关键术语

binary search tree（二叉查找树）
binary tree（二叉树）
breadth-first traversal（广度优先遍历）
depth-first traversal（深度优先遍历）
greedy algorithm（贪婪算法）
Huffman coding（霍夫曼编码）
inorder traversal（中序遍历）
postorder traversal（后序遍历）
preorder traversal（前序遍历）
tree traversal（树的遍历）

本章小结

1. 二叉查找树（BST）是一种分层的数据结构。学习了如何定义和实现 BST 类。学习了如何向 / 从 BST 插入和删除元素。学习了如何使用中序、后序、前序、深度优先以及广度优先搜索来遍历 BST。
2. 迭代器是一个提供了遍历像集合、线性表或二叉树这样的容器中的元素的统一方法的对象。学习了如何定义和实现遍历二叉树中元素的迭代器类。
3. 霍夫曼编码是一种压缩数据的方案，它使用较少的比特来编码经常出现的字符。字符的编码是使用二叉树基于它在文本中出现的次数来构建的，该二叉树称为霍夫曼编码树。

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

25.2 ～ 25.6 节

*25.1 （在 BST 中添加新方法）向 BST 类中添加以下新方法：

```
/** Displays the nodes in a breadth-first traversal */
public void breadthFirstTraversal()

/** Returns the height of this binary tree */
public int height()
```

*25.2 （测试完全二叉树）完全二叉树是指叶子结点都在同一层的二叉树。在 BST 类中添加一个方法，如果这棵树是完全二叉树，返回 true。
（提示：完全二叉树中的结点个数是 $2^{depth}-1$。）

```
/** Returns true if the tree is a full binary tree */
boolean isFullBST()
```

**25.3 （不使用递归实现中序遍历）使用栈替代递归，实现 BST 中的 inorder 方法。编写一个测试程序，提示用户输入 10 个整数，将它们保存在一个 BST 中，调用 inorder 方法来显示这些元素。

**25.4 （不使用递归实现前序遍历）使用栈替代递归，实现 BST 中的 preorder 方法。编写一个测试程序，提示用户输入 10 个整数，将它们保存在一个 BST 中，调用 preorder 方法来显示这些元素。

**25.5 （不使用递归实现后序遍历）使用栈替代递归，实现 BST 中的 postorder 方法。编写一个测试程序，提示用户输入 10 个整数，将它们保存在一个 BST 中，调用 postorder 方法来显示这些元素。

**25.6 （找出叶子结点）在 BST 类中添加一个方法，返回叶子结点的个数，如下所示：

```
/** Returns the number of leaf nodes */
public int getNumberOfLeaves()
```

**25.7 （找出非叶子结点）在 BST 类中添加一个方法，返回非叶子结点的个数，如下所示：

```
/** Returns the number of nonleaf nodes */
public int getNumberofNonLeaves()
```

***25.8 （实现双向迭代器）java.util.Iterator 接口定义了一个前向迭代器。Java API 也提供定义了一个定义双向迭代器的 java.util.ListIterator 接口。研究 ListIterator 并定义一个 BST 类的双向迭代器。

**25.9 （树的 clone 和 equals 方法）实现 BST 类中的 clone 和 equals 方法。两棵 BST 树如果包含相同的元素，则它们是相等的。clone 方法返回一棵 BST 树的完全一样的一个副本。

25.10 （前序迭代器）添加以下方法到 BST 类中，返回一个迭代器，用于前序遍历 BST 中的元素。

```
/** Returns an iterator for traversing the elements in preorder */
java.util.Iterator<E> preorderIterator()
```

