Module Five Binary Search Tree Reflection & Pseudocode

The following is a reflection the code & it’s functions.

The pseudocode for the “Binary Search Tree” “BST” revolves around handling common operations like insertion, searching, removal of nodes, & navigation of the tree in order. It begins with a main function, where the program reads a CSV file, loads data into the tree & provides a user-friendly menu to allow various actions.

For example, a user can insert bids into the tree, searches for specific bid by their id, or remove the bids from the tree. It also tracks the time it takes to perform actions like reading in the CSV data or search for the bids, giving insight of it’s efficiency of the processes for the user. The loop then will run until the user decides to exit the application by selecting the appropriate from the menu.

Each operation in the tree has details listed from the pseudocode details how nodes are navigated based on the bidIDs. For example, the **Insert** function starts at the root and compares each new bid to the current node’s ID. If the new bid is smaller, the function moves to the left; if larger, it moves right until the correct spot is found to insert the bid. The **InOrder** function ensures that bids are printed in a sorted order by recursively visiting the leftmost nodes, printing their values, and then visiting the right nodes.

Similarly, the **Search** function begins at the root and continues left or right depending on the comparison of bid IDs until it finds the desired node. If the **Remove** function finds the node to be deleted, it adjusts the tree structure accordingly to maintain proper BST order, using a successor to replace the deleted node if necessary.

BinarySearchTree

The code is organized into the following components:

Class: BinarySearchTree

* + Contains:
    - Private members: size, root, and inOrder()
    - Public methods: Constructor, Destructor, InOrder(), Insert(), Remove(), Search(), and Size()

BinarySearchTree()

* + Default constructor that sets root to nullptr

BinarySearchTree::Insert(Bid)

* + Adds a Bid as a new node in the tree

BinarySearchTree::InOrder()

* + Loops through the tree, starting at root, and prints four values from the Bid struct to the console

BinarySearchTree::Remove(String)

* + Searches for a string starting at root and deletes the node if found

BinarySearchTree::Search(String)

* + Searches for a string starting at root and returns the matching node if found

BinarySearchTree::Size()

* + Returns the tree's size

strToDouble

* + Converts CSV file data into usable values

Bid

* + A struct holding bid information

Node

* + A struct holding node data, consisting of a Bid and two pointers (left and right)

loadBids

* + Reads CSV data and adds bids to the BinarySearchTree

main

* + Drives the application with a menu for user input (e.g., load data, view bids, delete a node, exit) and reports algorithm timing using time.h

Main Function

1. Read command-line arguments for the CSV file path (use default if none)
2. Show a menu until the user chooses to exit:
   * For option 1: Load CSV data into the tree and report how long it took
   * For option 2: Call InOrder() to view the bids
   * For option 3: Search for a specific bid and report the search time
   * For option 4: Remove a bid
3. Exit and display "Goodbye"

BinarySearchTree::Insert(Bid)

1. Start at root
2. If root is null, add the Bid as the root node
3. Otherwise, navigate the tree:
   * If the new Bid is smaller than the current node’s, move left
   * If it's larger, move right
   * Insert the new Bid where the correct spot is found
4. Increment size

BinarySearchTree::InOrder(Node)

* Recursively visit the left node, print the bid, then visit the right node

BinarySearchTree::Search(String)

1. Start at root
2. Search for a node with a matching bidId
   * If found, return the node
   * If not, keep searching left or right
3. If not found, return an empty Bid
4. Start at root
5. Search for the node to delete
6. If found:
   * Adjust tree structure
   * Replace the deleted node with its successor if necessary