行上。使用下面的集合测试程序:

```
double[][] set1 = {{1, 1}, {2, 2}, {3, 3}, {4, 4}};
double[][] set2 = {{0, 1}, {1, 2}, {4, 5}, {5, 6}};
double[][] set3 = {{0, 1}, {1, 2}, {4, 5}, {4.5, 4}};
```

*7.16 (对二维数组排序) 编写一个方法, 使用下面的方法头对二维数组排序:

```
public static void sort(int m[][])
```

这个方法首先按行排序, 然后按列排序。例如: 数组{{4, 2}, {1, 7}, {4, 5}, {1, 2}, {1, 1}, {4, 1}}排序后就成了{{1, 1}, {1, 2}, {1, 7}, {4, 1}, {4, 2}, {4, 5}}。

***7.17 (金融风暴) 银行会互相借钱。在经济艰难时期, 如果一个银行倒闭, 它就不能偿还贷款。一个银行的总资产是它当前的余款减去它欠其他银行的贷款。图7-8就是五个银行的情况图。每个银行的当前余额分别是2500万美金、1亿2500万美金、1亿7500万美金、7500万美金和1亿8100万美金。从节点1到节点2的方向的边表示银行1借给银行2共计4千万美金。

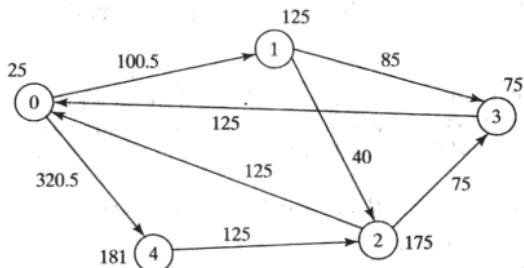


图7-8 银行之间互相借款

如果银行的总资产在某个限定范围以下, 那么这个银行就是不安全的。它借的钱就不能返还给借贷方, 而且这个借贷方也不能将这个贷款算入它的总资产。因此, 如果借贷方总资产在限定范围以下, 那么它也不安全。编写程序, 找出所有不安全的银行。程序如下读取输入。它首先读取两个整数 n 和 $limit$, 这里的 n 表示银行个数, 而 $limit$ 表示要保持银行安全的最小总资产。然后, 程序会输出描述 n 个银行的 n 行信息, 银行的 id 从0到 $n-1$ 。每一行的第一个数字都是银行的余额, 第二个数字表明从该银行借款的银行, 其余的就都是两个数字构成的数对。每对都描述一个借款方。每一对数字的第一个数就是借款方的 id , 第二个数就是所借的钱数。例如, 在图7-8中五个银行的输入如下所示 (注意: $limit$ 是201):

```
5 201
25 2 1 100.5 4 320.5
125 2 2 40 3 85
175 2 0 125 3 75
75 1 0 125
181 1 2 125
```

银行3的总资产是 $75+125$, 这个数字是在201以下的。所以, 银行3是不安全的。在银行3变得不安全之后, 银行1的总资产也降为 $125+40$ 。所以, 银行1也不安全。程序的输出应该是:

```
Unsafe banks are 3 1
```

提示 使用一个二维数组 $borrowers$ 来表示贷款。 $borrowers[i][j]$ 表明银行 i 贷款给银行 j 的贷款额。一旦银行 j 变得不安全, 那么 $borrowers[i][j]$ 就应该设置为0。

*7.18 (打乱行) 编写一个方法, 使用下面的方法头打乱一个二维整型数组的行:

```
public static void shuffle(int[][] m)
```

编写一个测试程序, 打乱下面的矩阵:

