Catherine Kuntoro
CS 146
Professor Aniket Chandak
28 April 2020

Homework 4 Extra Credit: AVL and Splay Tree

An AVL tree is another type of binary search tree that can balance itself. For every node of an AVL tree, the height of the node's left and right subtrees can only differ by at most one. If the height of either the left or right subtree differs for more than one, then the AVL tree will rebalance itself through means of either left or right rotations. An AVL tree has a worst-case O(n) of space complexity, and for its search, insert, and delete methods, each will take O(log n) time because an AVL tree will definitely be a balanced tree, hence growing only in log(n) time, which is the time a tree heigh grows.

AVL tree structure has the advantage of having a faster search time since it will be more balanced in comparison to Red-Black Trees since the left and subtree's height can only differ by strictly one. However, since the height has to be maintained in AVL trees, AVL trees will perform more rotations than Red-Black trees, which could slow down insertion() and deletion(). Therefore, for operations such as insertion() and deletion(), AVL trees would be slower than Red-Black Trees. Therefore, AVL trees would be a better data structure if the user wants to mainly search for a node or perform lots of lookups, rather than doing lots of insertion or deletion.

The four possibilities of rotation include a "left-left" case, where the left subtree (call it X) of the root node has a height that is greater than the root's right subtree. Then, X's left subtree (call it Y) also has a height that is greater than X's right subtree. In this case, right rotate the root node to allow X to be the new root node. The previous root node will be X's right subtree. If X previously had any right subtree, it will be the previous root node's new left node
The second possibility is the "left-right case", where the height of the root node's left subtree (call it X) is greater than the root node's right subtree by more than one. Then X's right subtree's height (call it Y) is also greater than X's left subtree's height by more than one. In this case, first, left-rotate Y so that X would be its left subtree. Any left subtree that Y has will be X's new right subtree. Then, right Y so that Y will be the new root, and the old root will be Y's right subtree. Any right subtree Y has will be the root node's new left subtree.
The third possibility is the "right-left" case, where the height of the root node's right subtree (call it X) is greater than the root node's left subtree by more than one. Then, X's left subtree (call it Y) is also greater than X's right subtree's height by more than one. In this case, first, right rotate Y so that X would be Y's right subtree. Any right subtree Y has will be X's left subtree. Then,

left-rotate Y with the root node so Y will be the root node. The old root node will be Y's left subtree. Any left subtree Y has will be the old root node's new right subtree.
The last possibility is the "right-right" case, where the height of the root node's right subtree (call it X) is greater than the root node's left subtree by more than one. Then, X's right subtree (call it Y) has a height that is greater than X's left subtree by more than one. So, left rotate the root node so that it would be X's new left subtree. X thus would be the new root node. Any left subtree X has would be the old root node's new right subtree.

A splay tree is another self-balancing binary search tree, but recently accessed elements are made so that it could be accessed quickly again. A splay tree has a worst-case space complexity of O(n), and for the search(), insert(), and delete() operation, the worst-case time are O(log n) for each operation.

An advantage splay tree has is that nodes that are accessed frequently will be at or near the top of the tree, thus lookups for such node can be done very quickly. However, a big disadvantage of splay trees is that the height can be linear, which could happen by accessing all n elements inside the splay tree in increasing order. However, the splay tree's unique feature if useful if the user wants to access a certain node often, since the search operation() does not need to be redone for the entire tree, but only the upper areas of the tree. In real life, splay trees are used for a network router.

Splay trees can rotate just like Red-Black trees or AVL trees. Then, splay tree has a "zip" step. This step will occur when a node (call it X) that is the child subtree of the root node needs to be moved up to be quickly accessed. Therefore, the root node will either move to the left or right subtree of X, and X will be the new node. Accessing X would be faster since X would be the first node returned. Splay tree's "zig-zig step" occurs when the node to be moved up (call it X) is further down the tree. In that case, keep on pushing X upwards by switching its parent values by X's position. A "zig-zag" step works like how AVL's "right-left" and "left-right" rotation works.