25.11 （显示树）编写一个新的视图类，水平显示树，根在左边，如图 25-23 所示。

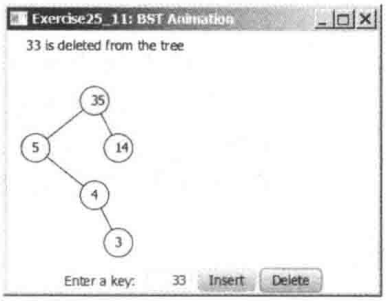


图 25-23 一棵二叉树水平显示

**25.12 （测试 BST）设计和编写一个完整的测试程序，测试程序清单 25-5 中的 BST 类是否符合所有要求。

**25.13 （在 BSTAnimation 中添加新按钮）修改程序清单 25-9，添加三个新按钮——Show Inorder、Show Preorder 和 Show Postorder——以便在标签中显示结果，如图 25-24 所示。还需要修改 BST.java 来实现 inorderList () 、preorderList () 和 postorderList () 方法，这样，这些方法就能以中序、前序和后序返回一个由结点元素构成的 List，如下所示：

```
public java.util.List<E> inorderList();
public java.util.List<E> preorderList();
public java.util.List<E> postorderList();
```

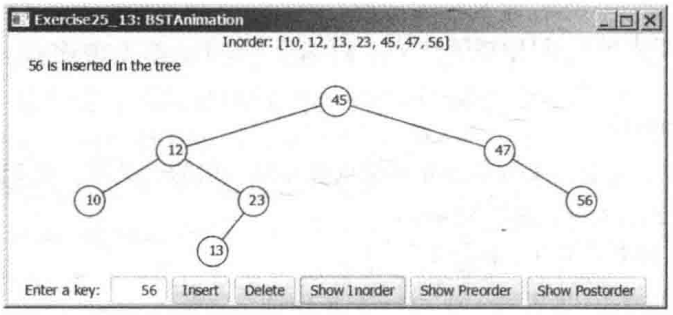
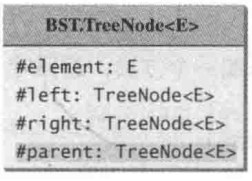


图 25-24 当单击图中的 Show Inorder、Show Preorder 或者 Show Postorder 按钮，就会在标签中分别以中序、前序和后序显示元素

*25.14 （使用 Comparator 的泛型 BST）修改程序清单 25-5 中的 BST，使用泛型参数和一个 Comparator 来比较对象。定义一个构造方法，使用 Comparator 作为它的参数，如下所示：

```
BST(Comparator<? super E> comparator)
```

***25.15 （BST 的父引用）通过添加一个到某结点的父结点的引用来重新定义 TreeNode，如下所示：



重新实现 BST 类中的 insert 和 delete 方法，为树中的每个结点更新父结点。在 BST 中添加以下新方法：

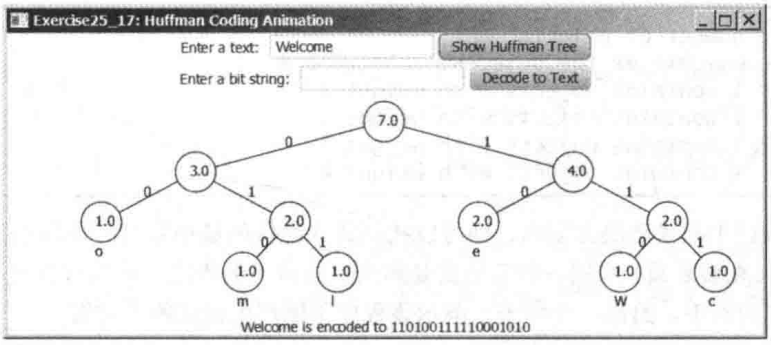
```
/** Returns the node for the specified element.
  * Returns null if the element is not in the tree. */
private TreeNode<E> getNode(E element)

/** Returns true if the node for the element is a leaf */
private boolean isLeaf(E element)

/** Returns the path of elements from the specified element
  * to the root in an array list. */
public ArrayList<E> getPath(E e)
```

编写一个测试程序，提示用户输入 10 个整数，将它们添加到树中，从树中删除第一个整数，然后显示到所有叶子结点的路径。下面是一个运行示例：

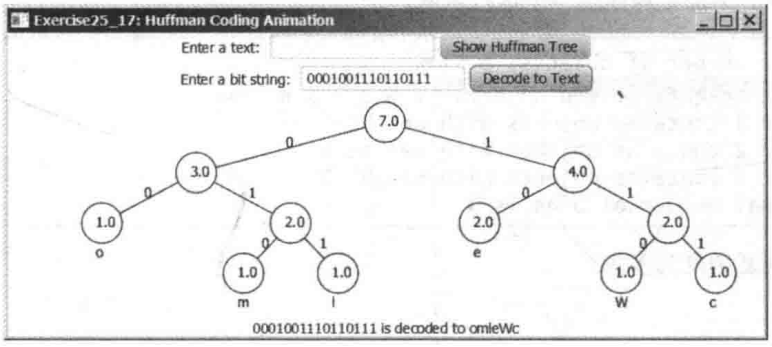
```
Enter 10 integers: 45 54 67 56 50 45 23 59 23 67 ↵Enter
[50, 54, 23]
[59, 56, 67, 54, 23]
```

