Michelle Zhu
Data Structures

November 10, 2018

This program completes the insert operations of a sequence of integers for both a binary search tree (BST), an AVL tree, and a splay tree from a user-inputted file. The binary search tree inserts elements in such a way that all the keys in the node's left subtree are less than the node's key and all the keys in the node's right subtree are greater than the node's key. The AVL inserts in a similar fashion, but it uses balance factors (-1, 0, 1) to maintain the balance of the tree; when the right and left subtree differ in height by more than 1 (or has a balance factor greater 1 or less than -1), the program rebalances so that the balance factors are restored to [-1,1]. The splay tree similarly uses rotations to make the most recently inserted node the root, but does not keep track of balance factors.

The program also executes removals for another sequence of integers from the file. In the binary search tree, the program recursively searches the right or left subtree of the root depending on if the key is greater or less than the key at the root and deletes the node when the key is found. The AVL tree similarly removes keys, but implements single and double rotations if the tree becomes unbalanced after a deletion of a node. For the splay tree, when a node is deleted, the maximum value in the node's left subtree becomes the new root of the tree. The program outputs the states of each tree after insertion using a print function, as well as the states of the two trees after each deletion. For each operation, the number of comparisons is counted, and the program outputs which tree accomplishes the operations more efficiently based on the number of comparisons.

For the binary search tree, its insert and delete operations have a worst case complexity of O(n), where the desired item is at a leaf node and height equals the number of nodes. For the AVL tree, because of its balancing mechanism, its insert and delete operations have a worst case complexity of O(log n). When a tree is unbalanced, or more similar to a linked list (e.g. a chain of only right children), it is more efficient to have an AVL implementation so that subsequent access of certain elements are faster at O(log n). While AVL may take up more memory in order to account for the balance factors, it only changes the complexity by a constant factor; time complexities of AVL operations are still faster than those of the BST. Splay trees offer amortized O(log n) operations; a sequence of m operations takes O(m log n) time.

When the file p2_test2.txt is inputted, the binary search tree makes 35727 comparisons, while the AVL tree makes 2812 comparisons. The splay tree makes 1580 comparisons. The AVL tree was determined to be more efficient than the BST based on number of comparisons.

Compile instructions:

$ make
$ ./p2 fileName