Hiep Ha

CS-300

Professor Nathan Lebel

Oct. 4, 2024

5-2 Assignment: Binary Search Tree

**Task 1: Define structures for the tree node and housekeeping variables.**

DEFINE Node structure:

- Contains a Bid object

- Contains two pointers: left (Node) and right (Node)

DEFINE BinarySearchTree class:

- Contains a root (Node) initialized to null in the constructor

**Task 2: Implement inserting a bid into the tree.**

FUNCTION Insert(bid)

IF root is null THEN

CREATE a new Node with bid

SET it as root

ELSE

CALL addNode(root, bid)

END IF

FUNCTION addNode(node, bid)

IF bid.bidId is less than node.bid.bidId THEN

IF left child is null THEN

CREATE a new Node as left child

ELSE

RECURSE addNode with left child

END IF

ELSE

IF right child is null THEN

CREATE a new Node as right child

ELSE

RECURSE addNode with right child

END IF

END IF

END FUNCTION

**Task 3: Implement removing a bid from the tree.**

FUNCTION Remove(bidId)

CALL removeNode(root, bidId)

FUNCTION removeNode(node, bidId)

IF node is null THEN

RETURN null

END IF

IF bidId is less than node.bid.bidId THEN

RECURSE removeNode on left child

ELSE IF bidId is greater than node.bid.bidId THEN

RECURSE removeNode on right child

ELSE (node with bidId found)

IF node has no children THEN

DELETE the node

RETURN null

END IF

IF node has one child THEN

REPLACE node with its child (left or right)

DELETE the node

RETURN the child

END IF

IF node has two children THEN

FIND the smallest node in the right subtree

REPLACE node’s bid with the smallest node’s bid

RECURSE removeNode on the right subtree

END IF

END IF

RETURN updated node

END FUNCTION

**Task 4: Implement searching the tree for a bid.**

FUNCTION Search(bidId)

SET current node to root

WHILE current node is not null DO

IF bidId matches current node’s bidId THEN

RETURN current node’s bid

END IF

IF bidId is less than current node’s bidId THEN

MOVE to the left child

ELSE

MOVE to the right child

END IF

END WHILE

RETURN an empty bid // If the bid is not found

END FUNCTION

**Task 5: Complete the function to display all bids.**

FUNCTION InOrder()

CALL inOrder(root)

FUNCTION inOrder(node)

IF node is not null THEN

RECURSE inOrder on left child

DISPLAY the current node's bid

RECURSE inOrder on right child

END IF

END FUNCTION

**Code Reflection**

The primary purpose of this code is to implement a binary search tree to store and manage bids. The techniques used include recursive traversal for insertion, deletion, and search operations, ensuring that the tree maintains its sorted order.

One challenge encountered was managing node deletion, especially when dealing with nodes that have two children. This was resolved using a common strategy of replacing the node with the smallest value from the right subtree. Another challenge was ensuring the recursive functions handled edge cases, such as an empty tree or deleting nodes with only one child. These were overcome by carefully managing pointers during recursion.