BST insert algorithm

Due to the difficult in understanding references, insertion into a BST is a tricky thing at first. The main idea is to move the element down through the tree, as though searching for it, until we find an empty space ("below" a leaf) to insert the new node.

Pseudo-code:

To insert a value at a given node:

1. *If the node is NULL, create a new node here with the value.*
2. *Otherwise, if the data at this node is less than the new value:*
   Insert the value in the right subtree.
3. *Otherwise, if the data at this node is greater than the new value:*
   Insert the value in the left subtree.

Another version:

Compare data of the root node and element to be inserted.

1. *If the data of the root node is greater, and if a left subtree exists, then repeat step 1 with root = root of left subtree. Else, insert element as left child of current root.*
2. *If the data of the root node is greater, and if a right subtree exists, then repeat step 2 with root = root of right subtree. Else, insert element as right child of current root.*

Quite a few different implementations can be found on the next pages.

```
public void insertNode(int key) { //client method
    root = insertNode(root, new Node(key));
}

/* a very compact recursive implementation, takes a bit to understand though.  practice adding a
few leaves to a BST to mentally track how it works */
private Node insertNode(Node currentParent, Node newNode)
{
    if (currentParent == null)
        return newNode;

    else if (newNode.key > currentParent.key)
        currentParent.right = insertNode(currentParent.right, newNode);

    else if (newNode.key < currentParent.key)
        currentParent.left = insertNode(currentParent.left, newNode);

    return currentParent;
}
```

```
public void insert(int key) { //client method
   this.root = insert(root, key);
}

/* Another slick bottom-up recursive implementation.  This version also applies the "x = change(x)"
pattern and may take time to understand.  Running it through BlueJ's debugger may help */
private Node insertRec(Node root, int key)
{
    //If the tree is empty, return a new node
    if (root == null)
        root = new Node(key);

    //Otherwise traverse tree
    if (key < root.key)
        root.left = insert(root.left, key);

    else if (key > root.key)
        root.right = insert(root.right, key);

    //return the (unchanged for non-empty trees) node pointer
    return root;
}
```

```
public void insert(int data) { //client method
   this.root = insert(this.root, data);
}

//similar to the previous version, but the extra "else" makes it bit more intuitive
private Node insert(Node node, int data)
{
   if (node == null)
      node = new Node(data);

   else {
      if (data <= node.data)
         node.left = insert(node.left, data);

      else
         node.right = insert(node.right, data);
   }

   return node; //in any case, return the new pointer to the caller
}
```

(more on next page)

```
public void insert(int newVal) //iterative version, with a while loop
{
        if (this.root == null)
                this.root = new Node(newVal);

        else
        {
                Node current = this.root;

                while (true)
                {
                        if (newVal < current.val) {
                                if (current.left == null) { //found a leaf, add new node
                                        current.left = new Node(newVal);
                                        break; //found insertion position, terminate
                                }
                                else
                                        current = current.left;
                        }
                        else if (newVal > current.val) {
                                if (current.right == null) {
                                        current.right = new Node(newVal);
                                        break; //found insertion position, terminate
                                }
                                else
                                        current = current.right;
                        }
                        else //the duplicate case we're ignoring
                                break;
                }
        }
}
```

```
/* another iterative solution, this time with a "parent" reference to remember where the leaf
should go (rather than doing it inside the loop */
public void insert(int key)
{
        if (this.root == null) {
                this.root = new Node(key);
                return;
        }

        Node temp = this.root, parent = null;

        while (temp != null) //traverse tree until leaf is found
        {
                parent = temp; //update parent reference

                if (key < temp.val) //move through tree appropriately
                        temp = temp.left;

                else if (key > temp.val)
                        temp = temp.right;
        }

        if (key < parent.val) //add the new leaf node on correct side
                parent.left = new Node(key);

        else if (key > parent.val)
                parent.right = new Node(key);

}
```

(more on next page)

```
/* Another recursive implementation, this time where the entire tree being empty is handled by the
client method.  Might be easier to understand for some */
public void insert(int newVal)
{
        if (this.root == null)
                this.root = new Node(newVal);

        else
                this.insert(newVal, this.root);
}

public void insert(int newValue, Node node)
{
        if (newValue < node.val)
        {
                if (node.left != null) //haven't reached a leaf yet
                        insert(newValue, node.left);

                else //reached a leaf, insert a new node
                        node.left = new Node(newValue);
        }

        else if (newValue > node.val)
        {
                if (node.right != null)
                        insert(newValue, node.right);

                else
                        node.right = new Node(newValue);
        }
}
```

```
//Another working but sub-optimal recursive solution, similar to the previous method
public void insert(int value) {
    if (overallRoot == null)
        overallRoot = new IntTreeNode(value);

    else if (overallRoot.data > value)
        insert(overallRoot.left, value);

    else if (overallRoot.data < value)
        insert(overallRoot.right, value);

    // else overallRoot.data == value; a duplicate (don't add)
}

private void insert(IntTreeNode root, int value) {
    if (root.data > value)
    {
        if (root.left == null)
            root.left = new IntTreeNode(value);
        else
            insert(overallRoot.left, value);

    }
    else if (root.data < value)
    {
        if (root.right == null)
            root.right = new IntTreeNode(value);
        else
            insert(overallRoot.right, value);
    }
    // else root.data == value; a duplicate (don't add)
}
```

```
//don't even try, coding is too hard
public void insert(int newVal) {
         return;
}
```