Garett Nguyen

CS 241

Dr. Tang

Project 1

The program creates a binary search tree that uses recursion as a means of traversal through each node. It includes standard methods in the BST interface, such as a find method, insertion, deletion, finding a nodes predecessor and successor, and also contains methods to traverse the tree through pre order, in order, and post order traversal methods. The tree stores on data of type int, and does not take hold a parent reference in its nodes.

The rest of the program is built on the interactive interface with the user, which prompts the user to input data and press enter after each piece of data. This saves storage space, as it no longer necessitates holding the values into an array and parsed into the tree. After the tree is initialized with the given values, the program is able to run further operations, which are defined in the BST class. They are the insertion, deletion, predecessor, and successor methods. These methods are called by using a switch block, which takes the first value of a key the user inputs.

Most importantly in this program is the functioning of the tree, which took considerable testing to perfect. In a few cases, testing was very straight forward, as the answer should constant if the methods worked correctly. For example, the traversal methods have only one solution per tree, so it was simple to find when these methods worked or did not work. It was also helpful to have these methods as a means of testing the structure of the tree after insertions or deletions, because the accuracy of the pre order, post order, and in order would also determine the correctness of the tree. In most cases, these methods worked fairly well, only producing an extra value after the delete methods, which merely noted that there was a problem with the delete method.

Insertion too was fairly simple. There was only one case in which insertion had to work differently depending on the argument, and that was if the value to be inserted already exists within the tree. To this, I simply called the find method to check if the value exists in the tree before proceeding with insertion. The delete method, however, is much more complex, as there are several cases of deletion. The first step in deletion is to check if the value in question is in the tree, so the find method is called. If it does not exist in the tree, it returns null without deleting anything. If the value exists within the tree, the method traverses recursively from the root to the node in question, and there, depending on the position of the node, the method behaves differently. If the node in question has no children whatsoever, the reference from the parent node is deleted. If the node has one child, left or right, the child is assigned to the parent of the node in question. If there are two children, the predecessor replaces the value, and the predecessor could be either the left child, or the rightmost in the left subtree. In testing this, there must have been at least four instances of deletion, each with different circumstances. Each of the test conditions operate correctly, using the simplest method of deletion, simply creating a reference to the next-next node, as often as possible. However, if necessary, the tree will simply use the predecessor to replace. If the root of the tree is deleted, the predecessor or right child if there is no predecessor, will replace it as the root of the tree.

While a simple program, this project helped me understand how a tree works, especially the recursive nature of tree traversal. It further cements my understanding of how deletions and insertions work, which can be relatively complex. Overall, I see the tree as a very useful data structure that is considerable efficient in comparison to others that I’ve learned thus far. This program has given me the framework to use the binary search tree so that I can use it in the future, and I think that learning from this project, I have a solid foundation of understanding of this data structure.