Lab: Trees and Traversals
CSC 207, “Algorithms and Object-Oriented Design”
Department of Computer Science
Grinnell College
May 6, 2021

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? What would be the visibility access of the nested BinaryNode class? (You don’t need to put BinaryNode as the nested class inside BinarySearchTree.)

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 static 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.)

Note that the toString method calls itself as long as there is a left node in a given subtree. After that, it displays the element at the root of the subtree. Then, it calls itself on the right subtree.

4. (2 points) Extend Weiss’s implementation of BinarySearchTree with a toString method that calls the toString method, with the root node as the method argument, and returns the result. (the toString method of the BinaryNode class is a helper method.) 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. **(3 points)** Extend Weiss’s implementation of \texttt{BinarySearchTree} with a public method that computes the tree’s internal path length. Have this public method call a helper recursive method that computes the internal path length. 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.

6. Another measure of the stringiness or bushiness of a (non-empty) binary search tree is its \textit{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.

   Extend Weiss’s implementation of \texttt{BinarySearchTree} with a public method \texttt{height} that computes the tree’s height.

   The recursive case of this method needs to check whether the left subtree, right subtree or both are empty before making a recursive call.

## 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. There are three primary ways of arranging the three fields of a \texttt{BinaryNode}:

- \texttt{element, left, right} (“preorder”)
- \texttt{left, element, right} (“inorder”)
- \texttt{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 \texttt{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.

The recursive implementations of the tree traversals are shown in the book.

7. **(3 points)** Implement the postorder traversal as an additional public method \texttt{postOrder} in the \texttt{BinarySearchTree} class. This method needs to be \texttt{iterative} instead of recursive. Use two stacks to implement the postorder traversal. Push the tree root into the first stack. At the beginning of each iteration, pop an element at the top of the first stack and push it into the second stack. Then, push the right child and left child of a rooted subtree into the first stack in some order. After getting out of the iterations, pop the elements from the second stack. This gives the postorder traversal of a binary search tree.

   Test your postorder traversal by defining a method \texttt{displayPostOrder} that calls \texttt{postOrder} method.