CSE 4304-Data Structures Lab. Winter 2021

Lab-06

Date: July 11, 2021 (Sunday)

Target Group: All Lab groups

Topic: BST, AVL Trees

Instructions:

- Task naming format: fullID_T01L05_1A.c/cpp
- If you find any issues in problem description/test cases, comment in the google classroom.
- If you find any tricky test cases which I didn't include and others might forget to handle, please comment! I'll be happy to add.
- Modified sections will be marked with BLUE colour.

Task-1:

A series of values are being inserted in a BST. Your task is to show the balance factor of every node after each insertion.

balance_factor= heightofLeftSubtree - heightofRightSubtree

The following requirements have to be addressed:

- Continue taking input until -1.
- After each insertion, print the nodes of the tree in an inorder fashion. Show the balance_factor beside each node within a bracket.
- Each node has the following attributes: data, left pointer, right pointer, parent pointer and height of that node. (store balance_factor if needed)
- Your code should have the following functions:
    - Void Insertion (key) // iteratively inserts key
    - Void Update_height(node) // update the height of a node
    - Int height(node) // returns the height of a node
    - Int balance_factor(node) // returns balance factor

Sample Input	**Sample Output**
12 8 5 11 20 4 7 17 18 -1	12(0) 8(0) 12(1) 5(0) 8(1) 12(2) 5(0) 8(0) 11(0) 12(2) 5(0) 8(0) 11(0) 12(1) 20(0) 4(0) 5(1) 8(1) 11(0) 12(2) 20(0) 4(0) 5(0) 7(0) 8(1) 11(0) 12(2) 20(0) 4(0) 5(0) 7(0) 8(1) 11(0) 12(1) 17(0) 20(1) 4(0) 5(0) 7(0) 8(1) 11(0) 12(0) 17(-1) 18(0) 20(2)

Note:

- If the balance_factor isn't stored, showing it for every node might result in O(n) [it's allowed in this solution]. However, after insertion, height needs to be updated only for the ancestors O(**logN**).

Task-2:
Utilize the functions implemented in Task-1 to provide a complete solution for maintaining a 'Balanced BST'.
The program will continue inserting values until it gets -1. For each insertion, it checks whether the newly inserted node has imbalanced any node or not. If any imbalanced node is found, 'rotation' is used to fix the issue.

Your program must include the following functions:

- Void insertion (key) // iteratively inserts key
- Void update_height(node) // update the height of a node
- Int height(node) // returns the height of a node
- Int balance_factor(node) // returns balance factor
- Left_rotate(node)
- Right_rotate(node)
- check_balance(node):
  - check if a node is imbalanced and calls relevant rotations.
- print_avl(root) // inorder traversal

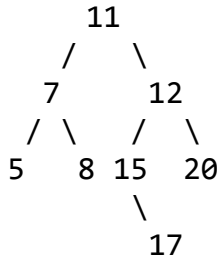
Sample Input	**Sample Output**
12	12(0) Balanced Root=12
8	8(0) 12(1) Balanced Root=12
5	5(0) 8(1) 12(2) Imbalance at node: 12 LL case Left_rotate(12) Status: 5(0) 8(0) 12(0) Root=8
11	5(0) 8(-1) 11(0) 12(1) Balanced Root=8
20	5(0) 8(-1) 11(0) 12(1) 20(0) Balanced

	Root=8
15	5(0) 8(-2) 11(-1) 12(1) 15(0) 20(0) Imbalance at node: 8 RL case Right_rotate(12), Left_rotate(8) Status: 5(0) 8(1) 11(0) 12(0) 15(0) 20(0) Root=11
7	5(-1) 7(0) 8(2) 11(1) 12(0) 15(0) 20(0) Imbalance at node: 8 LR case Left_rotate(5), Right_rotate(8) Status: 5(0) 7(0) 8(0) 11(0) 12(0) 15(0) 20(0) Root=11
17	5(0) 7(0) 8(0) 11(-1) 12(1) 15(-1) 17(0) 20(0) Balanced Root=11
-1	Status: 5(0) 7(0) 8(0) 11(-1) 12(1) 15(-1) 17(0) 20(0)

Note:

- **Do not** use any recursive implementation

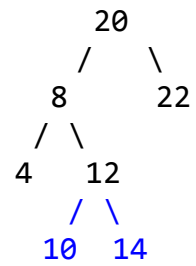
The status of the tree is finally supposed to be like this:



Task-3:

Let T be a rooted tree. The **Lowest Common Ancestor**(LCA) between two nodes n1 and n2 is the lowest node in T with both n1 and n2 as descendants (we allow a node to be its descendant).
The LCA of n1 and n2 in T is the shared ancestor of n1 and n2 located farthest from the root. Computation of lowest common ancestors may be helpful, for instance, as part of a procedure for determining the distance between pairs of nodes in a tree.

Let's assume two nodes of a BST are n1 and n2. Your task is to find out their LCA. For example, suppose the status of the tree is as follows:



LCA of 10 & 14: 12
LCA of 4 & 22: 20
LCA of 4 & 8 : 8
LCA of 4 & 12 :8 (20 is also a common ancestor, but not 'lowest')
LCA of 8 & 14: 8
LCA of 8 & 22: 20
LCA of 20 & 8 : 20
LCA of 10 & 12: 12

Steps:

- At first, a sequence of N integers will be inserted (until -1)
- The numbers are inserted in an AVL Tree. The final status of the tree shown after all nodes are inserted.
- Then there will be a number **T**, followed by T pairs of n1 & n2
- For each pair, your task is to print the LCA.

(PTO)

Sample Input	**Sample Output**
Input sequence 12 8 5 11 20 15 7 17 -1	Final status: 5(0) 7(0) 8(0) 11(-1) 12(1) 15(-1) 17(0) 20(0)
10 5 8 7 12 17 20 7 8 8 7 12 5 12 20 12 17 8 15 11 8	 7 11 12 7 7 11 12 12 11 11

Note:

- Use AVL Tree
- The solution must be O(**logN**)