ECE 5460/6460: VLSI Design Automation

Homework 0
Due: 9/8/2022

AVL TREE ASSIGNMENT

An AVL tree is a self-balancing binary search tree. In such a tree, the difference between the heights of two child sub-trees of any node, i.e, balance factor should not be more than 1. If anytime this balance factor is more than 1, then re-balancing of the tree should be performed. The re-balancing of the tree is done by one or more tree rotations. To understand tree-rotations better: http://en.wikipedia.org/wiki/Tree_rotation.

1. **Programming Assignment (100 Points):** Program an AVL Tree that can perform the following actions:

   (a) Creating tree
   (b) Insertion of a node
   (c) Deletion of a node
   (d) Searching a node
   (e) Deleting Tree
   (f) Traversal of the tree - Pre-order, Post-order and In-order
   (g) Check Balance

   To understand these actions more clearly, you can view this AVL link:

   https://www.cs.usfca.edu/ galles/visualization/AVLtree.html

   (The Print button only shows the inorder traversal.)

2. **Input Format:** Your program should first print out the actions that can be performed:

   Ex: Choose the option to be performed:

   (1) Create_Tree
   (2) Insertion
   (3) Deletion
   (4) Search
   (5) Traversal
   (6) Delete_Tree
   (7) Check_Balance

   This input option should be of *int* type only.

   a. **Create Tree:**
   This option should create a tree by reading an input file, an example is attached below, which contains of one integer per line that creates the tree.
   The example input file (**input.txt**) is attached along with this document. (We may test your code with other input files in the same format).
   Every time this option is chosen, the existing tree should be deleted (*delete_tree()*) and a tree with the given input file should be created.

b. **Insertion:**

   This option should insert a node in the existing tree.

   The input integer value should be typed in by the user.

   If no tree has been created before this option then the entered integer value should be inserted anyway.

   By using this option multiple times, you should be able to create a tree.

   Do not allow duplicate nodes in the tree.

c. **Deletion:**

   This option should delete a node from the existing tree. The input integer value should be typed in by the user.

   If no tree has been created before this option, the program should print out " *Node cannot be deleted. Empty Tree!*"

   If the entered integer value is not found in the created tree, the program should print out "*Node cannot be found.Deletion cannot be performed*".

   When a node is deleted, it can be replaced with either the largest node in the left sub-tree or the smallest node in the right sub-tree. Give the left sub-tree priority for ease of grading.

d. **Search:**

   This option should search a node in the existing tree. The input integer value should be typed in by the user.

e. **Traversal:**

   Selecting this option should print out the options to choose the type of traversal to be performed:

   i. ***Pre_order***
   ii. ***Post_order***
   iii. ***In_order***

   The input to choose the type of traversal should be of *int* value.

f. **Delete_Tree:**

   Choosing this option should delete the whole existing tree. No inputs required.

g. **Check_Balance:**

   This option should not take any input. It should just check if the balance_factor (as mentioned above) is 1 or more.

   If the balance_factor $>$ 1, tree_rotation operations should be performed to balance the tree.

   Apart from being an independent option, this operation should be performed automatically after every create_tree, insertion and deletion operation to check if the tree is an AVL tree (balance_factor = 1) or not.

3. **Output Format:** The output should be printed on your console **AND** written to a file. The filename should be of the format lastname_A#.txt. Make sure it clearly prints the actions as follows:

   a. **Create Tree:**

      After creating the tree, this option should print the values of the tree in **<u>Pre-order</u>** traversal.

b. **Insertion:**
   After inserting a node, this option should print the updated tree in **Pre-order** traversal.

c. **Deletion:**
   After deleting a node, this option should print the updated tree in **Pre-order** traversal.

d. **Search:**
   This option should print the position, i.e, height at which the node was found.

e. **Traversal:**
   This option should print the tree for the chosen type of traversal.

f. **Delete_Tree:**
   After successfully, deleting the whole tree, this option should just print "Tree Deleted"

g. **Check_balance:**
   This option should print the balance_factor every time its performed.

For ease of grading, please output the name of the action followed by the action's result (one action/result per line of the file). For example:

```
...
Create Tree: 36 17 5 30 28 81 37 76 83
Check balance: 1
Delete Tree: Tree Deleted
```

4. **Submission:** Please submit a <A#_Homework0>.tar.gz file that contains the following:

   a. A text file with instructions on extracting and executing your program
   b. Your code with comments (all *.h and *.cpp files that you have written)
   c. Your executable file

5. **Grading:** The assignment will be graded on the following criteria:

   a. **Correct Execution (90 Points)** - We will be testing your program for different input files. Your program should compile and execute properly, without any errors and warnings for every input.

      i. **Create tree (15 Points)**
      ii. **Insertion (15 Points)**
      iii. **Search (15 Points)**
      iv. **Deletion (15 Points)**
      v. **Traversal (5 Points for each traversal - Total 15 Points)**
      vi. **Delete_Tree (15 Points)**
      vii. **Check_Balance (Optional)**

   b. **Code commenting (10 Points)** - will be awarded for properly commenting the code.