Kurt Westerman

Advanced Data Structures

Project 2.1

March 20, 2018

**Binary Search Tree**

**Concept**

Binary search trees are a type of recursive data structure comprised of individual nodes which each have some value. The values are relatively quantified between the nodes. Each left child of a parent node must be valued less than its parent and each right child must be valued more than its parent. No duplicate nodes are allowed. If a node is inserted or deleted the tree structure’s validity is maintained after the operation.

**Design**

A main class BST, short for binary search tree, and its sub structure Node will store all the data and perform all the relevant functions of a binary search tree for integer values. Each tree object has a private pointer to a node object called root and an integer s which stores the number of nodes in the tree. Each Node has a private integer for the data and two pointers: one to its left node, and one to its right. The tree object must perform the insert, remove, show, height, size, check public functions and have a working default constructor. These public functions sanitize inputs and often pass the root node by reference into various recursive private helper functions to maintain proper functionality and structure of a binary search tree.

**Testing**

After running the code 20 times it always completed accounting for many test cases including inserting an already existing value, deleting nodes with any combination of children, and deleting nodes that don’t exist. Here is one test outputs of the code bellow:

---------------------------------

Tree: size=0 height=0

The tree is empty

---------------------------------

Tree structure valid

---------------------------------

A seed has been planted.

---------------------------------

Tree: size=1 height=1

pre-order: 34

post-order: 34

34

/ /

---------------------------------

41 inserted

---------------------------------

Tree: size=2 height=2

pre-order: 34 41

post-order: 41 34

34

/ 41

/ /

---------------------------------

deleting 41

41 deleted

---------------------------------

Tree: size=1 height=1

pre-order: 34

post-order: 34

34

/ /

---------------------------------

69 inserted

---------------------------------

Tree: size=2 height=2

pre-order: 34 69

post-order: 69 34

34

/ 69

/ /

---------------------------------

87 inserted

---------------------------------

Tree: size=3 height=3

pre-order: 34 69 87

post-order: 87 69 34

34

/ 69

/ 87

/ /

---------------------------------

deleting 87

87 deleted

---------------------------------

Tree: size=2 height=2

pre-order: 34 69

post-order: 69 34

34

/ 69

/ /

---------------------------------

85 inserted

---------------------------------

Tree: size=3 height=3

pre-order: 34 69 85

post-order: 85 69 34

34

/ 69

/ 85

/ /

---------------------------------

41 inserted

---------------------------------

Tree: size=4 height=3

pre-order: 34 69 41 85

post-order: 41 85 69 34

34

/ 69

41 85

/ / / /

---------------------------------

25 inserted

---------------------------------

Tree: size=5 height=3

pre-order: 34 25 69 41 85

post-order: 25 41 85 69 34

34

25 69

/ / 41 85

/ / / /

---------------------------------

48 inserted

---------------------------------

Tree: size=6 height=4

pre-order: 34 25 69 41 48 85

post-order: 25 48 41 85 69 34

34

25 69

/ / 41 85

/ 48 / /

/ /

---------------------------------

98 inserted

---------------------------------

Tree: size=7 height=4

pre-order: 34 25 69 41 48 85 98

post-order: 25 48 41 98 85 69 34

34

25 69

/ / 41 85

/ 48 / 98

/ / / /

---------------------------------

49 inserted

---------------------------------

Tree: size=8 height=5

pre-order: 34 25 69 41 48 49 85 98

post-order: 25 49 48 41 98 85 69 34

34

25 69

/ / 41 85

/ 48 / 98

/ 49 / /

/ /

---------------------------------

Tree structure valid

10 inserts with random deletes successful.

**Conclusion**

The BST class represents a true binary search tree for all intents and purposes of integers. Implementing this data structure in C++ has really solidified my understanding of the distinction between public and private functions of objects and reinforced my understanding of passing objects by reference vs value. This has been a great exercise in object-oriented programming. The next project should test my ability of polymorphism through inheritance.