CS 2124 Data Structures

Programming Project 3

In this project, we will be implementing the basic functionality of a Binary Search Tree, and then we will use the Binary Search Tree to maintain a set of accounts and orders associated with the accounts (e.g., as we might do if we were working for Amazon).

**Files provided in the project:**

1. BinarySearchTree.h. This file contains the structs needed for the BinarySearchTree as well as the prototypes for several functions that you will need to implement. In particular you need to implement the following functions in the file BinarySearchTree.c:
   1. **BinarySearchTree newBinarySearchTree()** which should allocate the memory for a new binary search tree, initialize the variables, and return the address.
   2. **void freeBinarySearchTree(BinarySearchTree tree)** which should free the tree (including any nodes currently in the tree).
   3. **NodeT \*allocateNode(Element value)** which should allocate memory for a new node, store “value” inside this node, and return the address of the node.
   4. **NodeT \*search(BinarySearchTree tree, int searchValue)** which should **recursively** search the subtree rooted at p for a node containing a key value equal to searchValue (can use a separate function that takes a NodeT\* as a paramter for the recusion). If such a node exists, return a pointer to the node. If no such node exists, return NULL.
   5. **int insert(BinarySearchTree tree, Element value)** Create a node to store the given Element and add it as a leaf in the BinarySearchTree. Do not worry about balancing the tree for this project. Return true if the insert worked successfully, and return false if the node already existed in the tree. I recommend writing a secondary function to recursively search for where to insert the node as discussed in class.
   6. **void printInOrder(BinarySearchTree tree)** Using an in-order traversal, print one node on a line in the following format: “accountNumber accountBalance”. I recommend writing a secondary function to recursively traverse the nodes.
   7. **void printPreOrder(BinarySearchTree tree)** Using a pre-order traversal, print one node on a line in the following format: “accountNumber accountBalance”. I recommend writing a secondary function to recursively traverse the nodes.

2) smallInput.txt. Each line of the file contains a command. The first word on the line will tell you what type of command it is. The options will be CREATE, SALE, or PRINT. If the command is CREATE, it will be followed by the account number. You should then insert a new node into your BinarySearchTree that contains this account number and an accountBalance that is initially 0. If the command is SALE, then there will be two more numbers. The first will be the account number that the sale is associated with, and the second number will be the amount of money that this account owes us for that sale. Search the BinarySearchTree for the node containing this account number, and add the sale amount to the accountBalance for that node. If the command is PRINT, then the next term will either be INORDER or PREORDER. You should simply call the corresponding print function.

3) largeInput.txt. Similar to smallInput.txt, but contains 20,000 accounts and 100,000 sales. Use the previous input to test your code, then run on this file to see how it handles large inputs.

3) abc123p3.c. Rename this file to your abc123. This file is mostly empty right now. It contains the main() function that you should complete. In main, you should create a new BinarySearchTree, read lines from whatever input file you are working with, and call the correct functions to execute the commands.

5) Makefile. Update the makefile to reflect your abc123. Compile using **make**. Execute the program using **./p3** (the program should open the input file in main(). Make sure you are opening the largeInput.txt file when you submit).

**Extra Credit:**

2 different tasks, and you can do one or both to receive extra credit. Task 2 is considerably more work than Task 1 and will be weighted heavier accordingly.

1. Implement a singly-linked list to perform the same task on the largeInput.txt file. Measure the running time using the system clock and compute the difference between the running time using the tree versus using the linked list.
2. Implement the AVL tree as discussed in class, and use the system clock to compute the difference in the running time between the AVL tree and the Binary Search Tree (difference may not be as large since the input file was generated randomly).

**Submitting**

Zip up your project folder in a file name abc123Project3.zip (with your own abc123) and submit on the dropbox on Blackboard by the due date.