That's enough to get our `TreeMap` class up and running. It's still missing an iterator that allows us to access the items in order (to print out a membership list, for example). We leave it as an exercise to add the additional functionality.

7.7 Chapter Summary

We have covered some basic algorithms and data structures for implementing trees. Here is a quick rundown of important highlights:

- A tree is a non-linear container class for storing hierarchical data or for organizing linear data so it can be accessed efficiently.
- Trees are commonly stored using a linked structure but can also be stored as an array.
- Many tree applications use binary trees, which means that each node has zero, one, or two children, but it is also possible to implement trees with an arbitrary number of children.
- The binary search tree property is that for every node, the value of each node in its left subtree is less than or equal to the node's value and the value of each node in its right subtree is greater than the node's value.
- A binary search tree can support a $\Theta(\log n)$ implementation of the search, insertion, and deletion operations while maintaining the binary search tree property.
- Tree algorithms are often written using recursion since the tree itself is a recursive data structure.
- The three common binary tree traversal orders are: preorder, in-order, and postorder. An in-order traversal of a binary search tree produces the items in sorted order.

7.8 Exercises

True/False Questions

1. Every node in a tree has at most two children.

2. The depth of a tree node is the number of nodes between it and the root of the tree.
3. A tree has exactly one root node.
4. A complete binary tree is necessarily a full binary tree.
5. A full binary tree is necessarily a complete binary tree.
6. An in-order traversal of any binary tree produces the items in sorted order.
7. A postorder traversal of an expression tree yields the postfix (reverse Polish) form of an expression.
8. The worst-case search time for a binary search tree is $\Theta(n)$.
9. Every subtree of a binary search tree is also a binary search tree.
10. Since binary trees are non-linear, they cannot be easily implemented using an array.

Multiple Choice Questions

1. A tree is a natural representation of

   a) arbitrarily interconnected data.
   b) linear data.
   c) hierarchical data.
   d) sappy data.

2. Which of the following is not necessarily true of a non-empty tree?

   a) it has height of at least 0
   b) it has at least one leaf
   c) it has at least one root
   d) all of the above are true of a non-empty tree

3. In an expression tree, non-leaf nodes represent:

   a) operands.
   b) operators.
   c) parentheses.
   d) tokens.

4. In what order should an expression tree be traversed to evaluate the expression?

   a) preorder
   b) in-order
   c) postorder
   d) precedence order

5. Which of these design patterns allows clients to traverse a data structure without knowing its internal structure?

   a) visitor pattern
   b) iterator pattern
   c) both a and b
   d) none of the above

6. Which of the following orders will produce a binary search tree with the best search times?

   a) inserting the items in a random order
   b) inserting the items in order
   c) inserting the items in reverse order
   d) all will result in the same search times

7. What is the running time of the recursive tree traversals?

   a) $\Theta(1)$ b) $\Theta(\log n)$ c) $\Theta(n)$ d) $\Theta(n \log n)$

8. What is the maximum number of items in a binary tree with a height of 5?

   a) 5 b) 31 c) 32 d) 63

9. What is the minimum height of a tree with 64 nodes?

   a) 5 b) 6 c) 7 d) 32

10. What is the maximum height of a tree with 64 nodes?

    a) 6 b) 7 c) 63 d) 64

Short-Answer Questions

1. What is the drawback of the array/list representation of a general binary tree? What types of binary trees would be particularly well suited to the array/list representation?

2. Consider the binary search tree from the left side of Figure 7.8 (before the 6 is deleted). List the order that the nodes would be visited for each traversal order (preorder, in-order, and postorder).

3. Write an invariant for the `BST` class.

4. Write pre- and postconditions for the `delete` operation of the `BST` class.

5. A tree sort algorithm proceeds by inserting items into a binary search tree and then reading them back out with an in-order traversal. What is the asymptotic running time of sorting n items using a tree sort. Discuss both worst case and expected case results.

6. Consider the mathematical expression $3 + 4 * 5$. Draw two different expression trees whose in-order traversals produce this expression. Evaluate both of your trees using the evaluation algorithm given in section 7.3. Which tree corresponds to the "usual" interpretation of this expression?

7. Using the `TreeNode` class, write an expression that would produce the tree structure shown in Figure 7.5.

8. In the chapter, we saw that a value in a binary search tree can be deleted by replacing the item in its node with its in-order predecessor. As was noted, it would also work to use the in-order successor. Suppose that instead of always doing one or the other we implement a strategy that chooses between these two "on the fly." Suggest a suitable criterion for selecting which one to use and write pseudocode for an algorithm that performs the criterion test.

Programming Exercises

1. Write unit tests for the `BST` class.

2. Write and test a recursive version of the `find` function in the `BST` class.

3. Write `preorder` and `postorder` traversal generators for the `BST` class. For example, to generate a list for a preorder traversal, we could write code like this `list(myBST.preorder())`.

4. Write a `__copy__` method for the `BST` class.

5. Add a `__len__` operation to the `BST` class. Calling `len(myBST)` should return the number of items in `myBST`.