Project Number: 1

Project Title: CS241 Summer 2018 Project 1

Class: CS 241

Diane Margo

[dmmargo@cpp.edu](mailto:dmmargo@cpp.edu)

July 4, 2018

Written description:

Section 1: Project Description

The project takes data from the user of integers separate by spaces. This program builds a binary search tree using the values entered and makes three traversal lists: pre-order traversal, in-order traversal, post-order traversal. After it makes these lists, it prompts the user to input a command. H for help, or “display this message”, I for “insert a value”, D for “delete a value”, P for “find predecessor”, S for “find successor”, and E for “exit the program.”

Each time the user inserts or deletes a value, the in-order traversal gets printed with the updated list. When find a predecessor or successor, the user must enter the command P or S and follow up with a space to enter a value.

Section 2: Project Specification

The goal of this project is to work with the binary search tree.

Starting with the transverses, we have the inOrder() method that passes the node into the inOrderTranverse() method which checks if the node is null or not. Since the order of inOrder is left-root-right, the program would use recursion to call itself to get the leftChild of the node first, root second (printing), and recursion for the rightChild. The rest of the traverses are the same depending on it’s order. Pre-order is root-left-right; post-order is left-right-root.

Next is the addEntry and removeEntry. To add an entry, method addEntry has a newEntry passed in, an integer value inputted from the user. The input is then tested to see if the input already exists or not. If not, the addingTreeNode() method will have a node and newEntry passed in to make a new node depending on how big the entry is in comparison to the rest of the numbers on the tree. If the newEntry is smaller than the node it is compared to, the tree method will go to the leftChild and compare further through recursion. Right child checks to see if the new value inputted from the user is bigger or not compared to the data at that root node. After the comparison is finished, it returns the node it is located at.

The deleing a node is like adding a node. Once the method can’t find the value to be greater than or less than, it goes to the else statement where it checks to make sure that the value is equal to the value at that node. It finds the successor value and deletes the node.

Search method looks for a specific node. It goes through the tree and looks for the specific node.

The predecessor finds the node that appears before the user given value; the successor is the value after the user input value. The two methods are really like each other, checking to see if the node has that content. After, it determines which direction to go depending on what the user is looking for. If the leftChild or rightChild does not equal null in your while statement, the method will continue to take the information on the left or right. It will then return the value.

In the driver class, we make a new object called the BST tree. We take the data of the initial sequences separated by spaces and put that into an array. We also add each component of that array to the tree as a new entry (addEntry). After that, we play around with the data making it into traverses and displaying the traverses. Commands are asked from the user and put into if statements. The program will stop after the command E is executed by the user.

Section 3: Testing Methodology

To ensure that I wouldn’t run into problems, I looked at the book/slides for reference and looked online to see how the methods were implemented into the code, what kind of return values they should have, and what kind of operations they did. To make sure there would be no errors in what would be returned, all variables were set to a default number or null. Another thing is to check that the signs are right.

First, checking to make sure that each method works one by one is the best way to ensure that one does not get errors. I did the traverses first, which weren’t at all a problem since we did a lot of practice on these during class. That was the start of my recursion for leftChild and rightChild.

Later, I did the Search, addEntry, and removeEntry. These methods had their similarities. The only thing to really check for is so that the node did not equal to null to know that the entry already exists before adding the entry; checking to see if the node equaled null was to check if the node doesn’t exist in the tree. After that, the value would be passed onto the actual method where the program can go through adding the value along with make it’s recursions, using greater than or less than and equal to for deletion.

Successors and Predecessors needed to make sure that the node did not equal null or else there was no node with the value given. If not, the program would continue a find the left or right child depending on what was being searched for.

The driver had its char value for commands and rules of having spaces for inputs from the user. In order to validate the user’s input, there is an else statement if the user did not put any of the command’s instruction inputs.

Section 4: Lesson Learned

From this, I learned to search a binary search tree with recursions and new methods, knowing which entry is greater than or less than the other node’s entry. I also learned recursion that did not end on an infinite loop. Organizing the nodes and where it needed to go was a bit repetitive throughout the project. Methods were similar to each other.

Transverses were dependent on the order and were just switched in lines depending on which transverse was used.

The add and remove method were similar in what to check for and knowing which direction to go. The only difference was that the remove had an else statement for entries that were equal to the value that the user wanted to delete.

Finding the successor and predecessor was also as similar as add and remove were to each other. Finding the two, successor and predecessor, was as simple as checking to make sure the value existed, knowing which direction to go whether a successor or predecessor, and getting the value of the left or right child as long as the child was not null.

Lesson 5: Analysis of Output

The output contains the list of numbers that were given to be put into a tree, the traverses, the commands, the insert, delete, predecessor, successor, and exit. It shows how values exist already or don’t exist yet.

Program Output:

