```
13        model.getShortestPath(NineTailModel.getIndex(initialNode));
14
15      System.out.println("The steps to flip the coins are ");
16      for (int i = 0; i < path.size(); i++)
17        NineTailModel.printNode(
18          NineTailModel.getNode(path.get(i).intValue()));
19    }
20  }
```

该程序提示用户输入一个包含 9 个 H 和 T 字母的初始结点，就像第 8 行中的字符串，从字符串中得到一个字符数组（第 9 行），用一个模型来创建图并得到广度优先搜索树（第 11 行），得到一个从初始结点到目标结点的最短路径（第 12 ～ 13 行），然后显示这个路径上的结点（第 16 ～ 18 行）。

复习题

28.26 NineTailModel 中图的结点是如何创建的？

28.27 NineTailModel 中图的边是如何创建的？

28.28 在程序清单 28-13 中调用 getIndex("HTHTTTHHH".toCharArray()) 会返回什么？在程序清单 28-13 中调用 getNode(46) 会返回什么？

28.29 程序清单 28-13 中第 26 行和第 27 行交换，程序还会工作吗？为什么？

关键术语

adjacency list（邻接线性表）
adjacency matrix（邻接矩阵）
adjacent vertices（邻接顶点）
breadth-first search（广度优先搜索）
complete graph（完全图）
cycle（回路）
degree（度）
depth-first search（深度优先搜索）
directed graph（有向图）
graph（图）
incident edges（连接边）
parallel edge（平行边）
Seven Bridges of Königsberg（哥尼斯堡七孔桥问题）
simple graph（简单图）
spanning tree（生成树）
tree（树）
undirected graph（无向图）
unweighted graph（非加权图）
weighted graph（加权图）

本章小结

1. 图是一种有用的数学结构，可以表示现实世界中实体之间的联系。已经学习了如何使用类和接口来对图建模，如何使用数组和链表来表示顶点和边，以及如何实现图的操作。
2. 图的遍历是指访问图中的每个顶点一次并且只有一次的过程。学习了两种遍历图的常用方法：深度优先搜索（DFS）和广度优先搜索（BFS）。
3. 深度优先搜索和广度优先搜索可以解决许多问题，如检测图是否连通，检测图中是否存在环，找出两个顶点之间最短路径等。

测试题

回答位于网址 www.cs.armstrong.edu/liang/intro10e/quiz.html 的本章测试题。

编程练习题

28.6 ～ 28.10 节

*28.1 （检测一个图是否是连通的）编写一个程序，它从文件读入图并且判定该图是否是连通的。文件

中的第一行包含了表明顶点个数的数字（n）。顶点被标记为 0,1,…,n-1。接下来的每一行，以 u v1 v2… 的形式描述边（u,v1）、（u,v2），以此类推。图 28-21 给出了两个文件对应的图的例子。

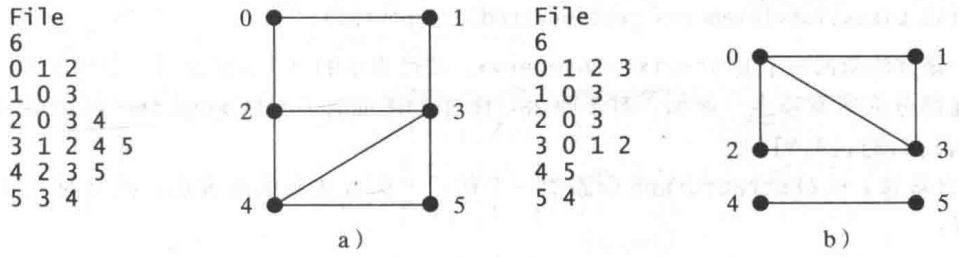


图 28-21 图的顶点和边存储在一个文件中

程序应该提示用户输入文件的名字，应该从文件中读取数据，创建 UnweightedGraph 的一个实例 g，然后调用 g.printEdges() 来显示所有的边，并调用 dfs() 来获取 AbstractGraph.Tree 的一个实例 tree。如果 tree.getNumberOfVerticeFound() 与图中的顶点数目相同，那么图就是连通的。下面是这个程序的运行示例：

```
Enter a file name: c:\exercise\GraphSample1.txt Enter
The number of vertices is 6
Vertex 0: (0, 1) (0, 2)
Vertex 1: (1, 0) (1, 3)
Vertex 2: (2, 0) (2, 3) (2, 4)
Vertex 3: (3, 1) (3, 2) (3, 4) (3, 5)
Vertex 4: (4, 2) (4, 3) (4, 5)
Vertex 5: (5, 3) (5, 4)
The graph is connected
```

