DS2030 Data Structures and Algorithms for Data Science
Lab 5 (Take Home) Due on October 2 11.59pm

Instructions

- You are to use Python as the programming language. Use may use Visual Studio Code (or any other editor you are comfortable with) as the IDE.
- You have to work individually for this lab.
- You are not allowed to share code with your classmates nor allowed to use code from the internet. You are encouraged engage in high level discussions with your classmates; however ensure to include their names in the report/code documentation. If you refer to any source on the Internet, include the corresponding citation in the report/code documentation. If we find that you have copied code from your classmate or from the Internet, you will get a straight fail grade in the course.
- The submission must be a zip file with the following naming convention - rollnumber.zip. The Python files should be contained in a folder named after the question number.
- Include appropriate comments to document the code. Include a `read me` file containing the instructions on for executing the code. The code should run on institute linux machines.
- Upload your submission to moodle by the due date and time. Do not email the submission to the instructor or the TA.

This lab will improve your understanding of AVL Trees, Single and Double Rotations to maintain the height balance, Insertion and Deletion operations in an AVL Tree.

AVL Tree Implementation

1 Introduction

An AVL tree is a self-balancing binary search tree (BST) where the difference between heights of left and right subtrees for every node is at most one. This height difference is known as the *balance factor* and can be -1, 0, or 1 for a balanced AVL tree.

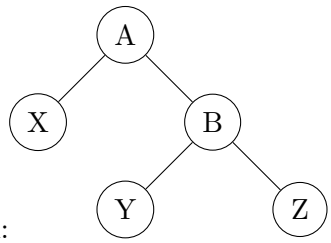
2 Rotations

To maintain the AVL property during insertions and deletions, the tree may need to be rebalanced using rotations. There are four types of rotations:

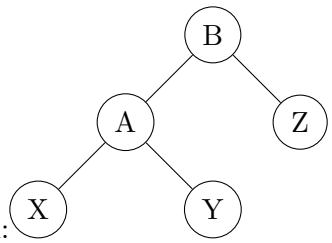
2.1 Left Rotation (LL Rotation)

A left rotation is applied when a node becomes unbalanced due to a right-heavy subtree. If node A has a right child B, then the left rotation around A makes B the new root of this subtree, and A becomes the left child of B.

Initial:



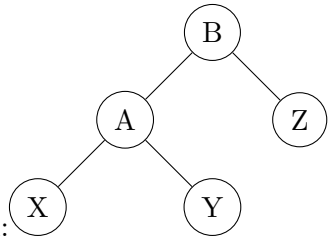
After LL Rotation:



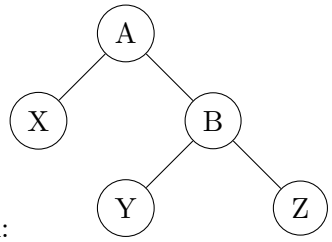
2.2 Right Rotation (RR Rotation)

A right rotation is applied when a node becomes unbalanced due to a left-heavy subtree. If node B has a left child A, then the right rotation around B makes A the new root of this subtree, and B becomes the right child of A.

Initial:



After RR Rotation:



2.3 Left-Right Rotation (LR Rotation)

An LR rotation is required when the left subtree is heavy but has a right-heavy child. First, a left rotation is performed on the left child, followed by a right rotation on the unbalanced node.

2.4 Right-Left Rotation (RL Rotation)

An RL rotation is needed when the right subtree is heavy but has a left-heavy child. First, a right rotation is performed on the right child, followed by a left rotation on the unbalanced node.

3 Insertion in AVL Trees

Insertion in an AVL tree is similar to insertion in a regular BST. After insertion, the balance factor of each node is checked, and rotations are applied if needed to maintain the AVL property.

3.1 Steps for Insertion

1. Insert the new key in the appropriate position following BST rules.
2. Traverse up from the inserted node to the root, updating balance factors.
3. If a node becomes unbalanced, apply the necessary rotation(s) to restore balance.

4 Deletion in AVL Trees

Deletion in an AVL tree follows the BST deletion method but may lead to an imbalance. After deletion, the balance factor of each node is checked, and rotations are applied to maintain the AVL property.

4.1 Steps for Deletion

1. Remove the node using the standard BST deletion procedure.
2. Traverse up from the deleted node's parent, updating balance factors.
3. If a node becomes unbalanced, apply the necessary rotation(s) to restore balance.

5 Implementation

Implement all the functions of an AVL tree. The code should output a menu asking the input from the user. The menu options are 1 (for insert), and 2 (for delete) For example 1 23 would result in adding the key 23 to the AVL Tree. 2 20 would result in removal (if present) the key 20 from the AVL Tree.

6 References

1. M Goodrich, R Tamassia, and M. Goldwasser, "Data Structures and Algorithms in Python", 1^{st} edition, Wiley, 2013.