Daniel Schween

CS 300 SNHU

08/04/2024

**Module Five BST: Code Reflection**

The code is broken down into the following functions and classes:

* Class BinarySearchTree
  + Class definition of the BinarySearchTree objects that hold
* Private members: size, root, and inOrder()
* Public methods: Constructor, destructor, InOrder(), Insert(), Remove(); Search(); Size()
* BinarySearchTree()
* Default constructor that sets the root to a null pointer
* BinarySearchTree::Insert(Bid)
* A Bid is passed
* The function then adds the passed Bid as a new Node in the tree (nodes)
* BinarySearchTree::InOrder()
* Function loops through the BinarySearchTree starting at the root and outputs, to the console, 4 values from the Bid struct
* BinarySearchTree::Remove(String)
* Function starts at the root and searches for the String
* Upon finding the String the Node containing it is freed from memory and tree is updated
* BinarySearchTree::Search(String)
* Function starts at the root and searches for the bidId
* BinarySearchTree::Size()
* A get method used to access the private member, size
* strToDouble
* Used to convert the CSV file data into useable value
* Bid
* Struct containing the data
* Variable is used within the vector that will be sorted
* Node
* Struct containing the Node data
* Consists of a Bid and 2 pointers: left and right
* loadBids

- Function used to read in the csv data

- Can read the csv path in from arguments or use a default path

* Main
* main is the primary driver for the application

**PSEUDOCODE:**

**BinarySearchTree::InOrder()**

**CHECK** if Node is null and if so return

**CALL** node’s left pointer which will find the left most Node

**OUTPUT** to console: *bidId, title, amount, fund*

**CALL** node’s right pointer which will find the right most Node

**END**

**BinarySearchTree::PostOrder()**

**CHECK** if Node is null and if so return

**CALL** node’s left pointer which will find the left most Node

**CALL** Node’s right pointer which will find the right most Node

**OUTPUT** to console: bidId, title, amount, fund

**END**

**BinarySearchTree::PreOrder()**

**CHECK** if Node is null and if so return

**CALL** node’s left pointer which will find the left most Node

**CALL** Node’s right pointer which will find the right most Node

**OUPUT** to console: bidId, title, amount, fund

**END**

**BinarySearchTree::Insert(*Bid*)**

**CHCK** if the root is Null

**IF** the root is Null,

**CREATE** a new Node containing Bid

**IF** root is not Null

**LOOP** until Node is Null

**IF** bidId within Bid is less than Node’s current bidId

**IF** *Node’s* left pointer is Null

**SET** left pointer to a new Node containing Bid

**RETURN** Node

**ELSE** set Node at the left pointer

**ELSE**

**IF** the Node’s right pointer is Null

**SET** right pointer to a new Node containing Bid

**ELSE** set Node at the right pointer

**END**

**BinarySearchTree::Search(*String*)**

**CREATE** a new Node pointer called current

**SET** current *Node* to the *root*

**LOOP** until current Node is NULL

**IF** the Node at current Node contains a bidId equal to 0

**RETUN** current Node’s Bid

**IF** the Node at current Node contains a bidId less than 0

**SET** current Node equal to the left Node

**ELSE**

**SET** current Node equal to the right Node

**RETURN** an empty Bid

**END**

**BinarySearchTree::addNode(Node\* node, Bid bid)**

**IF** node is larger than zero, add to the left of tree

**IF** no left node

**CREATE** a new left node

**ELSE**

**RECURSE** down the left node

**ELSE**

**ADD** node to the right subtree

**IF** right node equals NULL

**CREATE** new right node

**BinarySearchTree::RemoveNode(node, bidId)**

**IF** node is Null

**RETURN** node

**RECURSE** down the left subtree

**IF** bidId is less than 0

**REMOVE** left node

**ELSE IF** bidId is greater than 0

**DELETE** node

**ELSE**

**IF** there are no children and is leaf node

**DELETE** leftnode

**ELSE IF** one child to the left of tree

**DELETE** left node

**ELSE IF** one child to the right of the tree

**DELETE** right node

**ELSE**

**WHILE** left node is not NULL

**SET** temp to left node

**SET** node to bid

**REMOVE** right node and bidId

**RETURN** node

**END**