BTree Lab Reflection
Ryan Blocker
April 27th 2024
Craig Partridge
CS320

Normally, I would paste my code below, but the file is so big that it would take multiple pages. However, I have uploaded it to Zybooks so you can view it there.

Approach: This was by far the hardest lab. For most of it, I thought I was doing it wrong because there were so many methods that needed to be implemented, especially for deletion. I thought I had missed a library or anything that could implement this functionality for me. This code is pretty ugly and it got super confusing. However, this is the approach I settled on: starting with the insert method, I handle the normal edge cases and then jump into the main insert method. This method checks whether I am less than the length of the node keys, and if it is, it increases by one. If it's not, it inserts the key at that position. If the length of the keys is greater than the max size, it splits the node. If it's not a child, it inserts the value there.

Moving on to the split method, where I split the tree node in half. I take a node as input and create two new nodes, left and right, by splitting the keys of the original node. Then, I update the parent node, if it exists, by inserting the new keys and nodes into the correct positions. If the parent node becomes full, then the split method is recursively called on the parent node. After all of this is done, the original node is removed from the tree by setting its parent to none.

The contains method contains a search function that recursively searches for the value given to the contains method and traverses the tree until it's found. If it is found, it returns true.

Finally, the delete method was the hardest of all. First, I checked for edge cases and then I searched for the value in the given node and its children. I started by initializing a variable, eye, to zero. Then, I iterated over the keys in the current node until I found a key that was greater than or equal to the given value. If the key matches the given value, it checks if it has any children at all. If it doesn't, it removes the key from the node and handles the underflow. I'm pretty sure this is incorrect because I'm only getting 30 out of 45 on this assignment, and I could use some guidance. I keep failing the second insertion and deletion test, as well as the large tree test. I think there's something wrong with how it's being removed.

Modularity: As far as modularity goes, this code is probably the worst I've done. Everything isn't in a single file, and it's pretty hard to follow, to be honest. I wish we could submit more than one file, like a file with multiple different files, to make things cleaner. If I were to do this, I would have made a utility class and a node class, and kept the given main.py BTree() class in its own file.

Big O Complexity: As far as I believe, my Big O Time complexity is O(log n). In my insertion operation, it begins at the root and proceeds downward. At each level, it may or may not split a node. In the worst case scenario, it's only traversing the height of the tree, which takes at most O(max_size). So, insertion is O(log n). My deletion algorithm is unfinished, but so far it's similar to insertion and has the same time complexity. So overall, my algorithm is O(log n).