Marc Glass

Prof Michael N. Rissover

CS300 DSA: Analysis and Design

4/1/2023

Module 5 Code Reflection

* **Code Reflection:**
  + This code creates a BST data structure from data in a csv file and allows the user to insert, remove and search for a particular object in the list. I am not really sure if I understood the instructions entirely, because there are many functions yet to be completed that aren’t included in the instructions. I did them anyways, since it wouldn’t have worked without them. I was able to pretty much use the zybook examples to complete the required functions, however, there were several rough patches. A few hang ups that I had were that I continue to forget about the “compare()” function, which certainly makes everything much easier when trying to compare objects. I also could not for the life of me figure out how to translate the remove algorithm into this code, so I went online for some help. I am still not returning anything from the search when the program runs, so I don’t know if the remove method works entirely. I would love to know what is wrong with the search function, since it seems this is a recurring problem for me. Thankfully, many of the issues that I have had were questions that have been asked quite a few times on such sites as Stack Overflow, and I was even able to clear some things up. That being said, I still feel like I have a lot to learn.

**Pseudocode or Flowchart:**

Insert( bid ){

If the root is null

Make the root equal to a new node

Set the root’s bid data equal to the given bid

Set the left and right child nodes to null

Else

Add new node

}

Search(bidId){

Set current node equal to root

While the current node is not null

If the current nodes data matches,

Return the current node’s data

If the current node is less than the given data, bidId

The current node points to the left child

If the current node is greater than the given data, bidId

The current node points to the right child

Return bid

}

addNode(node, bid){

if the node is greater than the given data(bid)

if the node has no left child

this node becomes the left child

else add node to tree

else

if node has no right child

this node becomes the right child

else add node to tree.

}

inOrder(node){

if the node is not null

CALL recursive inOrder to left child

PUT to output bid data

Call recursive inOrder for right child

}

postOrder(node){

if the node is not null

CALL recursive postOrder to left child

CALL recursive postOrder to right child

PUT bid data to output

}

preOrder(node){

if the node is not null

PUT bid data to output

CALL recursive postOrder for left child

CALL recursive postOrder for right child

}

removeNode(node, bidId){

if the node is not null

return node

if the bidId is less than the nodes bidId

left child is removed recursively

if the bidId is greater than the nodes bidId

right child is removed recursively

if there are no children, delete node and set node to null

if right child is null and the left child is not null

set temporary variable equal to node

set node equal to left child

delete temporary variable

if right child is not null and the left child is null

set temporary variable equal to node

set node equal to right child

delete temporary variable

if right is not null and left is not null

set temporary variable to right child

while right child’s left child is not null

set temporary variable to right child’s left child

SET node’s data to temporary variable’s data

REMOVE node’s right child

Return the node

}