提示：使用 new UnweightedGraph(list,numberOfVertices) 来创建一个图，其中 list 是包含 AbstractGraph.Edge 对象的一个线性表。使用 new AbstractGraph.Edge(u,v) 来创建一条边。读取第一行来获取顶点的数目。将接下来的每一行读入一个字符串 s 中并且使用 s.split("[\\s+]") 来从字符串中提取顶点并产生从顶点开始的边。

*28.2 （为图创建文件）修改程序清单 28-1 来创建一个文件 graph1。文件形式在编程练习题 28.1 中描述。从程序清单 28-1 中第 8 ～ 21 行定义的数组创建这个文件。图的顶点数为 12，它存储在文件的第一行。文件的内容应该如下所示：

```
12
0 1 3 5
1 0 2 3
2 1 3 4 10
3 0 1 2 4 5
4 2 3 5 7 8 10
5 0 3 4 6 7
6 5 7
7 4 5 6 8
8 4 7 9 10 11
9 8 11
10 2 4 8 11
11 8 9 10
```

*28.3 （使用堆栈实现深度优先搜索）程序清单 28-8 描述的深度优先搜索使用的是递归。设计一个新的算法而不使用递归实现它。使用伪代码描述该算法。通过定义一个名为 UnweightedGraphWithNonrecursiveDFS 的新类来实现它，该类继承自 UnweightedGraph 并且覆盖 dfs 方法。

*28.4 （寻找连通部分）创建一个名为 MyGraph 的新类作为 UnwightedGraph 的子类，其中包含找到图中所有连通部分的方法，方法头如下：

```
public List<List<Integer>> getConnectedComponents();
```

该方法返回一个 List<List<Integer>>。线性表中的每个元素是另一个线性表，它包含了连通部分的所有顶点。例如，对于图 28-21b 中的图而言，getConnectedComponents() 返回 [[0,1,2,3],[4,5]]。

*28.5 （找出路径）在 AbstractGraph 中添加一个使用下面方法头的新方法，找出两个顶点之间的路径：

```
public List<Integer> getPath(int u, int v);
```

该方法会返回一个 List<Integer>，它包含由顶点 u 到顶点 v 的路径上的所有顶点。使用广度优先搜索方法，可以获取由顶点 u 到顶点 v 的最短路径。如果顶点 u 和顶点 v 之间不存在路径，方法返回 null。

*28.6 （探测回路）在 AbstractGraph 中添加一个使用下面方法头的新方法，判定图中是否存在环：

```
public boolean isCyclic();
```

*28.7 （找出回路）在 AbstractGraph 添加一个使用下面方法头的新方法，找出图中的环：

```
public List<Integer> getACycle(int u);
```

该方法返回一个 List，它包含从顶点 u 开始的回路上的所有顶点。如果图中没有回路，方法返回 null。

**28.8 （测试二分图）回顾一下，如果图的顶点可以分为两个不相交的集合，而且同一个集合中的顶点之间不存在边，那么这个图是二分的。在 AbstractGraph 中添加一个新方法来检测图是否是二分的：

```
public boolean isBipartite();
```

**28.9 （得到二分集合）在 AbstractGraph 中添加一个新方法，如果图是二分的，返回这两个二分集合：

```
public List<List<Integer>> getBipartite();
```

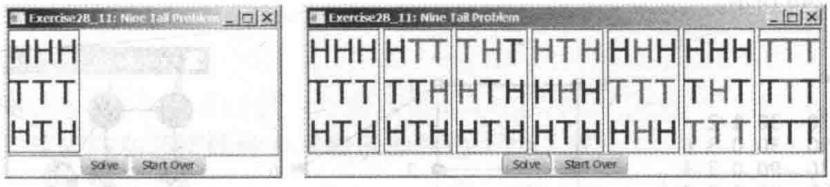
该方法返回一个包含两个子线性表的 List，每一个都包含了一个顶点集合。如果图不是二分的，方法返回 null。

