Haussmann, Benjamin

Diaz – Winter 2018

CS 241-01

Section 1.

Build a binary tree that takes an initial input of integers. Display the Post, Pre, and In order traversals of the tree. Allow the user to add, remove, and find the predecessor and successor of a given entry.

Section 2.

The program is designed to allow the user to input the initial values of the tree. After the entries are entered, the Pre, Post and In order traversals of the tree will be displayed. Once this is established, the user can then add or remove entries as they wish. All entries are restricted to integers. They are also provided options to find the successor or predecessor of a specified entry. If any wrong input is detected, the program will notify the user and allow them to reissue the command. If the user would like to exit, they can press ‘e’. The commands are designed to not be case sensitive. All integer entries should be separated by spaces.

Section 3.

To test the program I designed a tree that was not full. I inputted the tree and tried to make the program throw an exception. I tested basic features and checked for functionality before moving onto extreme niche cases. I first checked the Pre, Post, and In order traversals and made sure they were correct. Once that was established, I then moved onto the predecessor and successor methods. When I believed those were near completion, I started testing those and had a lot of adjustments to make. I realized that once they were working, the extreme cases of finding the predecessor of the first integer in the traversal or the successor of the last number would cause an exception, I added some code to check and take care of these extremes.

Section 4.

This program was well rounded and had many challenges., The book provided a lot of assistance, but ultimately, it just took time to sit down and think everything through. It has been a long time since I’ve had to take in multiple integers from one lines, so that was a refresher. I also gained a very intimate understanding of binary search trees. They seem incredibly useful and fast at large scales. With only a bog O of log n, these trees seem very useful for storing data and radically decreasing search and retrieval times. The way all of the different methods came together was neat. This particular data structure is fairly complex and requires many different methods and helper methods to operate, but it all seems to be worth it for the speed and reliability it provides, not to mention, being node based, it is particularly memory friendly at scale.