Mohammed Awan

Project Report

**Program Code Layout:**

So the way I went about assembling this programs was to first follow the step in the instructions, but I didn’t do it the exact same way. The source code consists of five class, the first two are two different node classes one for the nodes in the linked list for the hash table and that class PersonNode. The other one is the class for the nodes in the BST and it is called TreeNode. You just store the node info using each of these classes. I split them into to different classes to make it easier to call on the different nodes depending on the situation. The next two classes that I made are the LinkedList and BinaryTree classes. In these classes you can do different operations such as add, delete, update, etc. The next class I have is the HashTable class and this just takes each linked list and stores it in an array to use as part of a HashTable. I took the functions of the hash table that were supposed to be in TelephoneBook and put them in HashTable just to organize it a little better. The final class as I just mentioned is the TelephoneBook class and this is the main class that you run and it acts as a main menu of sorts. You can input a cvs file to populate the structures, or you can do any operations. The operations you can do are inserting, retrieving, updating, or deleting a person’s info, you can display the whole phone book, and you can do tests for the runtime of the data structure. You can also write the phone book to a cvs file at the end as well.

**Performance Analysis:**

Just as set up for all of the tests performed, the way I ran the test was to first populate each data structure with 10,000 elements. After that I did the operations 10,000 times as well. and then compared the runtimes.

So for the first test I tested the performance of a binary search tree hash table with a a size of 7. The results showed that besides being relatively equal for the Insert operation’s performance, the binary search tree was faster in every other operation.

The next set of tests that I ran were testing the performance of the different hash table sizes. So that there is no confusion, on the x- axis the number inside the parentheses is the load factor and the number on the out side is the table size. I’m assuming that the number of nodes is a constant 20,000 which it isn’t exactly, but just for ease of testing I left it. So as seen above when the load factor is extremely high then it takes a significantly longer time to complete the operations and once the load factor is less than 1 then the operations are much smaller. Also after the load factor is less than 1 the time efficiency doesn’t seem to change. Also this isn’t show in the time comparison of the performance, but if you make a table that is excessively large than it will take up space unnecessarily. So the table size I would choose would probably be around 25,00 because it would keep chaining down to a minimum and also allow for many more people to be inserted into the data structure.