CSc 3102: Introduction to Binary & Binary Search Trees

Overview of Key Concepts

- Characteristics of Some ADTs
- Terminology
- ADT Binary Search Tree Operations
- Binary Search Tree Representation
- Binary Tree Traversal

1 Characteristics of Some ADTs

The ADTs unordered list, stack, and queue are all **position-oriented**, and their operations have the form:

- Insert a data item at the i^{th} position.
- Delete the data item at the i^{th} position.
- Ask question about the data item at the i^{th} position.

The ADT ordered list is **value-oriented**. Its operations are of the form:

- Insert a data item containing the value x.
- Delete a data item containing the value x.
- Ask a question about a data item containing the value x.

The next two ADTs we will discuss are binary tree ADT and, the more useful, binary search tree ADT. While all the ADTs we have covered thus far are linear, trees are non-linear. The main difference between the binary tree ADT and the binary search tree ADT is that while binary tree ADT is positioned-oriented, binary search tree ADT is value-oriented. We will implement only the binary search tree ADT in this course.

2 Terminology

Definition 1. A tree is a set that is either empty or consists of one or more nodes, partitioned into root node and subsets that are subtrees of the root.

Definition 2. The parent of node n is the node directly above node n in the tree.

Definition 3. A child of node n is a node directly below node n in the tree.

Definition 4. The root is the only node in the tree without a parent.

Definition 5. A leaf is a node without children.

Definition 6. Siblings are nodes with a common parent.

Definition 7. An ancestor of node n is any node on the path from the root to n.

Definition 8. A descendant of node n is a node on a path from n to a leaf.

Definition 9. A subtree of node n is a tree that consists of a child (if any) of n and the child's descendants.

Definition 10. The height of a tree is the number of edges on the longest path from the root to a leaf. An empty tree has a height of -1; a tree with only one node has a height of 0.

Definition 11. A binary tree is a set of nodes that is either empty or partitioned into a root node and one or two subsets that are binary subtrees of the root. Each node has at most two children, the left child and the right child.

Definition 12. The left(right) child of node n is a node directly below and to the left(right) of node n in a binary tree.

Definition 13. The left(right) subtree of node n in a binary tree is the left(right) child (if any) of node n plus its descendants.

Definition 14. A binary search tree is a binary tree where the value in any node n is greater than the value in every node in n's left subtree, but less than the value of every node in n's right subtree.

Definition 15. A (binary) tree without any node is said to be an *empty* (binary) tree.

Definition 16. A perfect binary tree is a binary tree of height h with no missing nodes. All leaves are at level n and all other nodes each have two children.

Definition 17. A full binary tree is a binary tree in which each node has 0 or two children.

Definition 18. A complete binary tree is a binary tree of height h that is perfect to level $h-1$ and has level h filled in from left to right.

Definition 19. A balanced binary tree is a binary tree in which the left and right subtrees of any node have heights that differ by at most 1.

3 ADT Binary Search Tree Operations

Definition 20. The ADT binary search tree is a binary tree in which the value in any node n is greater than the value in every node in n's left subtree, but less than the value of every node in n's right subtree along with the following operations:

1. Create an empty binary search tree.
2. Destroy a binary search tree.
3. Determine whether a binary search tree is empty.
4. Insert a new item into a binary search tree.
5. Delete the item with a given search key from the binary search tree.
6. Traverse the items in a binary search tree in preorder, inorder, or postorder.

4 Binary Search Tree Representation

Although binary search trees may be implemented using arrays, it is typically implemented using pointers. Here is pseudocode for a representation of an extensible binary search tree with relevant metadata elements.

```
type BSTree
------------------------------------------------
Integer cmp {a trichotomous comparator}
Integer size
Node Ref root {reference to the root node}
------------------------------------------------
type Node
------------------------------------------------------
Node Ref left {reference to the left child node}
Node Ref right {reference to the right child node}
TreeItemType data {data must be comparable}
------------------------------------------------------
```

Here is a diagram showing a binary search tree with 1 node, 3 nodes and an empty binary tree.

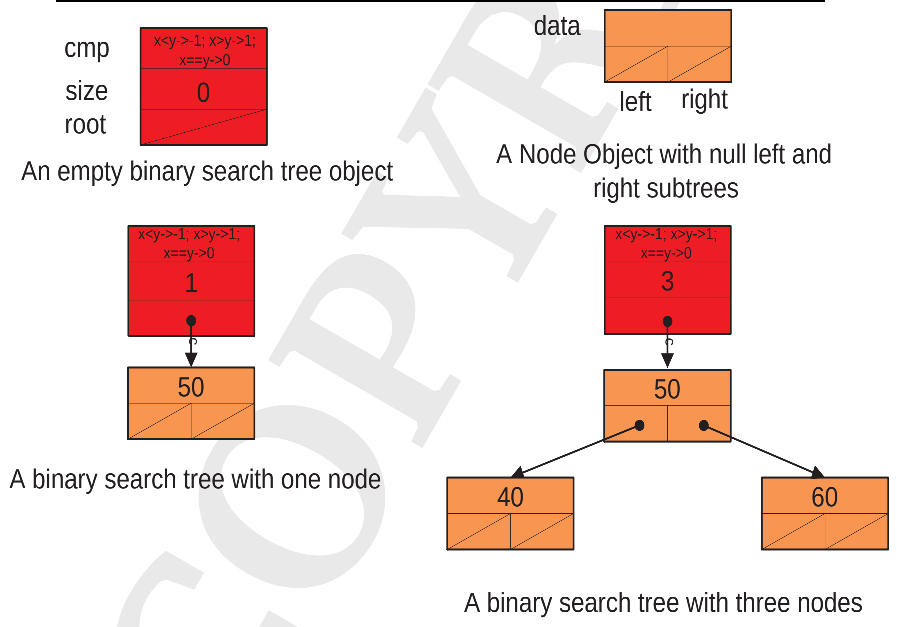
cmp
size
root
x<y->-1; x>y->1; x==y->0
0
An empty binary search tree object
data
left
right
A Node Object with null left and right subtrees
x<y->-1; x>y->1; x==y->0
1
50
A binary search tree with one node
x<y->-1; x>y->1; x==y->0
3
50
40
60
A binary search tree with three nodes

Figure 1: A Visual Representation of a Binary Search Tree

5 Binary Tree Traversal

There are three major binary tree traversal algorithms:

5.1 Preorder Traversal

```
if (T is not empty)
{
   visit the root of T
   preorder(LeftChild(T))
   preorder(RightChild(T))
}
```

5.2 Inorder Traversal

```
if (T is not empty)
{
   inorder(LeftChild(T))
   visit the root of T
   inorder(RightChild(T))
}
```

5.3 Postorder Traversal

```
if (T is not empty)
{
   postorder(LeftChild(T))
   postorder(RightChild(T))
   visit the root of T
}
```

Problem 1. Show the binary search tree before and after each of the following data is inserted into it. 92 24 6 7 11 8 22 4 5 16 19 20 78

Problem 2. Give the preorder and postorder traversals of the tree from Problem 1 after the insertion of the last key.