5-2 Code Reflection

Armon Wilson

SNHU

CS300 – Data Structures & Algorithms

**Insert(Bid)**

**Purpose:**

To insert a new bid into the binary search tree, maintaining the binary search tree property (left child < parent < right child).

**Techniques Implemented:**

* Binary search tree traversal: The function starts at the root and compares the bid ID to determine whether to move left or right.
* Node insertion: A new node is created to hold the bid data and is inserted at the appropriate leaf position in the tree.
* Recursive (via addNode) and iterative implementations: The code provides both recursive and iterative versions of the insert operation.

**Challenges & Solutions:**

* Duplicate bid IDs: The original code didn't handle duplicate bid IDs.
  + Solution not presented through In-code comments/instructions.
* The first implementation of the addNode function unnecessarily created new Node objects when it could reuse the newNode created in the Insert function. This was fixed by passing the newNode into addNode.

**Pseudo-Code:**

FUNCTION Insert(bid)

newNode = CREATE Node(bid)

IF root is nullptr THEN

root = newNode

ELSE

current = root

WHILE current is not nullptr DO

IF bidId < current->bidId THEN

IF current->left is nullptr THEN

current->left = newNode

EXIT WHILE

ELSE

current = current->left

ELSE

IF current->right is nullptr THEN

current->right = newNode

EXIT WHILE

ELSE

current = current->right

**Search(string bidId)**

**Purpose:**

Search for a Bid with the given bidId in the binary search tree.

**Techniques Implemented:**

Tree Traversal: Iterative traversal of the tree, following the binary search tree property to efficiently find the target bid.

**Challenges & Solutions**:

No challenges during implementation. All tests successful.

**Pseudo-Code:**

FUNCTION Search(bidId)

current = root

WHILE current is not nullptr DO

IF current->bidId == bidId THEN

RETURN current->bid

ELSE IF bidId < current->bidId THEN

current = current->left

ELSE

current = current->right

RETURN empty Bid object // bidId not found

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

**Purpose:**

Recursively adds a Bid to the tree**.**

**Techniques Implemented:**

Recursion: The function calls itself to traverse down the tree until it finds the correct insertion point.

**Challenges & Solutions:**

No challenges during implementation. All tests passed.

**Pseudo-Code:**

FUNCTION addNode(node, bid)

IF node is nullptr THEN

RETURN new Node(bid)

IF bidId < node->bidId THEN

node->left = addNode(node->left, bid)

ELSE

node->right = addNode(node->right, bid)

RETURN node

**inOrder(Node\* node)**

**Purpose:**

Traverse and print the binary search tree in the specified order.

**Techniques Implemented:**

Recursively traverse the tree to visit the nodes in the correct order.

**Challenges & Solutions:**

Initial comments had the wrong function name mentioned. These were corrected to accurately reflect the traversal being performed.

**Pseudo-Code:**

FUNCTION inOrder(node)

IF node is not nullptr THEN

inOrder(node->left)

PRINT node->bid

inOrder(node->right)

**postOrder(Node\* node)**

**Purpose:** Traverse and print the binary search tree in the specified order.

**Techniques Implemented:**

Recursively traverse the tree to visit the nodes in the correct order.

**Challenges and Solutions:**

Initial comments had the wrong function name mentioned. These were corrected to accurately reflect the traversal being performed.

**Pseudo-Code:**

PostOrder

FUNCTION postOrder(node)

IF node is not nullptr THEN

postOrder(node->left)

postOrder(node->right)

PRINT node->bid

**preOrder(Node\* node)**

**Purpose:**

Traverse and print the binary search tree in the specified order.

**Techniques Implemented**:

Recursively traverse the tree to visit the nodes in the correct order.

**Challenges & Solutions:**

Initial comments had the wrong function name mentioned. These were corrected to accurately reflect the traversal being performed.

**Pseudo-Code:** FUNCTION preOrder(node)

IF node is not nullptr THEN

PRINT node->bid

preOrder(node->left)

preOrder(node->right)

**removeNode(Node\* node, string bidId)**

**Purpose:**

Remove a node with the given bidId from the binary search tree, maintaining the binary search tree property.

**Techniques Implemented**:

* Recursion: The function calls itself to traverse down the tree until it finds the node to remove.
* Node Deletion: Handles different cases:
  + Leaf node (no children)
  + Node with one child
  + Node with two children (finding inorder successor)
* Pointer Manipulation: Updates parent and child pointers to maintain the tree structure after removal.

**Challenges & Solutions:**

I had a logic issue where if the node to be deleted was the last remaining node (the root), the root pointer of the tree was not being updated to nullptr after deletion. This resulted in a dangling pointer, which could cause crashes or undefined behavior. To fix this, I had to add an explicit check in the Remove function to see if the root node was being deleted. If it was, the root pointer of the binary search tree would be set to nullptr after the deletion was completed.

**Pseudo-Code:**

FUNCTION removeNode(node, bidId)

IF node is nullptr THEN

RETURN node

ELSE IF bidId < node->bidId THEN

node->left = removeNode(node->left, bidId)

ELSE IF bidId > node->bidId THEN

node->right = removeNode(node->right, bidId)

ELSE // Match found

IF node is leaf node THEN

DELETE node

node = nullptr

ELSE IF node has one child THEN

tmp = node->left OR node->right // Whichever is not nullptr

DELETE node

node = tmp

ELSE // Two children

tmp = findMinNode(node->right)

node->bid = tmp->bid

node->right = removeNode(node->right, tmp->bidId)

RETURN node