Assignment: Project Binary Search Tree

Zinedin Bautista

Professor Alex Lai

University of Advancing Technology

Data Structures and Algorithms

July 7, 2020

Assignment: Project Binary Search Tree

Introduction: A binary search tree is a data structure that is based on nodes. In which the left subtree of every node contains the smaller or smallest value while the right subtree of every node contains the larger or largest value in the binary search tree. Also when it comes to deleting a node in a binary search tree there are three cases to take into consideration. Case number one is if the node that is going to be deleted has no child (no other node attached to it from below) then it can simply just be removed. Case number two is if the node that is going to be deleted has a child then the child node must replace the location of the deleted parent node. Then there is case three which is the trickiest one as it involves deleting a node that has two children attached. If the deleted node has two children then it either has to be replaced by the largest value in the left side of the tree in its section, or be replaced by the smallest value in the right side of the tree in its section. The reason for this is to keep the binary search tree balanced as one of the major principles of a binary search tree is to keep it balanced. The binary search tree has many advantages and disadvantages.

Advantages: First off the binary search tree is drastically better than linear searching. And the binary search tree is very efficient and simpler to implement compared to other data structure algorithms. The nodes in the binary search tree are dynamic which means there is no need to worry about giving it a initial/fixed size. Which allows the data structure to grow and shrink as it allocates and deallocates memory. And finally If the binary search tree is balanced it will perform insertions, deletions, and searching at great speed. This is because of the way the binary search tree stores keys within the nodes. Also the binary search tree if constructed

correctly can perform in logarithmic time which is very helpful if you have a large amount of nodes.

Disadvantages: First off the binary search tree is a lot more complicated than linear searching so it's best to now use a binary search tree for a very small number of elements. If a binary search tree is not kept balanced this will lead to major drawbacks in performance and efficiency. If you have an unbalanced binary search tree this would lead to a worst case scenario in terms of Big-O Notation O(n) (n = number of nodes). Also the shape of the binary search tree will depend on the order of insertions, and this will lead to a degenerated binary search which will lead to a worst case scenario, and also would mean that the performance and efficiency of the binary search tree will be massively hindered. And finally depending on the data set you might experience long search times. This could be because if you insert or search a data element in the binary search tree the key of each called node has to be compared with the key of the data element that is going to be either searched or inserted. As you can see a lot of the disadvantages of the binary search tree is all due to it being unbalanced, so if you are able to maintain the balance of the binary search tree then you can mitigate some of the problems with the binary search tree.

Programmer's Guide:

BinarySearchTree.h

Class Node: The class node is a templated class that contains the definition for the node in the binary search tree and is also a data structure for each node in the binary search tree. All of its attributes are placed into public access. All the attributes in the class Node are int key which is an integer value storing the key for each node. The node* left which is pointing and

referencing the left side of the binary search tree, and the node* right which is pointing and referencing the right side of the binary search tree

Class BinarySearchTree: This class is also a class template but this on the other hand contains the definition for the binary search tree as well as all the function prototypes for the binary search tree. All of the attributes are under public except for Node getNode which is uber a private specifier. The node root acts as the beginning of the binary search tree and will point to the first nod in the binary search tree. There is a constructor for the binary search tree where the root will be initialized as null. As mentioned before this class contains all the functions prototypes, and the functions that are going to be used for the program is interNode, preorder, inorder, postorder, deleteNode, findMax, findMin, and finally getNode. The insertNode is a recursive function that will allow the user to input a number into the binary search tree. Secondly you have the preorder recursive function which will display the binary search tree by root, left, right. Thirdly you have an inorder recursive function which will display the binary search tree by left, root, right. Fourthly you have the postorder recursive function which will display the binary search tree by left, right, root. Then you have the deleteNode recursive function which as its name implies will delete a number from the binary search list via the user input. Then you have the findMax and findMin functions which will find the smallest and largest value in the binary search tree. And finally you have the getNode function which will point to created or deleted nodes.

BinarySearchTreeV1.0.h: (this file was converted into a header file due to the templates)

getNode(): This function definition contains a node that will be pointing and initialized to a new node. The new node will be filled in with the key value of the binary search tree, while setting the left and right node as null. Then finally it will return the data of newNode.

insertNode(): This function definition has an if statement that will essentially stop whenever the root node has reached null, in which it will then return the value of getNode. There is a second if statement that will basically traverse the right node if the user value is greater than the key and insert the new node to the right side. Then there is the last if statement which will do the opposite and traverse the left node if the input value is less than the key and then insert the new node to the left side. And finally it will return the root node data.

findMax(): The function definition has an if statement that will run if the root node has reached null in which it will return the root data. It also has a while statement that will basically keep traveling to the right side of the binary search tree until the right node is null, in which it will proclaim the farthest node on the right as the biggest value. Finally it will return the root data.

findMin(): The function definition has an if statement that will run if the root node has reached null in which it will return the root data. If also has a while statement that will keep traveling to the left side of the binary search tree until the left node is null, in which it will proclaim the farthest node on the left as the smallest value. And then finally it will return the root data.

deleteNode(): The function definition for the deleteNode has an if statement that will stop until the root node has reached null in which if the if statement's condition is met it will return the getNode value. The delete function will handle the three cases mentioned before which

was deleting a node with no child, deleting a node with one child, and deleting a node with two children. It has an if else if statement in which the if statement will traverse to the left if the input value is less than the key in which if that condition is met it will delete the node from the left side. The else if statement will do the opposite where it will traverse to the right node and delete the node from the right side if the input value is greater than the key. After that there is an else statement with a if, else if statement nested. Within the nested if, else if statement the if statement will have a condition is which if the left side of the node is null then it will create a template node pointing to the right side, free the root node, and then return the data of the temp node. And within the else if statement it will have an opposite condition in which if the right side of the node is null then it will create a temporary node pointing to the left side, free the root node, and return the data of the temp node. Then outside the nested if,else if statement a temporary node will find the smallest value on the right side of the tree to replace the deleted node. Then finally it will return the root node data.

preorder(): The function definition for preorder will basically display the binary search tree by processing the root node, left node, and right node in that order.

inorder(): The function definition for inorder will basically display the binary search tree by processing the left node, root node, and right node in that order.

postorder(): The function definition for postorder will basically display the binary search tree by processing the left node, right node, and root node in that order.

BinarySearchTreeMain.cpp

The main cpp file starts off by initializing and creating a binary search tree by naming it myBinarySearchTree. There are nine numbers that by default have already been

inserted into the binary search tree. The primary purpose was for testing to check if all the functions worked as intended by comparing the integers via the hard-coded version versus the version using functions. It will start off by displaying the nine integers inserted into the binary search tree by default. And then it will display a menu where the user can interact with the different functions by typing specific numbers that correlate to the functions using a do,while loop with a switch statement, and an integer variable that would collect the user input on what function they wanted to interact with. If the user wants to insert or delete an integer from the binary search tree the program will ask again for more user input. If the user wants to insert an integer into the binary search tree then the program will ask the user to give it a number to insert in which once that's done it will execute the insertNode function. On the other hand if the user wanted to delta an integer from a binary search tree then it will ask the user what number they wanted removed from the binary search tree, in which once that's done it will execute the deleteNode function. Other than that the other functions (findMax, findMin, preorder, inorder, and postorder) will be executed automatically as they do not require more user input.