Lab 4 – B-Trees

In this Lab we work with B-Trees. We have to solve nine different problems that require certain trees and return specific details. This lab had extremely similar questions to the exercise paper on B-Trees that we had received in class, so that helped to complete this lab.

Question 1: To find the height of the tree, I just went to any leaf, because the leaves will always be on the same level. So basically, I counted down until I hit a leaf. Running time is O(n)

Question 2: To create a sorted list, I decided to start from the leaves, and just read as it traversed to the end of each node. Tricky thing was to read the nodes that were not part of the leaf as it traversed, to make sure everything was in order. Since the nodes of the item was always one less of its children nodes, I created an if statement that printed the node as it passed by. Trying to have it automatically read after it processes a child node will result in an error, as reading that last child node will always be one more than the length of the parent. Also, I learned that you can simply append to an array while I was working on this, so that’s something really convenient when using python. Running time is O(n^2)

Questions 3 and 4: To Find the min/max, I traversed the left/right side of the tree until I reached the depth, in which I then returned the left/right most of that node. For traversing left I always went to child[0] and to reach the right I used child[-1]. Code checked to see if was the correct depth, then it checked whether or not it was on a leaf or not to decide if it should loop again. Running time was O(n)

Question 5: To count the node at a depth, I traversed every node up until the desired depth, and if it was the desired depth I would add on to a sum. Each node at the desired depth added 1, while every node above that depth would just traverse downwards. Running time is O(n^2)

Question 6: Similar to question 5, but instead of adding a 1 when it reached the node, I had it read the node and print each value in that node. Running time is O(n^2)

Question 7: For this one, I traversed every node again, but each time I looked at a node, I checked to see if it was the same as the max, for which I used .max\_items. In using this, the method should work for the max of any tree. The max for my code was 5, in which there are no max nodes. The running time is O(n^2).

Question 8: Extremely similar to question 7, but I changed the code so that it would only check to see if the node was full, if said node was a leaf. That’s it. Running time is of O(n^2)

Question 9: Now this was a particularly tricky one. You have to find an item within the tree, and return the depth of said item. I faced two problems when trying to get this to work. 1. How do I properly track the depths until I find said item? 2. How do I return -1 if I don’t find the item within the tree? The problem I faced with number 1, is that the for statement that I had that traversed the tree, did not stop even if I found the selected item, so I ended up creating a double negative that canceled out any additions to the depth I was trying to find, and if I found that specific key, it would ignore that double negative rule. However, I could not fix my second problem, which was returning -1 if I could not find k. my problem being is I cannot distinguish when I reach the last leaf along with whether I found k or not, so my program instead returns depth 1 if the key is not found – WHICH IS TROUBLING, as all the values of depth 1 that are found are grouped up with it. The running time for what I do have is O(n^2)

(I am not uploading pictures of my code as it would make the paper much larger than it needs to be – each question is properly labeled in the source code.)

Overall this helped me understand B-trees better. I’ve learned how to properly traverse them and find different details. However, as far as number nine goes, I think the issue I’m having in solving it is more based off of recursion, rather than the B-tree itself. So, my main concern right now is how to properly coordinate information as I go through recursive statements. One more interesting thing I learned from working with B-trees is that using [-1] reaches the end of an array, and using [-2] reaches the 2nd to last, and so on.

Appendix: <https://github.com/Penguinhedgehog/CS2302-PatrickBrannan/blob/master/CS2302/Lab%204/Lab%204.py>

***I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the expirements, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class. – Patrick Brannan***