Binary Tree Traversals

CS 331 — Fall, 2010

Revision 1.0

Assigned 2010-10-18
Due 2010-11-01

1 Objectives

In this lab you will create a binary search tree and write some traversal iterators for it.

- Be able to write iterators for DFS, BFS, Frontier, Preorder, Postorder, and Inorder.

2 Binary Search Trees

The tree you implement will be similar to the BST from last time. This tree will only accept `int` (don't use generics!) and will only be a container, not a dictionary. You need to implement `void add(int)`, but not `boolean find(int)`, or `void delete(int)`.

3 Given Files

A directory has been created in your repository called `traversal-lab`, in it is the file `Iterator.java`. This file gives you the interface for the iterators. No other files are given, other than the Makefile.

You will need to create two files at a minimum: `Tree.java`, which will contain your `Tree` class, and `TestTree.java`, which will contain your test cases.

You will very likely want to have other files, such as `List.java`, etc.; feel free to add them as you find necessary.

4 Your Work

You will need to implement at least the following methods. You may add more if you feel they are necessary. If you do so, do not make your test cases depend on them, since they are not part of the spec.

constructor This just creates an empty tree.

void add(int i) Insert an element into the tree. There will be only one place in the tree where it should be inserted. This should run in $\mathcal{O}(\lg n)$ time. If the integer already exists in the tree, simply return.

Iterator mkBFSIterator() You did this in your last lab, so it should be relatively straightforward to port it to this one.

Iterator mkDFSIterator() This iterator traverses in depth-first order.

Iterator mkPreorderIterator() Gives a preorder traversal.

Iterator mkInorderIterator() Gives an inorder traversal.

Iterator mkPostorderIterator() Gives a postorder traversal.

Iterator mkFrontierIterator() Iterates over the leaves of the tree, from left to right.

You do not have to implement size, find, or delete. You may if you want, but do not write tests that depend on them or else the grading script will break and blame you for it.

Hint: some of these are much much easier if you maintain a parent pointer.

5 Tests

As always, you should write good test cases. These will count for more than half the grade. Be very sure that you use the correct method names!

You should create several different trees to test these iterators. Some traversal patterns are tricky, and that will not always be revealed in every tree.