CPSC 2150 – Algorithms and Data Structures II

Assignment 2: AVL trees

Total - 100 Marks

"Good design adds value faster than it adds cost."
- Thomas C. Gale

Learning Outcomes

- Design, implement and analyze the tree-based algorithms in terms of time and space complexity.
- Design and implement the operations of AVL trees.
- Design and implement effective object-oriented programming by using inheritance
- Write recursive solutions to non-trivial problems, such as tree traversals.
- Develop C++ code based on the existing constraints.

Resources

- Chapter 10.2 of the text book

Description

This is a practice on design, implement and analyzing problems using AVL trees.

[50 marks] Part A: Provide an appropriate data structure and necessary methods to build an AVL tree class which enables you to complete the rest of this assignment. Define your class with the generic types. The AVL class must be designed as a subclass of BST class (Assignment 3). Then add balancing routines and enough member function as needed in the rest of this assignment.

Implement single and double rotations and make sure that during insertions and deletions your tree maintains the AVL property. I recommend writing a check_ballance() method that ensures your tree is balanced and using this for testing. For full credit insertion and deletion must be logarithmic time. This means you have to compute the height of a node in constant time. I recommend writing the tree first using an expensive, recursive height() method and once it's working, develop another mechanism for computing the height. More details about inserting and deleting node in AVL trees is coming in part B.

Part B: The following non-member functions are required to be programed as part of **testAVL.cpp**. In order to implement the following functions, you can call member function(s) of AVL class accordingly. Your design in part A will affect your functions in this part.

[10 marks] void DeleteAVLTree(AVLTree *tree): Frees an AVL tree and all it's nodes.

[10 marks] AVLTree *BuildAVLTree(string filename): Takes a file name as input and builds an AVL tree from the information in the file. The first number in the file(n) will represent the number of data points to be put into the tree. The following n integers (greater than 1000 and less than 5000) are data for the tree. For example, the following shows an input file that includes 3 nodes and those nodes are 1068, 2567, and 1342.

```
3
1068
2567
1342
```

[8 marks] void InsertNodeAVL(AVLTree *tree, int element): This function should insert a node according to the AVL insertion rules. For simplicity, the rules of AVL insertion have been summarized for you here:

• When inserting a node, ensure that data of every node is greater or equal than the data of the nodes in its left subtree, and strictly less than those in its right subtree.

• After insertion you have to re-calculate balance factors of all the nodes from the new node to the root. So, follow parent pointers back toward the root, calculating each node's new balance factor based on

i) its previous balance factor,
ii) whether you have arrived there from a left or right child. Calculation stops under one of 3 conditions:
   1. You re-calculate a node's balance factor as balanced
   2. You reach the root, and no node's re-calculated balance factor is unbalanced.
   3. You re-calculate a node's balance factor as unbalanced.
   If you reach the third condition then you have to rebalance the tree around the earliest unbalanced ancestor (EUA).

In order to rebalance, you have to consider 4 separate cases (we will only show 2 cases, as the others are symmetrical). For this, we will call the left child of the EUA the EUAC (child):

> 1. If the newly-inserted node is in the left subtree of EUAC, perform a single right rotation around the EUA. After the rotation both EUA and EUAC should have balance factors set to balanced, and no other balance factors are changed. 2. If the newly-inserted node is in the right subtree of EUAC, call EUAC's right child EUAGC (grandchild) and perform a double rotation (LR)

[8 marks] void DeleteNodeAVL(AVLTree *tree, int element): This function should delete only one node if there is duplicate nodes, according to the AVL deletion rules. For simplicity, the rules of AVL deletion have been summarized for you here:

i) Delete the node using delete by copy

ii) After deletion you have to re-calculate balance factors of all the ancestors of deleted node to the root. If the node is balanced you can move on to the next ancestor. Otherwise, there are two cases:

   1. The node is right heavy, then do left rotation
   2. The node is left heavy, then do right rotation

[4 marks] void printAVL(AVLTree *tree): Prints pre-order and in order traversal of the given AVL tree.

[10 marks] calculate the time complexity of your functions in part B except main function.

The main function of your test program (testAVL.cpp) should be compatible with the following main function. By compatible it means the number of parameters of each function should match. The type of parameters is determined by your design and implementation. This makes some constraints in your design, which is likely to occur in the real world.

```
int main() {
  // declaration of your variables ...
  n = getInput();  // n is a non-negative integer that can be read from user or
                   // generated randomly by computer
  genInputFile(n, filename); //generates an input file as explained in part B

  avl = BuildAVLTree(filename);
  cout << "height of AVL tree is:" << avl->height() << endl;
  printAVL(avl);

  cout << "Enter a value to insert: ";
  cin >> node;
  InsertNodeAVL(avl, node);
  cout << "height of AVL tree is:" << avl->height() << endl;
  printAVL(avl);

  cout << "Enter a value to delete: " ;
  cin >> node;
  DeleteNodeAVL(avl, node);
  cout << "height of AVL tree is:" << avl->height() << endl;
  printAVL(avl);

  DeleteAVLTree(avl);
  printAVL(avl);

  return 0;
}
```

SUBMIT to D2L

Submit a zip file named **StudentNumber-Asgn2.zip** including all related files by the due date. For example, if your student number is 10023449, the submitted file must be named as **10023449-Asgn2.zip**.