Assignment # 04
CS 2001 – Data Structures
Fall 2021

Topics: Binary Search and AVL Trees

General Guidelines

1. Write neat and clean code. Avoid any memory leaks and dangling pointers while implementing the scenarios required in this assignment.
2. You can lose the marks if conventions are not strictly followed.
3. Peer plagiarism and the late submissions are strictly not allowed. In case, zero marks will be awarded for whole assignment
4. You need to submit the hard copy your solution as final submission. In other case, zero marks will be awarded.
5. Total Marks: 100

Submission Guidelines

1. Your assignment submission must be in hardcopy (i.e., handwritten, or printed form) and a scanned version should also be uploaded on the SLATE within the given deadline.
2. Analytical and mathematical questions can be handwritten, while code questions should preferably be computer-typed [however, handwritten solutions are also allowed for this assignment only]
3. **Deadline (Monday, 13th December till 04:00 pm)**

Question 1: AVL Tree Implementation [25+5+30+10 = 70 marks]

Implement a class BST that contains the following functions:

- Insert(): insert an element in the tree
- Delete(): delete an element in the tree
- Search(): searches the desired element in the tree
- findMax(): finds the maximum element in the tree
- findMin(): finds the minimum element in the tree
- inorderTraversal() : prints in-order traversal of the tree
- preorderTraversal(): prints pre-order traversal of the tree
- postorderTraversal(): prints post-order traversal of the tree
- treeHeight(): returns the height of the tree
- treeNodeCount(): returns the count of nodes in the tree
- treeLeavesCount(): returns the count of leaves in the tree
- printNodeLevel(): prints level of a node in the tree

The class will be created in a file “binarySearchTree.h” and the functions will be implemented in “binarySearchTree.cpp”. Make sure to create the class as a template so it can run for multiple data types.

Your main function should like:

```
int main(){

// Sample Input: 65 55 22 44 61 19 90 10 78 52

BST<int> myIntBST;

// test all functions here

}
```

Task 1: Convert a sorted array to a BST of minimal height (Use the library you implemented in task 1).

Task 2: Implement another class AVL (AVL.h, AVL.cpp) like you did in task 1. In addition to the methods in task 1, you also must implement methods that keep the tree balanced.

Task 3: Find a pair with a given sum in AVL tree (Use the library you implemented in task 2.

Question 2 : Binary search tree Applications [20+10=30 marks]

Task 1: Given an array nums that represents a permutation of integers from 1 to n. We are going to construct a binary search tree (BST) by inserting the elements of nums in order into an initially empty BST. Find the number of different ways to reorder nums so that the constructed BST is identical to that formed from the original array nums.

For example, given nums = [2,1,3], we will have 2 as the root, 1 as a left child, and 3 as a right child. The array [2,3,1] also yields the same BST but [3,2,1] yields a different BST.

Return *the number of ways to reorder* nums *such that the BST formed is identical to the original BST formed from* nums.

```
Input: nums = [3,4,5,1,2]
Output: 5

[3,1,2,4,5]
[3,1,4,2,5]
[3,1,4,5,2]
[3,4,1,2,5]
[3,4,1,5,2]
```

Task 2: Given the root of a Binary Search Tree (BST), convert it to a Greater Tree such that every value of the original BST is changed to the original value plus the sum of all nodes greater than the current in BST.

As a reminder, a binary search tree is a tree that satisfies these constraints:

- The left subtree of a node contains only nodes with values **less than** the node's value.
- The right subtree of a node contains only nodes with values **greater than** the node's value.
- Both the left and right subtrees must also be binary search trees.