**Southern New Hampshire University**

**Joshua Wozny**

[**joshua.wozny@snhu.edu**](mailto:joshua.wozny@snhu.edu)

**January 31, 2023**

**CS-300 Week 5: Binary Search Tree**

**REFLECTION**

This week we implemented a binary search tree data structure in C++. This is the third of several data structures that we will implement in C++ to better understand the advantages and disadvantages of each. Each data structure we will be studying can be used to store data for retrieval later using different techniques.

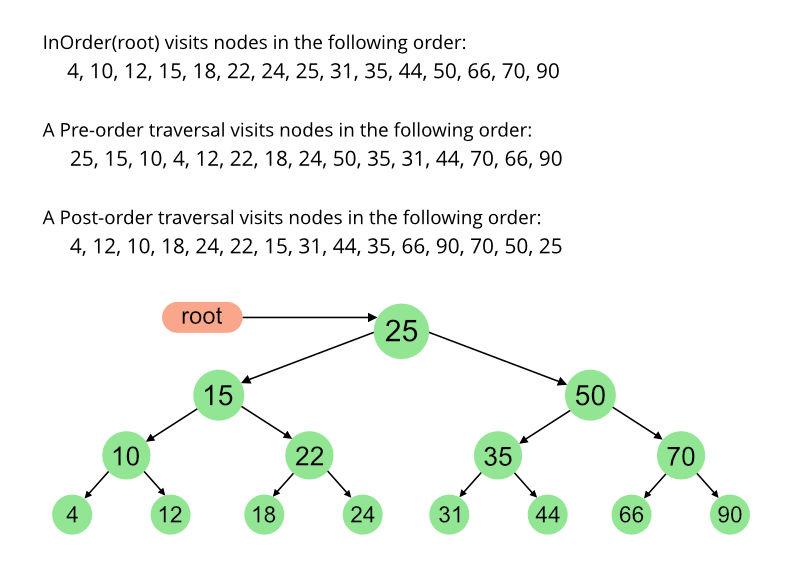
The binary search tree stores data into sorted nodes with each node having pointers to a left node for data object smaller than the current node, and a right node for larger object, for faster storage and retrieval. It is important that the proper selection of the root is used to maintain a balance tree. The nodes of the tree are similar in nature to the nodes of a LinkedList, and BinaryTree could be substitued wherever you might use a LinkedList, if keeping objects in a sorted order is important.

The biggest challenge I had was making sure I was traversing the tree correctly. This was particularly tricky when removing a node - but was initially difficult for inserting nodes as well. Under certain circumstances I would definitely use Binary Search Trees to manage nodes for a HasTable instead of a LinkedList.

**PSEUDOCODE**

**BinarySearchTree**

* Methods needed:
  + BinarySearchTree();
  + virtual ~BinarySearchTree();
  + void InOrder();
  + void PostOrder();
  + void PreOrder();
  + void Insert(Bid bid);
  + void Remove(string bidId);
  + Bid Search(string bidId);
* A node contains bid, pointer to the left child node and one to the right child node
* BinarySearchTree()
  + the root node is initialized to nullptr
* ~BinarySearchTree()
  + start at the root, loop over each node, detach from list then delete



https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/

* Insert(Bid bid)
  + if root equal to null ptr, root = newNode(bid)
  + otherwise, addNode(root, bid)
* addNode(Node\* node, Bid bid)
  + if node is larger then add to left
    - if no left node
      * left = newNode(bid)
    - else recurse down the left node
      * if node is larger then add to left
        + if no left node

left = newNode(bid)

* + else if if no right node
    - right = newNode(bid)
    - else recurse down the left node

if node is larger then add to left

if no left node

left = newNode(bid)

* + - * else if if no right node
        + right = newNode(bid)
* Remove(string bidId)
  + Starting at root, traverse tree until next.bid.bidId == bidId
    - current.right = next.left (1)
    - traverse next.right and add to the tree node(1)
    - set moved nodes to nullptr
* Search(string bidId)
  + set current node equal to root
  + keep looping downwards until bottom reached or matching bidId found
    - if match found, return current bid
      * if bid is smaller than current node then traverse left
      * else traverse left