***25.16 （数据压缩：霍夫曼编码）编写一个程序，提示用户输入一个文件名，显示文件中字符出现次数的表格，然后显示每个字符的霍夫曼编码。

***25.17 （数据压缩：霍夫曼编码的动画）编写一个程序，允许用户输入一个文本，然后显示基于该文本的霍夫曼编码树，如图 25-25a 所示。显示在一棵子树根结点的环中的子树的权重，显示每个叶子结点的字符，在标签中显示文本被编码后的比特。当用户单击 Decode Text 按钮时，一个比特字符串被解码为一个文本，显示在标签中，如图 25-25b 所示。



a)



b)

图 25-25 a）动画中显示给定文本的编码树，文本编码的比特显示在标签中；b）输入一个比特串，在标签中显示对应的文本

***25.18 （压缩一个文件）编写一个程序，使用霍夫曼编码将源文件压缩为目标文件。首先使用

ObjectOutputStream将霍夫曼编码输出到目标文件中，然后使用编程练习题17.17的BitOutputStream输出编码后的二进制内容到目标文件中。通过命令行传递文件信息：

```
java Exercise25_18 sourcefile targetfile
```

***25.19 （解压缩一个文件）前一个练习题压缩一个文件。压缩的文件包含了霍夫曼编码以及压缩的内容。编写一个程序，使用以下命令将一个源文件解压缩为目标文件：

```
java Exercise25_19 sourcefile targetfile
```

25.20 （应用首次满足法解决装箱问题）编写一个程序，将各种重量的物体装箱到容器中。每个容器可以容纳最多10磅。程序使用贪婪算法，将物体放置在它可以放下的第一个箱中。程序应该提示用户输入物体的总数以及每个物体的重量。程序显示需要装入物体的容器总数以及每个容器的内容。下面是一个程序的运行示例：

```
Enter the number of objects: 6
Enter the weights of the objects: 7 5 2 3 5 8
Container 1 contains objects with weight 7 2
Container 2 contains objects with weight 5 3
Container 3 contains objects with weight 5
Container 4 contains objects with weight 8
```

该程序可以产生最优解决方案吗，即可以找到装入物体的最小数目的容器吗？

25.21 （最小物体优先的装箱）采用一种新的贪婪算法重写前面的程序，将具有最小重量的物体放置在它首先适应的箱中。程序应该提示用户输入物体的总数以及每个物体的重量。程序显示需要装入物体的容器总数以及每个容器的内容。下面是一个程序的运行示例：

```
Enter the number of objects: 6
Enter the weights of the objects: 7 5 2 3 5 8
Container 1 contains objects with weight 2 3 5
Container 2 contains objects with weight 5
Container 3 contains objects with weight 7
Container 4 contains objects with weight 8
```

该程序可以产生最优解决方案吗，即可以找到装入物体的最小数目的容器吗？

25.22 （最大物体优先的装箱）采用一种新的贪婪算法重写前面的程序，将具有最大重量的物体放置在它首先适应的箱中。给出一个例子，演示该程序不能产生最优解决方案。

25.23 （最优装箱）重写前面的程序，使得它可以找到最优解决方案，可以使用最小数目的容器来装箱所有的物体。下面是程序的一个运行示例：

```
Enter the number of objects: 6 Enter
Enter the weights of the objects: 7 5 2 3 5 8 Enter
Container 1 contains objects with weight 7 3
Container 2 contains objects with weight 5 5
Container 3 contains objects with weight 2 8
The optimal number of bins is 3
```

程序的时间复杂度为多少？