CMSC 123: Data Structures

1st Semester AY 2019-2020

Prepared by: CE Poserio

[In Lab] Exercise 05: AVL::Delete Implementation

AVL Deletion

Deletion in an AVL tree is fairly similar with how insertion is implemented - delete the target node using BST's delete method and re-balance the tree.

The two major steps in deleting a new node, n, in an AVL, A:

1. Delete n using the BST deletion method.
2. Re-balance the tree as follows:
    a. Let m be the node which replaced n.
        i. if n had no child, m is the parent of n;
        ii. if n had one child, m is the only child of n; otherwise,
        iii. m is the parent of the predecessor/successor of n (in this case, the node n had two children, and its data items were replaced by its successor or predecessor).
    b. Find the **critical node**, which is the unbalanced ancestor of m. Let this be node c.
    c. Find the **pivot node**, which is the *heavier child* (*i.e.* child with larger height) of m. Let this be node b.
    d. Find node a, the heavier child of b (grandchild of node c).
    e. Perform appropriate rotations based on the configuration of nodes a, b, and c as described in the handout for handling imbalances during insertion.
3. Set m as its parent (go up one level) and repeat from step `2.b`. That is, check for imbalances until the root node.

Tasks

Implement and test the following functions

1. `AVL_NODE_PTR avlDelete(AVL_PTR, int);` - a function that deletes a node identified by the given `integer key` from an AVL tree; return the deleted node, if successful; otherwise, return `NULL`;

Since, `BST` implementation is reused, make sure that you have a complete header file `BST.h` and a fully working implementation file `BST.c`.

Submission

Submit your `AVL.c` to Google Classroom.

Questions?

If you have any questions, approach your lab instructor.