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};
```

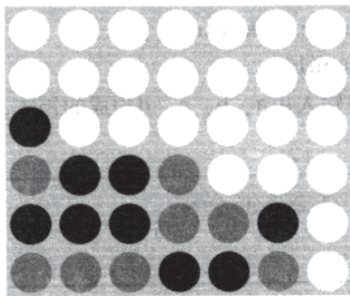
****7.19 (模式识别：连续四个相等的数)** 编写下面的方法，测试一个二维数组是否有四个连续的数字具有相同的值，这四个数可以是水平方向的、垂直方向的或者对角线方向的。

public static boolean isConsecutiveFour(int[][] values)

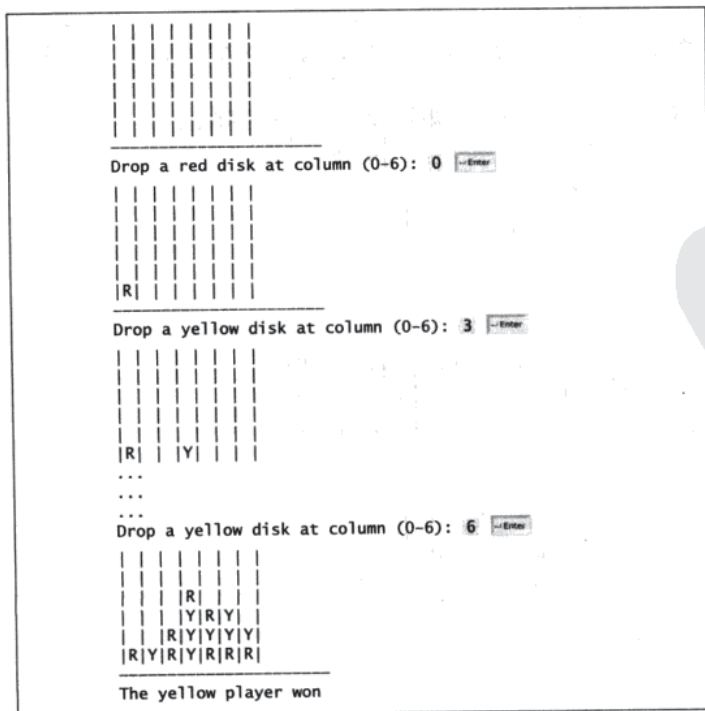
编写一个测试程序，提示用户输入一个二维数组的行数、列数以及数组中的值。如果这个数组有四个连续的数字具有相同的值，就显示true；否则，显示false。下面是结果为true的一些例子：

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

*****7.20 (游戏：连接四子)** 连接四子是一个两个人玩的棋盘游戏，在游戏中，玩家轮流将有颜色的棋子放在一个六行七列的垂直悬挂的网格中，如下所示。



这个游戏的目的是在反方出现一行、一列或者一条对角线上有四个相同颜色的棋子之前，正方能先做到。程序提示两个玩家交替地下红子Red或黄子Yellow。当丢下一子时，程序在控制台重新显示这个棋盘，然后确定游戏的状态（赢、平局还是继续）。下面是一个运行示例：



数字资源
PDG

***7.21 (游戏: 多种九宫格解决方案) 完整的九宫格问题的解决方案在补充材料VII.A中给出。一个九宫格问题可以有多种解决方案。修改补充材料VII.A中的Sudoku.java文件, 显示所有解决方案的总个数。如果存在多种解决方案, 显示两个解决方案。

*7.22 (代数问题: 2×2 矩阵的逆阵) 方阵A的逆阵用 A^{-1} 来表示, 满足 $A \times A^{-1} = I$, 这里的I是指一个单位矩阵, 它的对角线上的值都为1, 而其他位置的值都为0。例如: 矩阵 $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 的逆阵是 $\begin{bmatrix} -0.5 & 1 \\ 1.5 & 0 \end{bmatrix}$, 即

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} -0.5 & 1 \\ 1.5 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

可以使用下面的公式得到一个 2×2 的矩阵A的逆阵:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad A^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

实现下面的方法可以获取该矩阵的逆阵:

```
public static double[][] inverse(double[][] A)
```

如果 $ad-bc$ 为0, 则该方法返回null。

编写一个测试程序, 提示用户输入由a、b、c、d构成的矩阵, 然后显示该矩阵的逆阵。下面是一个运行示例:

```
Enter a, b, c, d: 1 2 3 4
-2.0 1.0
1.5 -0.5
```



```
Enter a, b, c, d: 0.5 2 1.5 4.5
-6.0 2.6666666666666665
2.0 -0.6666666666666666
```



*7.23 (代数问题: 3×3 矩阵的逆阵) 方阵A的逆阵表示为 A^{-1} , 满足 $A \times A^{-1} = I$, 其中I是所有对角线

上的值为1而其他所有值为0的单位矩阵。例如, 矩阵 $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 1 \\ 4 & 5 & 3 \end{bmatrix}$ 的逆阵是:

$$\begin{bmatrix} -2 & 0.5 & 0.5 \\ 1 & 0.5 & -0.5 \\ 1 & -1.5 & 0.5 \end{bmatrix}$$

也就是:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 1 \\ 4 & 5 & 3 \end{bmatrix} \times \begin{bmatrix} -2 & 0.5 & 0.5 \\ 1 & 0.5 & -0.5 \\ 1 & -1.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

对于一个 3×3 的矩阵:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

使用下面的公式可以得到其逆阵, 其中 $|A| \neq 0$:

数字资源

PDG

$$A^{-1} = \frac{1}{|A|} \begin{bmatrix} a_{22}a_{33} - a_{23}a_{32} & a_{13}a_{32} - a_{12}a_{33} & a_{12}a_{23} - a_{13}a_{22} \\ a_{23}a_{31} - a_{21}a_{33} & a_{11}a_{33} - a_{13}a_{31} & a_{13}a_{21} - a_{11}a_{23} \\ a_{21}a_{32} - a_{22}a_{31} & a_{12}a_{31} - a_{11}a_{32} & a_{11}a_{22} - a_{12}a_{21} \end{bmatrix}$$

$$|A| = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{31}a_{12}a_{23} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{11}a_{23}a_{32} - a_{33}a_{21}a_{12}$$

实现下面的方法可得到矩阵的逆阵：

```
public static double[][] inverse(double[][] A)
```

如果 $|A|$ 为0，那么方法返回null。

编写一个测试程序，提示用户输入一个矩阵的 a_{11} ， a_{12} ， a_{13} ， a_{21} ， a_{22} ， a_{23} ， a_{31} ， a_{32} ， a_{33} ，然后显示这个矩阵的逆阵。下面是一个运行示例：

```
Enter a11, a12, a13, a21, a22, a23, a31, a32, a33:
1 2 1 2 3 1 4 5 3
-2 0.5 0.5
1 0.5 -0.5
1 -1.5 0.5
```



```
Enter a11, a12, a13, a21, a22, a23, a31, a32, a33:
1 4 2 2 5 8 2 1 8
2.0 -1.875 1.375
0.0 0.25 -0.25
-0.5 0.4375 -0.1875
```



数字资源

PDG