28.10 （找出最短路径）编写一个程序，从文件中读取一个连通图。图存储在一个文件中，使用和编程练习题 28.1 中指定的一样的格式。程序应该提示用户输入文件名，然后输入两个顶点，最后显示两个顶点之间的最短路径。例如，对于图 28-21a 中的图，顶点 0 和顶点 5 之间的最短路径可以显示为 0 1 3 5。

下面是该程序的一个运行示例：

```
Enter a file name: c:\exercise\GraphSample1.txt Enter
Enter two vertices (integer indexes): 0 5 Enter
The number of vertices is 6
Vertex 0: (0, 1) (0, 2)
Vertex 1: (1, 0) (1, 3)
Vertex 2: (2, 0) (2, 3) (2, 4)
Vertex 3: (3, 1) (3, 2) (3, 4) (3, 5)
Vertex 4: (4, 2) (4, 3) (4, 5)
Vertex 5: (5, 3) (5, 4)
The path is 0 1 3 5
```

**28.11 （修改程序清单 28-14）程序清单 28-14 中的程序允许用户在控制台上为 9 枚硬币反面问题输入数据并且在控制台上显示结果。编写一个程序，让用户设置 9 枚硬币的初始状态（如图 28-22a 所示），然后单击 Solve 按钮来显示解决方案，如图 28-22b 所示。初始情况下，用户可以通过单击鼠标来翻转硬币。将翻转的单元设置为红色。



a) b)

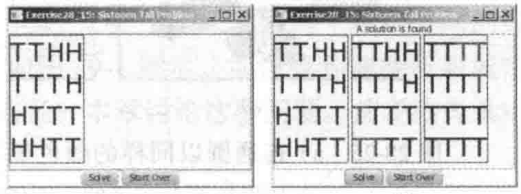
图 28-22 解决 9 枚硬币反面问题的程序

**28.12 （9 枚硬币反面问题的变体）在 9 枚硬币反面问题中，当翻转一个正面的硬币时，水平和垂直方向上的邻居也都被翻转。重新编写程序，假设对角线上的邻居也都被翻转。

**28.13 （4×4 16 枚硬币反面问题）程序清单 28-14，提供了 9 枚硬币反面问题的解答。修改该程序，成为一个 4×4 的矩阵中放置了 16 枚硬币。注意可能对于一个开始的模式并不存在解答。如果是这样，报告没有解答存在。

**28.14 （4×4 16 枚硬币反面问题的分析）本书中的 9 枚硬币反面问题使用的是 3×3 的矩阵。假设在一个 4×4 的矩阵中放置了 16 枚硬币。编写一个程序，找出不存在解答的开始模式的数目。

*28.15 （4×4 16 枚硬币反面问题的 GUI）修改编程练习题 28.14，使得用户可以设置 4×4 16 枚硬币反面问题的初始化模式（参见图 28-23a）。用户可以单击 Solve 按钮来显示解答，如图 28-23b 所示。开始时，用户可以点击鼠标按钮来翻转硬币。如果解答不存在，显示一条信息来报告该消息。



a) b)

图 28-23 解决 16 枚硬币反面问题的程序

**28.16 （诱导子图）给定一个无向图 $G=(V, E)$ 和一个整数 k，找出 G 的一个最大的诱导子图 H，H 中的所有结点的度≥ k，或者得到这样的子图不存在的结论。使用下面的文件头实现这个方法：

```
public static Graph maxInducedSubgraph(Graph g, int k)
```

