Implementing a Binary Search Tree
Report #08

By
Eduardo Castro

CS 303 Algorithms and Data Structures

October 12, 2014

1. Problem Specification

The goal of this assignment is to implement a binary search tree and for evaluating it we also need to prepare the class for doing the basic operations like adding, searching or deleting.

2. Program Design

The problem requires the creation of four classes: BinarySearchTree, where most of our work will be done/was done, a BinarySearchTreeTest, that's our driver class, a BinaryTree class, that represents the basic structure from a BinaryTree and a BinaryTreeNode, that represents a node from a BinaryTree.

The following steps were required to develop this program:

a) write a pseudocode for the sorting algorithm
b) write the methods inside BinarySearchTree
c) write the BinarySearchTreeTest class
d) use the driver class to display and measure the results

The following methods were defined within the BinarySearchTree:

a) delete(BinaryTree binaryTree, BinaryTreeNode noteToBeDeleted)
    a. deletes a node from the tree
a) treeMinimum(BinaryTree node)
    a. returns the node that corresponds to the extreme left from the tree, also known as the smallest value
b) transplant(BinaryTree binaryTree, BinaryTreeNode u, BinaryTreeNode v)
    a. transplants/change positions of two nodes
c) insert(BinaryTree binaryTree, BinaryTreeNode newNode)
    a. inserts a newNode into the binaryTree
d) search(BinaryTreeNode binaryTreeNode, int elementToBeFound)
    a. returns true if the elementToBeFound is inside the tree and false if it isn't
e) inorderTreeWalk(BinaryTreeNode node)
    a. goes along the tree in order and prints each element sorted

The following methods were defined within the BinaryTree and BinaryTreeNode classes:

f) getRoot
    a. get the root of the tree
g) setRoot(BinaryTreeNode root)
    a. sets a new root for a node
h) toRoot()
    a. points the cursor to the root
i) hasLeftChild()
    a. returns true if the node that the cursor is pointed has a right child
j) hasRightChild()
    a. returns true if the node that the cursor is pointed has a left child
k) toLeftChild()

    a. points the cursor the the left child
- l) toRightChild()
    a. points the cursor the the right child
- m) set(T data)
    a. set a new data to a node
- n) height()
    a. returns the height of the tree
- o) toString()
    a. converts to string
- p) getParent()
    a. returns the parent node
- q) setParent(BinaryTreeNode binaryTreeNode)
    a. sets a new parent to the node
- r) getLeft()
    a. returns the left child
- s) getRight()
    a. returns the right child
- t) setRight(BinaryTreeNode binaryTreeNode()
    a. sets a new right child to the node
- u) setLeft(BinaryTreeNode binaryTreeNode()
    a. sets a new left child to the node
- v) height()
    a. returns the height of the tree

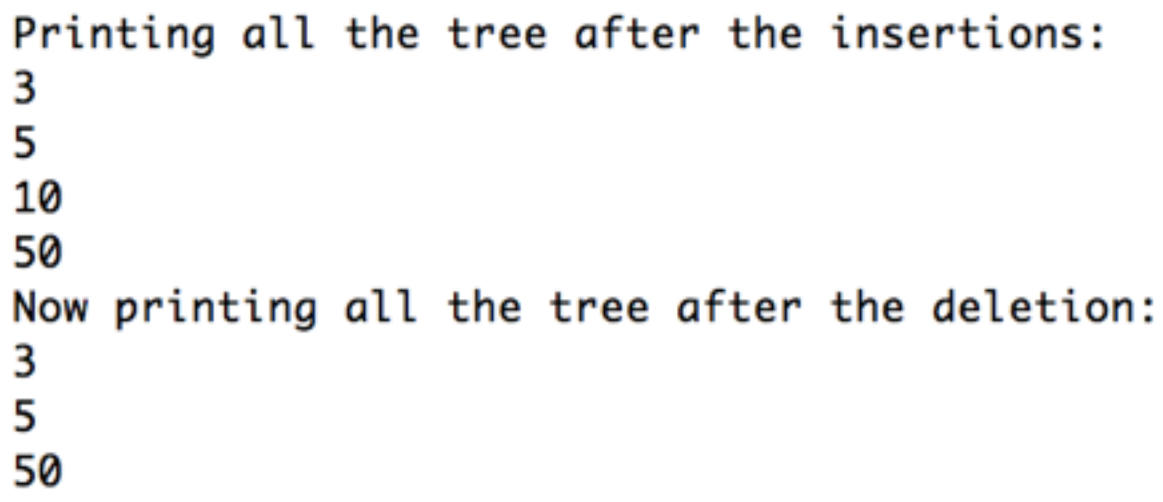
The Scanner class provided by Java was used to read in the necessary values within the provided driver program. The println method of the System.out object was used to display the inputs and results for the provided driver program.

3. Testing Plan

Insert some elements, prints the tree to check if it was inserted in order and than delete a middle element to check if it was well succeed.

4. Test Cases

The tests case:



```
Printing all the tree after the insertions:
3
5
10
50
Now printing all the tree after the deletion:
3
5
50
```

5. Analysis and Conclusions

The implementation of a BinarySearchTree actually it didn't seem to be so different than an implementation of a BinaryTree, there are just some different things. The insertion it is pretty easy and fast but the deleting it seems to be difficult. The search is also very easy and fast.

6. References

The parameters used was from the homework assignment provided in class, I also used as reference for learning how to work with compareTo the Java documentation and another references from an Algorithms course from Princeton.