**DHA Suffa University**



**Department of Computer Science**

**CS 2001L – Data Structures and Algorithms Lab**

**Fall 2019**

**Lab 10 – Binary Search Trees**

**Objective:**

Learning about implementation of Binary Search Trees.

# **Binary Search Tree**

# Binary Search Tree is a node-based binary tree data structure which has the following properties:

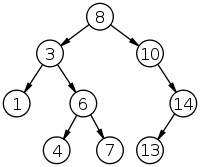


Figure 10.1 : Binary search tree

# The left subtree of a node contains only nodes with data less than the node’s data.

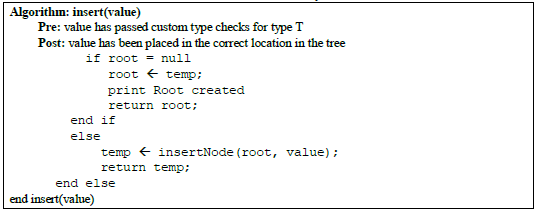
# The right subtree of a node contains only nodes with data greater than the node’s data.

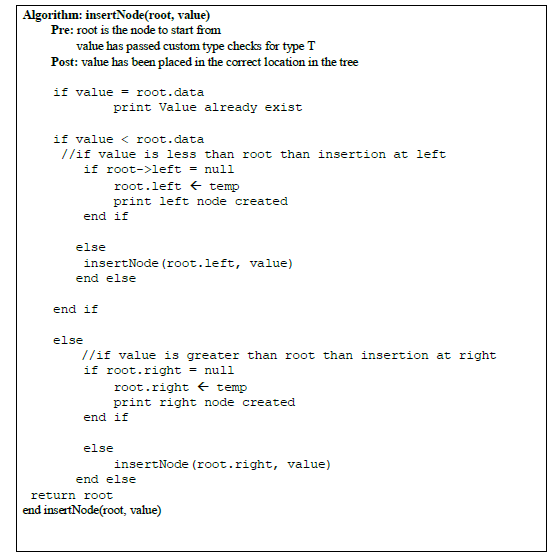
# The left and right subtree each must also be a binary search tree

Main applications of trees include:

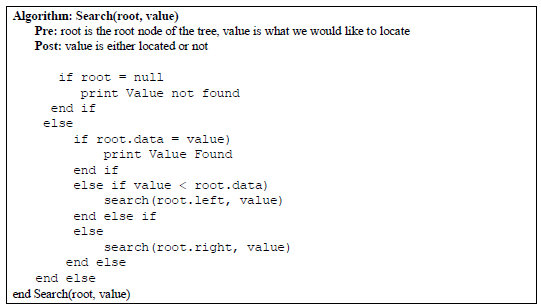
* Manipulate hierarchical data.
* Make information easy to search (see tree traversal).
* Manipulate sorted lists of data.
* As a workflow for compositing digital images for visual effects.
* Router algorithms

**Algorithm for Insertion:**

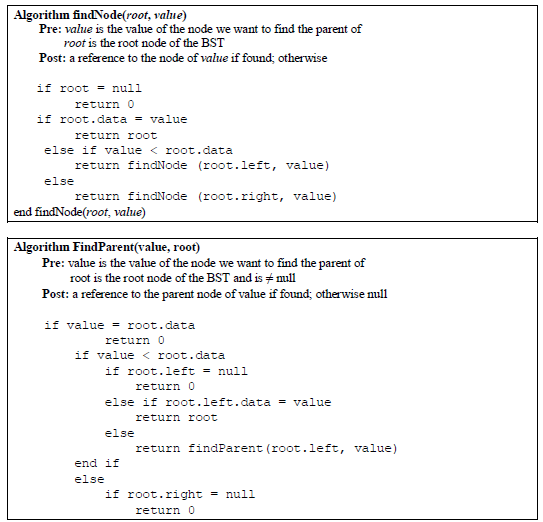
****

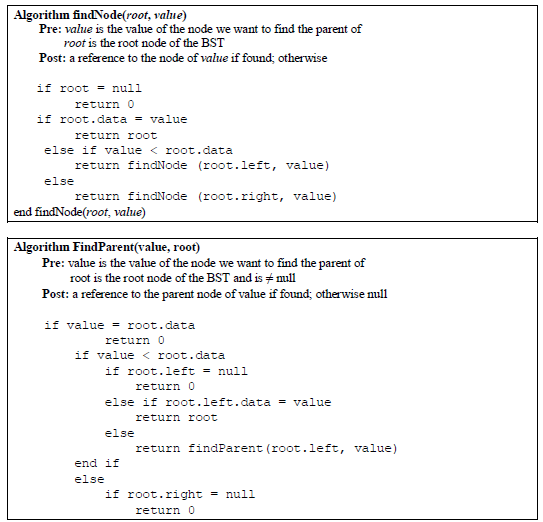
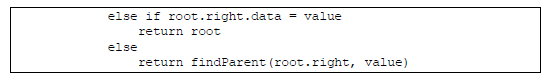
****

**Algorithm for Searching:**

****

**Algorithm for Deletion:**

****

** **

Removing a node from a BST is fairly straightforward, with four cases to consider:

1. The value to remove is a leaf node; or

2. The value to remove has a right subtree, but no left subtree; or

3. The value to remove has a left subtree, but no right subtree; or

4. The value to remove has both a left and right subtree in which case we promote the largest

value in the left subtree.

