Lab: Trees and Traversals
CSC 207, “Algorithms and Object-Oriented Design”
Department of Computer Science
Grinnell College
November 11, 2019

Setup

Java 9 doesn’t provide a standard `BinarySearchTree` class, so we’ll use Weiss’s implementation as a starting point. You’ll find it in the `BinarySearchTree.java` and `BinaryNode.java` files in the directory

/home/hajiamini/courses/CSC207/code/binarysearchtree/

1. In Eclipse, open a new project and copy in Weiss’s definitions of the `BinarySearchTree` and `BinaryNode` classes.

2. Could `BinaryNode` be implemented as a nested class within `BinarySearchTree`? Justify your answer.

Displaying the Structure and Contents of a Binary Search Tree

Neither Weiss’s implementation of `BinarySearchTree` nor his implementation of `BinaryNode` overrides the `toString` method, so that each class simply inherits that method from `Object`. As a result, their string representations are nearly opaque.

3. Extend Weiss’s implementation of `BinaryNode` with a recursive `toString` method that returns a string containing a left square bracket, the string representation of the left child node, a vertical bar, the string representation of the element, another vertical bar, the string representation of the right child node, and a right square bracket. (When the `left` or `right` field of the `BinaryNode` is null, `toString` should represent it by the string "null" instead of trying to issue a recursive call.)

4. Extend Weiss’s implementation of `BinarySearchTree` with a `toString` method that calls the `toString` method of the root node and returns the result. Write out what the string representation of a `BinarySearchTree<Integer>` containing the autoboxed `Integer` versions of the integers 42, 39, 61, 58, and 54 should be, and then test your program by building that binary search tree and then invoking `toString` and printing out the result.

Analyzing Binary Search Trees

In section 19.3 of the textbook, Weiss introduces an important way to measure the “stringiness” or “bushiness” of a non-empty binary search tree: its internal path length, defined as the sum of the depth of its nodes. The depth of a node in a binary tree is defined as 0 if the node is at the root of the tree and otherwise as 1 plus the depth of its parent node.

For example, the internal path of the bushy tree shown in Figure 19.19(a) on page 703 of the textbook is 34 (the depth of its root node is 0, and there are two nodes of depth 1, four of depth 2, and eight of depth 3). If the same fifteen nodes were arranged in an unbalanced tree, similar to the one shown in Figure 19.19(b) but longer, that tree’s internal path length would be $0 + 1 + 2 + \ldots + 14$, or 105.

5. Extend Weiss’s implementation of `BinarySearchTree` with a public method that computes the tree’s internal path length. (Hint: have the public method invoke a recursive private or protected method, and consider carefully what the parameters of the recursive method should be, i.e., what information it might be useful to pass from one recursive call to another.)

Another measure of the stringiness or bushiness of a (non-empty) binary search tree is its height. If the root node of a binary search tree has two non-empty subtrees its height is 1 plus the
greater of the heights of those subtrees; if it has one non-empty subtree, its height is 1 plus the height of that subtree; if both of its subtrees are empty, its height is 0.

6. Extend Weiss’s implementation of `BinarySearchTree` with a public method `height` that computes the tree’s height.

**Traversals**

Another way to examine the contents of a binary search tree would be to traverse the nodes of the tree, printing each element as it is encountered during the traversal. Combinatorics teaches us that there are six ways of arranging the three fields of a `BinaryNode`:

- `element, left, right` (“preorder”)
- `left, element, right` (“inorder”)
- `left, right, element` (“postorder”)

Each of these corresponds to a different way of traversing the tree of which the given node is the root. For instance, in a preorder traversal, the `element` field of the root node would be printed first, and then successive recursive calls to the traversal procedure would be used to print, first, the elements in the left subtree and then the elements in the right subtree.

7. Choose one of these traversal methods and implement it as an additional public method in the `BinarySearchTree` class.

8. It would be useful to have an `Iterator` for binary search trees that would produce the elements in all of the nodes of the binary search tree in one of the traversal orders described above. How could such an `Iterator` be implemented?

9. If we were working in Scheme, a traversal method might well take a procedure as its argument and apply that procedure to each element in a binary search tree, either collecting the resulting values in some data structure and returning it or performing some side effect on each element as it is encountered. Since Java 8, it has possible to do something rather similar in Java using “functional interfaces” such as the `Function` and `Consumer` interfaces (and the many other interfaces defined in the `java.util.function` package). Study the APIs for the `Consumer` interface and then suggest how to implement a generic `traverser` method that takes a `Consumer` as argument and applies it, for its side effect, to every element in the binary search tree.