Dylan Lovin

SNHU CS-300

2/9/2025

**Class BinarySearchTree**

The **BinarySearchTree** class represents a data structure designed for efficient bid storage and retrieval. It consists of:

* **Private members**:
  + size: Tracks the number of elements in the tree.
  + root: Points to the top node of the tree.
  + inOrder(): A private helper method for in-order traversal.
* **Public methods**:
  + BinarySearchTree(): Constructor that initializes an empty tree.
  + ~BinarySearchTree(): Destructor to free allocated memory.
  + InOrder(): Calls the in-order traversal function to display bid data.
  + Insert(Bid): Adds a new bid to the correct position in the tree.
  + Remove(String): Deletes a bid from the tree based on a given identifier.
  + Search(String): Locates a bid in the tree using the given identifier.
  + Size(): Returns the current number of elements in the tree.

**BinarySearchTree()**

This constructor initializes an empty tree by setting the root to nullptr.

**BinarySearchTree::Insert(Bid)**

* Receives a Bid object as input.
* If the tree is empty, the bid becomes the root.
* Otherwise, it finds the appropriate location for insertion:
  + Traverse the tree starting from the root.
  + If the bid ID is smaller than the current node’s ID, navigate left; otherwise, move right.
  + When an available position is found, a new node is created and assigned.
* Increments the size variable to reflect the added bid.

**BinarySearchTree::InOrder()**

* Performs an in-order traversal, beginning at the root.
* Displays four key attributes from each Bid.
* Uses recursion to visit the leftmost node first, then the current node, followed by the right subtree.

**BinarySearchTree::Remove(String)**

* Searches for the node containing the specified bid ID.
* If found, it removes the node while maintaining tree integrity:
  + If the node has no children, it is deleted directly.
  + If it has one child, the parent is linked to the child.
  + If it has two children, the successor (smallest node in the right subtree) is located and used to replace the deleted node.
* The size counter is adjusted accordingly.

**BinarySearchTree::Search(String)**

* Starts at the root and traverses the tree to locate a bid with the given identifier.
* Returns the bid if found; otherwise, an empty bid is returned.

**BinarySearchTree::Size()**

* Provides access to the size attribute.

**Supporting Functions**

* **strToDouble()**: Converts CSV data from string format to a numerical value.
* **Bid struct**: Holds bid-related details, used in sorting and retrieval.
* **Node struct**: Represents a tree node containing a bid and pointers to left and right children.

**loadBids Function**

* Reads bid data from a CSV file.
* Uses command-line arguments to specify the file path, falling back to a default if none is provided.
* Parses CSV data, storing relevant information in Bid structures before inserting them into the BinarySearchTree.

**Main Function Overview**

* Reads command-line arguments to determine the CSV file path.
* Provides a menu-driven interface for user interactions:
  + Load data from CSV.
  + Display bid data in sorted order.
  + Search for a bid.
  + Delete a bid.
  + Exit the application.
* Uses time functions to measure execution duration for operations like loading and searching bids.
* Ensures input validation before executing user-selected operations.

**Pseudocode for Key Operations**

**Main Function**

Read command-line arguments for CSV file path.  
Use default path if no arguments are provided.  
  
While user choice is not ‘9’:  
 Display menu options.  
 Get and validate user input.  
  
 If choice == ‘1’:  
 Start timer.  
 Call loadBids to populate BinarySearchTree.  
 Display the number of records loaded.  
 Stop timer and display elapsed time.  
  
 If choice == ‘2’:  
 Call InOrder() to print bid data.  
  
 If choice == ‘3’:  
 Start timer.  
 Call Search() with user-provided bid ID.  
 Stop timer and display search duration.  
  
 If choice == ‘4’:  
 Call Remove() with user-provided bid ID.  
  
 If choice == ‘9’:  
 Display ‘Goodbye’ and exit.

**BinarySearchTree::Insert(Bid)**

Set currentNode to root.

If root is null:

Create new Node with Bid and assign it to root.

Else:

While currentNode is not null:

If bid ID is smaller than currentNode’s bid ID:

Move left.

If left is null, insert new Node and exit loop.

Else:

Move right.

If right is null, insert new Node and exit loop.

Increment size.

**BinarySearchTree::InOrder(Node)**

If Node is null, return.

Recurse on left child.

Display bid details.

Recurse on right child.

**BinarySearchTree::Search(String)**

Set currentNode to root.

While currentNode is not null:

If currentNode’s bid ID matches String:

Return bid.

If String is smaller:

Move left.

Else:

Move right.

Return empty bid if not found.

**BinarySearchTree::Remove(String)**

Set curr to root, parent to null.

While curr is not null:

If curr’s bid ID matches String:

Handle deletion based on number of children:

No children: Remove node.

One child: Link parent to child.

Two children: Find successor, replace curr with successor, remove successor.

If String is smaller:

Move left.

Else:

Move right.