Austin Frey

Professor Ling

CS-300

November 27, 2022

BinarySearchTree Code Reflection & Pseudocode

**Code Reflection**

The purpose of the BinarySearchTree program is to expedite search and removal processes. With a worst-case runtime complexity of O(N) and space complexity of O(1) for a removal algorithm, binary search trees are one of the most efficient data structures. However, I did experience some difficulty developing the program because recursion can be challenging for me to debug. In addition, while developing the program, I encountered issues with the remove algorithm. As a result, I had to do additional research, exercises, and visual diagrams to follow what was happening with the program. Eventually, I realized that my algorithm was not accurately modifying nodes on the binary search tree. However, after fixing my mistake, the program worked perfectly.

**Pseudocode**

*Function Insert(Bid bid)*

Pass in: Bid bid

IF root node is null

CREATE new node with bid

ASSIGN new node to root

SET root’s left and right pointers to null

INCREMENT size for binary search tree

ELSE

Call: addNode(root node, bid)

END IF ELSE

Pass out: None

*End function*

*Function addNode(Node node, Bid bid)*

Pass in: node – starting node to search from; bid – the bid to be inserted

CREATE nodeToAdd

IF node’s bidId is larger than nodeToAdd’s bidId

IF node’s left pointer is null

SET node’s left pointer to nodeToAdd

INCREMENT size of binary search tree

ELSE

Call: addNode(node’s left pointer, bid)

END IF ELSE

ELSE

IF node’s right pointer is null

SET node’s right pointer to nodeToAdd

INCREMENT size of binary search three

ELSE

Call: addNode(node’s right pointer, bid)

END IF ELSE

END IF ELSE

Pass out: None

*End function*

*Function removeNode (Node\* &node, String bidId)*

Pass in: node – the node to start searching from; bidId – the key of the node to remove

IF node is null

RETURN

END IF

IF bidId is less than node’s bidId

Call: removeNode(node’s left pointer, bidId)

ELSE IF bidId is greater than node’s bidId

Call: removeNode(node’s right pointer, bidId)

ELSE *(Matching bidId has been found)*

IF node is a leaf (*node’s left and right pointers are null*)

DELETE node

ELSE IF node has two children (*node’s left and right pointers are not null*)

FIND node’s predecessor

MOVE predecessor’s bid to node

Call: removeNode(node’s left pointer, predecessor’s bidId)

ELSE (*node has one child)*

DETERMINE if the child is on the left or right side of node

ASSIGN a new node called child with node’s left/right child node

CREATE temporary Node to hold node

SET node to its child node

DELETE the temporary Node

END IF, ELSE IF, ELSE

END IF, ELSE IF, ELSE

Pass out: None

*End function*

*Function Search(string bidId)*

Pass in: bidId of the desired node

CREATE a Node for tracking (currentNode)

ASSIGN currentNode with the root node

WHILE currentNode is not null

IF currentNode’s bidId matches bidId

RETURN currentNode’s Bid

ELSE IF currentNode’s bidId is greater than bidId

SET currentNode to currentNode’s left child

ELSE

SET currentNode to currentNode’s right child

END IF, ELSE IF, ELSE

END WHILE

RETURN empty Bid

Pass out: Found Bid or empty Bid if no node has a matching bidId

*End function*

*Function inOrder(Node node)* (Print all bids)

Pass in: Node to start at

IF node is null

RETURN

END IF

IF node’s left child is not null

Call: inOrder(node’s left child)

END IF

Print node’s bid

Call: inOrder(node’s right child)

*End function*