**CS-300 Communication Binary Search Tree**

Zachary N. Locke

Southern New Hampshire University

Analysis and Design

Jack Lusby

October 19th, 2024

In this assignment, I developed a binary search tree (BST) to handle a collection of bids from CSV files provided by a municipal government data feed. The BST efficiently organizes and searches through thousands of bids. The main operations implemented include inserting bids, removing bids, and searching for a bid by its ID. Each function works recursively to traverse the tree and maintain its structure. In the Insert function, the tree is built by comparing each new bid with existing bids and placing it in the appropriate location, either in the left or right subtree, depending on its bid ID. The Search function traverses the tree to locate a bid, while the Remove function is designed to handle various scenarios, including removing nodes with zero, one, or two children.

The project was challenging in managing edge cases, such as removing a node with two children, which required careful handling to maintain the BST's structure. The code was written to ensure modularity and reusability, with the binary tree logic separated from the rest of the program to allow for future enhancements or reuse. Proper commenting and organization were maintained throughout the code to ensure readability and adherence to best practices in coding style.

START

1. FUNCTION LoadCourses(file):

a. TRY to open the CSV file containing course data:

i. IF file cannot be opened: PRINT "Error: Unable to open file." RETURN or EXIT the program.

b. FOR each line in the CSV file:

i. PARSE the line to extract courseId, courseTitle, and prerequisites.

ii. CREATE a Course object using the parsed data.

iii. CALL Insert(bst, course) to insert the course into the binary search tree.

c. CLOSE the file after all lines are processed.

2. FUNCTION Insert(bst, course):

a. IF root of the binary search tree is nullptr (tree is empty):

i. SET root to a new Node containing the course.

b. ELSE:

i. CALL addNode(root, course) to find the correct location for the new course.

3. FUNCTION addNode(node, course):

a. IF course.courseId is less than node.course.courseId:

i. IF node.left is nullptr: SET node.left to a new Node containing the course.

ii. ELSE: CALL addNode(node.left, course) (recur on left subtree).

c. ELSE:

i. IF node.right is nullptr: SET node.right to a new Node containing the course.

ii. ELSE: CALL addNode(node.right, course) (recur on right subtree).

4. FUNCTION Search(bst, courseId):

a. SET currentNode to root of the binary search tree.

b. WHILE currentNode is not nullptr:

i. IF currentNode.course.courseId matches courseId: RETURN currentNode.course.

ii. ELSE IF courseId is less than currentNode.course.courseId: SET currentNode to currentNode.left.

iii. ELSE: SET currentNode to currentNode.right.

c. RETURN null if no match is found.

5. FUNCTION Remove(bst, courseId):

a. CALL removeNode(root, courseId) to remove the node with the specified courseId from the binary search tree.

6. FUNCTION removeNode(node, courseId):

a. IF node is nullptr: RETURN node.

b. IF courseId is less than node.course.courseId:

i. CALL removeNode(node.left, courseId) (recur on left subtree).

c. ELSE IF courseId is greater than node.course.courseId:

i. CALL removeNode(node.right, courseId) (recur on right subtree).

d. ELSE:

i. IF node has no children: DELETE node. RETURN nullptr.

ii. ELSE IF node has only one child: RETURN node.left IF node.left is not nullptr, OTHERWISE RETURN node.right.

iii. ELSE (node has two children): FIND the minimum node in the right subtree. COPY the course data from that minimum node to node. CALL removeNode(node.right, minNode.courseId) to remove the duplicate.

7. FUNCTION InOrder(node):

a. IF node is not nullptr:

i. CALL InOrder(node.left) (visit left subtree).

ii. PRINT node.course (visit the current node).

iii. CALL InOrder(node.right) (visit right subtree).

8. FUNCTION PreOrder(node):

a. IF node is not nullptr:

i. PRINT node.course (visit the current node).

ii. CALL PreOrder(node.left) (visit left subtree).

iii. CALL PreOrder(node.right) (visit right subtree).

9. FUNCTION PostOrder(node):

a. IF node is not nullptr:

i. CALL PostOrder(node.left) (visit left subtree).

ii. CALL PostOrder(node.right) (visit right subtree).

iii. PRINT node.course (visit the current node).

10. Main Program:

a. DISPLAY menu options to the user:

i. Load Courses from CSV.

ii. Display All Courses (In-Order Traversal).

iii. Find a Course by ID.

iv. Remove a Course by ID.

v. Pre-Order Traversal.

vi. Post-Order Traversal.

vii. Exit.

b. WAIT for user input.

c. PERFORM the selected action by calling the appropriate function (e.g., LoadCourses, InOrder, Search, Remove).

END