Arturo Villalobos UIN: 827008236 email: rturovilla@tamu.edu

Dr. Teresa leyk

CSCE 221

29 March 2020

PA-3 Report

SOURCES USED (WEB)
https://www.youtube.com/watch?v=13TrhiKLZg8
https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/
https://www.geeksforgeeks.org/level-order-tree-traversal/
https://www.geeksforgeeks.org/print-level-order-traversal-line-line/

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.

On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Signature: Arturo Villalobos Date: March 29, 2020

1.) The purpose of this assignment was to explore the behavior of Binary search trees while also learning its relation to recursion and this can be seen in the way functions are implemented. Like in past assignments, declarations and definitions are separated into header and .cpp files, but unlike past assignments, many of the functions that are used in the main driver file needed to be paired with a helper function due to an emphasis on security and the recursive nature of practically all functions. In total there are 7 helper functions that I wrote in order to perform the required tasks. In the main driver file code all the constructors and basic functions are used and demonstrated in the first 60 lines but past that there are three for-loops responsible for reading all the datafiles provided in the starter code zip file and making trees out of the data, outputting inorder/level-by-level versions of the same tree, and outputting the total number of nodes in each tree. I want to point out that in each of the for-loops (one for all three types of trees being tested) the level-by-level function is commented out because otherwise the program takes very long to finish, but can very easily be un-commented by the grader. I had to delete the original starter code and implement the same functions in my own way, for some reason or another I could not seem to get the original code to work. Finally, there is a section of code I wrote in order to write to files with the number of nodes and the average search cost to make constructing the tables and plots easier for me. Feel free to uncomment this section too, but it takes a significant amount of time for the program to run this way.

2.) The data structure utilized in this assignment is very similar to linked list data structures we have used in the past, first making a struct of a node with its own attributes , then constructing a class made up of those nodes. In this program however, each node has two

pointers to other nodes and each BSTree has a special node called the root that can be used to find every other node in the tree.

3.)

a.) My solution to finding the individual search times for every node starts with one of my helper functions called depthfindr. This function finds the maximum depth of a given tree recursively with a big-O of O(n) because in this process the function will go through every node in the tree once. With this in mind my my update_search_times function calls a helper function called ust2 that takes in the root of a given tree, and sets the root's search time attribute to 1 + (depthfndr(this->root) - depthfndr(root)). Depthfndr is really a misnomer because what i'm actually doing is subtracting the height of the current node from the root's height. Ust2 will go through every node once , so in conjunction with depthfndr , updating search times for a tree has a big-O of $O(n^2)$

b.) Finding the average search cost is a simpler process. Due to the fact that i update search times everytime I insert a node , when i call the average search time function all i have to do is traverse the tree and sum up the searchtime attributes of every node and divide by the size of the tree which is also updated every time the insert function is called, making the function have a big-O of O(n).

4.) Assuming the given formulas are true the average search cost in terms of n for a linear-organized BST is O(((n+1)/2)) . The average search cost for a perfect binary search tree in terms of n is $O(\log_2(n+1))$.

5.)

a.) Perfect Tree

i.) The experimental and theoretical average search costs seem to be very close to each other save for some constants

b.) Linear Tree

i.) The experimental and theoretical average search times are again nearly identical except for a few constants

c.) Random Tree

i.) The trees that had randomized input most closely resemble the average search cost of a perfect tree which makes sense given what we know about quicksort best, worst and average cases.

[1] The equation for the first plot is blurry and small but it reads as y = 1.2538ln(x) + 0.2091