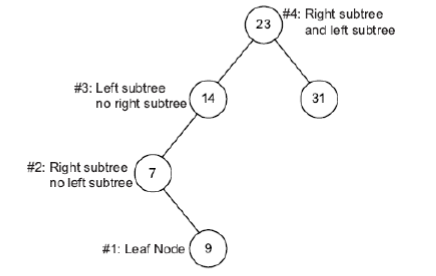
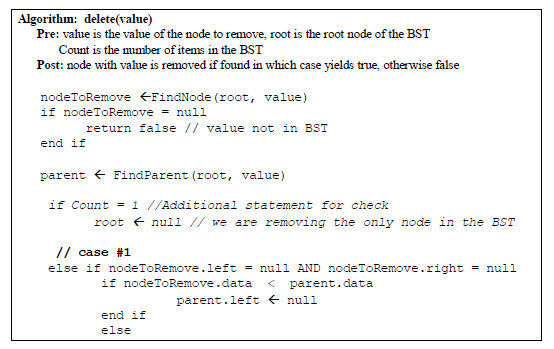
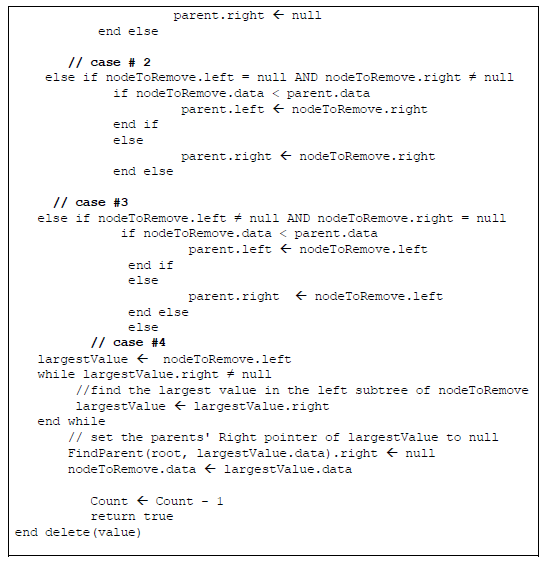
****

Figure 10.2 : Deletion in Binary search tree

**Algorithm: Delete(root, value)**

****

****

Tree Traversals (Inorder, Preorder and Postorder)

# Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.

# **Uses of Inorder** In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order.

|  |
| --- |
| **Algorithm:**Inorder(root)  **Pre:** root is the root node of the tree, value is what we would like to locate  **Post:** value is either located or not  inorderPrint( TreeNode \*root )  if ( root != NULL )   inorderPrint( root->left )   Print ‘root->item’  inorderPrint( root->right )  **endAlgorithm** Inorder(root) |

**Uses of Preorder**

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on of an expression tree

|  |
| --- |
| **Algorithm:**Preorder(root)  **Pre:** root is the root node of the tree, value is what we would like to locate  **Post:** value is either located or not  preorderPrint( TreeNode \*root )  if ( root != NULL )  Print ‘root->item’   preorderPrint( root->left )   preorderPrint( root->right )   end if  **endAlgorithm**Preorder(root) |

**Uses of Postorder**

Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.

|  |
| --- |
| **Algorithm:**Postorder(root)  **Pre:** root is the root node of the tree, value is what we would like to locate  **Post:** value is either located or not  postorderPrint( TreeNode \*root )  if ( root != NULL )  postorderPrint( root->left )  postorderPrint( root->right )  Print ‘root->item’  **endAlgorithm**Postorder(root) |

# Example Tree

Figure 10.3 : Traversal of Binary Tree

# **Following evaluation of Inorder,Preorder,postorder**

# **(a) Inorder (Left, Root, Right) :** 4 2 5 1 3

# **(b) Preorder (Root, Left, Right) :** 1 2 4 5 3

# **(c) Postorder (Left, Right, Root) :**4 5 2 3 1

# **Assignment**

Q.1) Create a BST with the following values 21, 16, 2, 25, 30, 14, 2, 60, 8, 15, 35, 40, 100, 55.

a) Write a function to find the height of the tree.  
b) Write a function that will print the RST of the BST only in preorder, postorder and inorder

Q.2) Create an identical tree as created in Q.1.

a) Write a function that will take both trees as argument and create a third final tree having nodes values equal to the sum value of nodes of the previous two trees.