CptS 122 – Data Structures

Lab 9: Developing a Binary Search Tree (BST) Class and Application in C++

Assigned: Week of March 17, 2025
Due: At the end of the lab session

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement, test, and apply a BST class in C++
- Compare and contrast *value* classes versus *container* classes
- Apply and implement *overloaded* functions and operators
- Discuss the properties of a BST
- Discuss the advantages and disadvantages of a BST

II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Create test cases for a program
- Design, implement and test classes in C++
- Declare and define *constructors*
- Declare and define *destructors*
- Compare and contrast *public* and *private* access specifiers in C++
- Describe what is an *attribute* or data member of a class
- Describe what is a *method* of a class
- Apply and implement *overloaded* functions
- Distinguish between pass-by-*value* and pass-by-*reference*
- Discuss *classes* versus *objects*

III. Overview & Requirements:

This lab, along with your TA, will help you navigate through designing, implementing, and testing a BST class in C++. It will also help you with understanding how to apply a BST object to an application.

Labs are held in a "closed" environment such that you may ask your TA questions. Please use your TAs knowledge to your advantage. You are required to move at the pace set forth by your TA. Have a great time! Labs are a vital part to your education in CptS 122 so work diligently.

Tasks:

NOTE: Parts of this lab are courtesy of Jack Hagemeister.

For this lab you will develop a BST class and use it to solve a sorting problem. You will create the class BSTNode declaration in a single .hpp (called BSTNode.hpp) and the class BST declaration in a single .hpp file (called BST.hpp). Define all of the functionality for class BSTNode in BSTNode .cpp and functionality for class BST in BST.cpp.

Task 1. Defining a class, BSTNode

Start this task by designing the BSTNode class for the BST. For the initial development you should just build the node to hold a std::string as its data. The BSTNode class will consist of a string and left and right pointers. It will initialize the node using it's constructor. You will also overload the stream insertion operator << to output a node. Will you need access and/or modify the contents of the nodes from outside the node? Yes! Then you should implement getters/setters. Note: you will be inserting data into the tree using recursion. How will this impact your getters in the BSTNode class? The data in the node is a std::string, should we pass the newData value into the setter using pass-by-reference? Recall, a std::string is an object type, and hence, if the object is not passed by reference, then a copy of the object is made (std::string copy constructor is invoked). Is this the intent? Probably not!

Task 2. Now create the BST class

Implement a BST class. You are now ready to define the BST class. You should create a data member for a pointer that will be the root of the BST. The pointers should be of type BSTNode. You will also implement the constructor and the destructor (should destroy the tree through postorder traversing of nodes).

Additionally, you need:

`insertNode()` - that adds an item to the BST. Recall the properties of a BST. The values in any left subtree are less than its parent node, and any values in the right subtree are greater than its parent node. Use recursion in your implementation!

`inOrderTraversal()` - that prints the contents of the BST inorder. Use recursion in your implementation!

`preOrderTraversal()` - that prints the contents of the BST preorder. Use recursion in your implementation!

`postOrderTraversal()` - that prints the contents of the BST postorder. Use recursion in your implementation!

`isEmpty()` - that is a Boolean function that indicates that the BST is empty or not.

`destroyTree()` – a private function, which is called from the destructor to delete each node in the the tree by postorder traversal.

You will overload the stream insertion operator << to output a BST in an elegant way.

NOTE: Listed below are the algorithms for the traversals.

In-Order Traversal:

1. Traverse the “left” subtree by recursively calling inOrderTraversal()
2. Access the “data” of the current node
3. Traverse the “right” subtree by recursively calling inOrderTraversal()

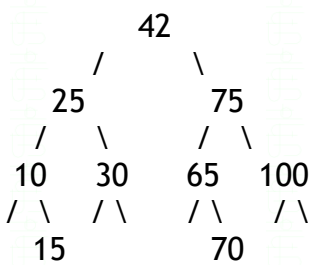
Pre-Order Traversal:

1. Access the “data” of the current node
2. Traverse the “left” subtree by recursively calling preOrderTraversal()
3. Traverse the “right” subtree by recursively calling preOrderTraversal()

Post-Order Traversal:

1. Traverse the “left” subtree by recursively calling postOrderTraversal()
2. Traverse the “right” subtree by recursively calling postOrderTraversal()
3. Access the “data” of the current node

Let’s say we have the following BST:



The in-order traversal would print: 10 15 25 30 42 65 70 75 100
The pre-order traversal would print: 42 25 10 15 30 75 65 70 100
The post-order traversal would print: 15 10 30 25 70 65 100 75 42 // The value in each node is not printed until the values of its children are printed

Task 3. Create a an application to sort strings

Create an application that populates an array with names (last, first) of your favorite people. Read the names from a text file, where each name is placed on a separate line in the form (last, first). Take the array and place all names in the array into a BST object. Traverse the BST inorder (you will need to modify inorder) and place the inorder strings back into the original array. The array of people's names is now sorted. QUESTION: When would it be a good idea to use a BST for sorting items? Do you know of other algorithms and data structures that are more efficient for a sorting task?

IV. Submitting Labs:

- You are not required to submit your lab solutions. You should keep them in a folder that you may continue to access throughout the semester.

V. Grading Guidelines:

- This lab is worth 10 points. Your lab grade is assigned based on completeness and effort. To receive full credit for the lab you must show up on time, work with a team, complete 2/3 of the problems, and continue to work on the problems until the TA has dismissed you.