Colton Parham, CDP210001, Dr. Zhao

Bonus assignment - Pseudocode

class AVL

{

// AVL Tree node

stat class Node

{

int data

Node left, right

}

// new node function

static nNode nodes(int data)

{

// data being associated with the key value

Node node = new Node

node.data = data

node.left = node.right = null

return node

}

// rotate right

static node rotateR(node xV)

{

Node yV = xV.left

xV.left = yV.right

yV.right = xV

return yV

}

// rotate left

static node rotateL(node xV)

{

Node yV = xV.right

xV.right = yV.left

yV.left = xV

return yV

}

static node splayAction(Node rootV, int keyVal)

{

if (rootV == null || rootV.keyVal == keyVal) then (return rootV)

if (rootV.keyVal > keyVal)

{

if (rootV.left == null) then (return rootV)

//Zig - Zag being performed here, with 2 lefts

if (rootV.left.keyVal > keyVal) then (rootV.left.left = splayAction(rootV.left.leftm keyVal), rootV = rotateR(rootV))

else if (rootV.left.keyVal < keyVal)

{

// recursively bringing the key as root, of the left->right

rootV.left.right = splayAction(rootV.left.right, keyVal)

if (rootV.left.right != null) then rotateL(rootV.left)

}

// i think right here we would need a conditioanal return statement that would sort of go into the idea of checking the root.left being null, and returning that value

return rootV.left == null <checker here, if statement potentially?>

rootV -> rotateR(rootV)

}

// keyVal stays in the right tree.

else

{

if (rootV.right == null) then (return rootV)

if (rootV.right.keyVal > keyVal)

{

rootV.right.left = splayAction(rootV.right.left, keyVal)

if (rootV.right.left != null) then (rootV.right rotateR(rootV.right))

// right-right (zig-zag)

else if (rootV.right.keyVal < keyVal) then (rootV.right.right = splayAction(rootV.right.right, keyVal), rootV = rotateL(rootV))

return rootV.right == null <checker here, if statement potentially?>

rootV -> rotateL(rootV)

}

}

}

static Node searchVal(Node rootV, int keyVal)

{

// search function

return splayAction(rootV, keyVal)

}

// just like the original project - using same preorder method ideas

stat void preorder(Node rootV)

{

print(rootV.keyVal + " - ")

preorder(rootV.left)

preorder(rootV.right)

}

// driver function or main

main function

// this is what will need to be inserted, and as demanded by the parameters zig->zag

rootV = nNode(a)

rootV.left = nNode(b)

rootV.right = nNode(c)

// zig zag

rootV.left.left = newNode(d)

rootV.left.left.left = newNode(e)

rootV.left.left.left.left = newNode(f)

preorder(rootV)

}