2413 Assignment 8

Due Date
4-4-16 11:59pm

For this assignment you will be extending your implementation of the Binary Search Tree to AVL Trees containing integers. Duplicate integers are allowed in the tree and the BST Property is loosened to allow equal elements on either sub-tree.

The Binary Search Property Applies to every node :

1. The elements of the left subtree are $\leq$ the current node's element
2. The elements of the right subtree are $\geq$ the current node's element

The AVL Tree Property extends the BST property with the following for every node:

1. The height of the left subtree and the right subtree differ by at most 1.

The new property will only impact the insert and remove functions, but you will need to create several auxiliary functions to help maintain the AVL Tree Property.

Same Input Commands From Standard In:

```
insert <integer>
remove <integer>
postorder
preorder
inorder
```

Suggested Helper Functions

```
node * RotateLeft(node *);
node * RotateRight(node *);
void RecalcHeight(node *);
int Height(node *);
node * Rebalance(node *);
node * getLeastNode(node *, node ** least);
```

You should be able to implement Rotate Left symmetrically from Rotate Right.

In order to generate gradeable output beyond traversals, your rotation functions will print the content of 3 nodes before performing their rotations. You will print the name of the rotation, the root's left child's element (or NULL), the root's element, and then the root's right child's element (or NULL)

Example Input and Rotation Output

```
insert 10
insert 20
insert 15
```

Output

```
RotateRight 15 20 NULL
RotateLeft NULL 10 15
```

0.1 Rotations

nodes * RotateRight(node * n) //similar for left rotation

1. Rotates n to the right (clockwise):
   (a) print the appropriate message (left child, root, right child)
   (b) the new root of the sub-tree will be the node on n's left.
   (c) n becomes n's left's right child.
   (d) n's left's right sub-tree becomes n's left sub-tree.
2. be sure to correct the heights of the nodes after the rotation.

0.2 Recalculate Height

void RecalcHeight(node * n)

1. Changes n's height to be 1+ the max of the height of n's right and left children.

0.3 Recalculate Height

int Height(node * n)

1. returns 0 when n is null otherwise return's n's height

0.4 Rebalance

node * Rebalance(node *n)

1. If n is null, return n;
2. Check the difference in heights between n's children.
3. If the difference is between -1 and 1, return n
4. Else there are 4 cases to consider.
5. n's left's height ¿ n's right's height
   (a) if n's left's right's height ¿ n's left's left's height, then RotateLeft on n's left
   (b) RotateRight on n
6. n's left's height ¡ n's right's height
   (a) if n's right's right's height ¡ n's rights's left's height, then RotateRight on n's right
   (b) RotateLeft on n
7. return the new root after rotation.

1 AVL Tree Insert

nodes * recAVLinsert(node * n, element e)

1. Recursively walk the tree to the correct position for inserting as in a BST.
2. Create the new node for the element as a leaf.
3. Rebalance the tree at the current node before returning from the recursive function.
4. return the resulting new root from rebalancing.

2 getLeastNode

nodes * getLeastNode(node * n, node ** least)

1. if n is null, set *least to null and return null.
2. Otherwise recursively walk the tree to the left until the node with no left child is reached.

3. set *least to point to this node.
4. return *least's right child as the new root of the subtree.
5. rebalance the tree after returning from each recursive call and return the new root.

3 AVL Tree Remove

If an instance of e is found, a single instance of element e is removed. To remove multiple occurrences of e, the function must be called until no remove occurs. The success or failure of removing an element e is communicated through the pointer to an integer.

nodes * recAVLremove(node * n, element e, int * removed)

1. Recursively walk the tree as in the BST search of e. If no node contains element e, set *removed to 0 and return the roots of the sub-trees encountered.
2. If an instance of e is found,
   (a) if the right sub-tree is empty,
      i. set *removed to 1,
      ii. delete the node containing e
      iii. return the left-subtree of the deleted node as the new root.
   (b) If the right sub-tree is not empty
      i. set *removed to 1
      ii. remove the least node from the right sub-tree (rebalancing the right sub-tree in the process)
      iii. replace the node containing e with the least node (updating height)
      iv. delete the node containing e
      v. return the rebalance of the new root (the least node).
3. rebalance on the return from recursive calls and return the result

4 postorder

Print the integers of the tree followed by a single white space in post-order traversal. When done, print a newline character.

1. print left subtree

2. print right subtree
3. print current node

If tree is empty, print: "Tree Is Empty" on its own line

5 preorder

Print the integers of the tree followed by a single white space in pre-order traversal. When done, print a newline character.

1. print current node
2. print left subtree
3. print right subtree

If tree is empty, print: "Tree Is Empty" on its own line

6 inorder

Print the integers of the tree followed by a single white space in in-order traversal. When done, print a newline character.

1. print left subtree
2. print current node
3. print right subtree

If tree is empty, print: "Tree Is Empty" on its own line

7 Input Output Pairs

7.1 Example

INPUT:

```
insert 10
insert 1000
insert 500
insert 750
insert 625
insert 575
insert 550
insert 600
insert 612
insert 590
insert 585
remove 1000
inorder
postorder
preorder
```

OUTPUT:

```
RotateRight 500 1000 NULL
RotateLeft NULL 10 500
RotateRight 750 1000 NULL
RotateRight 625 750 1000
RotateLeft 10 500 625
RotateLeft NULL 575 600
RotateRight 575 600 612
RotateLeft 10 500 575
RotateLeft 500 55 600
RotateRight 600 625 700
RotateRight 575 600 625
10 600 575 585 590 600 612 625 750
10 500 585 590 612 750 625 600 575
575 500 10 600 590 585 625 612 750
```