如果这样的子图不存在，方法返回 null。

提示：一个直观的方法是删除那些度小于 k 的顶点。随着顶点及其邻接边被删除，其他顶点的度可能会减小。继续这个过程直到没有顶点被删除，或者所有的顶点都被删除。

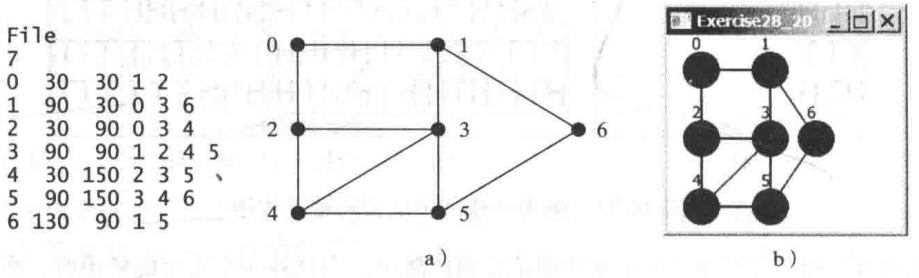
***28.17 （哈密尔顿环）补充材料 VI.E 给出了哈密尔顿路径算法的实现。在 Graph 接口中添加以下 getHamiltonianCycle 方法，并且在 AbstractGraph 类中实现它：

```
/** Return a Hamiltonian cycle
  * Return null if the graph doesn't contain a Hamiltonian cycle */
public List<Integer> getHamiltonianCycle()
```

***28.18 （骑士巡游回路）改写补充材料 VI.E 中示例学习的 KnightTourApp.java 程序，找出骑士访问棋盘的每个方块并且返回到起始方块的路径。将骑士巡游回路问题简化为寻找哈密尔顿环的问题。

**28.19 （显示一个图中的深度优先搜索 / 广度优先搜索树）修改程序清单 28-6 中的 GraphView，添加一个数据域 tree 和一个 set 方法。树中的边显示为红色。编写一个程序，显示图 28-1 中的图，以及从一个指定城市出发的深度优先搜索 / 广度优先搜索树，如图 28-13 和图 28-16 所示。如果输入了一个地图中没有的城市，程序在一个标签中给出错误信息。

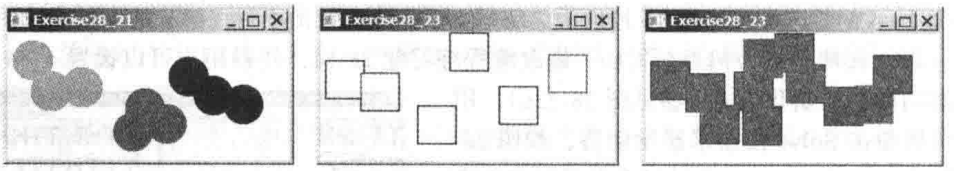
*28.20 （显示图）编写一个程序，从一个文件中读取一个图，然后显示它。文件的第一行包含了表示顶点个数的数字（n）。顶点被标记为 0，1，…，n-1。接下来的每一行，用 u x y v1 v2 的格式描述了 u 的位置（x，y）以及边（u，v1）、（u，v2），以此类推。图 28-24a 给出了对应图的文件的例子。程序提示用户输入文件名，从文件中读取数据并且使用 GraphView 在面板上显示图，如图 28-24b 所示。



a） b）

图 28-24 程序读取关于图的信息并且可视化显示它

**28.21 （显示连通圆集合）修改程序清单 28-10，以不同颜色显示连通圆的集合。也就是说，如果两个圆是连通的，则使用相同的颜色显示。否则，它们的颜色不同，如图 28-25 所示。（提示：参见编程练习题 28.4。）



a） b） c）

图 28-25 a）连通圆以同样的颜色显示；b）如果矩形不是连通的，则不使用颜色填充；c）如果矩形是连通的，则使用颜色填充

**28.22 （移动圆）修改程序清单 28-10，使得用户可以拖放和移动圆。

**28.23 （连通矩形）程序清单 28-10 允许用户创建圆并确定它们是否是连通的。为矩形重写该程序。程序使得用户可以在没有被矩形占据的空白区域点击鼠标来创建矩形。当矩形被添加，如果一些矩形是连通的则以填充方式绘制，否则不填充。如图 28-25b ～图 28-25c 所示。

*28.24 （删除圆）修改程序清单 28-10，使得用户可以通过在圆内点击删除圆。