PRESIDENCY UNIVERSITY

GAIN MORE KNOWLEDGE
REACH GREATER HEIGHTS

CSE 2001 - Data Structures and Algorithms

Module 3 : Trees

Java program to implement Binary Trees

```
//class to create nodes
class Node {
int key;
Node left, right;

public Node(int item) {
key = item;
left = right = null;
}
}

public class BinaryTree
{
    Node root;

      // Traverse tree
      public void traverseTree(Node node) {
        if (node != null) {
          traverseTree(node.left);
          System.out.print(" " + node.key);
          traverseTree(node.right);
        }
      }

      public static void main(String[] args) {

        // create an object of BinaryTree
        BinaryTree tree = new BinaryTree();

        // create nodes of the tree
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);

        System.out.print("\nBinary Tree: ");
        tree.traverseTree(tree.root);
      }
}
```

Java program to implement Binary Search Trees

```
//Binary Search Tree operations in Java

class BST {
class Node {
 int key;
 Node left, right;

 public Node(int item) {
   key = item;
   left = right = null;
 }
}

Node root;

BST() {
 root = null;
}

void insert(int key) {
 root = insertKey(root, key);
}

// Insert key in the tree
Node insertKey(Node root, int key) {
 // Return a new node if the tree is empty
 if (root == null) {
   root = new Node(key);
   return root;
 }
```

```java
 // Traverse to the right place and insert the node
 if (key < root.key)
   root.left = insertKey(root.left, key);
 else if (key > root.key)
   root.right = insertKey(root.right, key);

 return root;
}

void inorder() {
 inorderRec(root);
}

// Inorder Traversal
void inorderRec(Node root) {
 if (root != null) {
   inorderRec(root.left);
   System.out.print(root.key + " -> ");
   inorderRec(root.right);
 }
}

void deleteKey(int key) {
 root = deleteRec(root, key);
}

Node deleteRec(Node root, int key) {
 // Return if the tree is empty
 if (root == null)
   return root;

 // Find the node to be deleted
 if (key < root.key)
   root.left = deleteRec(root.left, key);
 else if (key > root.key)
   root.right = deleteRec(root.right, key);
 else {
   // If the node is with only one child or no child
   if (root.left == null)
     return root.right;
   else if (root.right == null)
     return root.left;

   // If the node has two children
   // Place the inorder successor in position of the
node to be deleted
   root.key = minValue(root.right);

   // Delete the inorder successor
   root.right = deleteRec(root.right, root.key);
 }

 return root;
}

// Find the inorder successor
int minValue(Node root) {
 int minv = root.key;
 while (root.left != null) {
   minv = root.left.key;
   root = root.left;
 }
 return minv;
}

// Driver Program to test above functions
public static void main(String[] args) {
 BST tree = new BST();

 tree.insert(8);
 tree.insert(3);
 tree.insert(1);
 tree.insert(6);
 tree.insert(7);
 tree.insert(10);
 tree.insert(14);
 tree.insert(4);

 System.out.print("Inorder traversal: ");
 tree.inorder();

 System.out.println("\n\nAfter deleting 10");
 tree.deleteKey(6);
 System.out.print("Inorder traversal: ");
 tree.inorder();
}
}
```

Java program to implement Binary Search Trees traversals

```
class Node {
int item;
Node left, right;

public Node(int key) {
item = key;
left = right = null;
}
}

class BST_Traversal {
// Root of Binary Tree
Node root;

BST_Traversal() {
root = null;
}

void postorder(Node node) {
if (node == null)
 return;

// Traverse left
postorder(node.left);
// Traverse right
postorder(node.right);
// Traverse root
System.out.print(node.item + "->");
}

void inorder(Node node)
{
if (node == null)
 return;

// Traverse left
inorder(node.left);
// Traverse root
System.out.print(node.item + "->");
// Traverse right
inorder(node.right);
}

void preorder(Node node) {
if (node == null)
 return;

// Traverse root
System.out.print(node.item + "->");
// Traverse left
preorder(node.left);
// Traverse right
preorder(node.right);
}

public static void main(String[] args) {
    BST_Traversal tree = new BST_Traversal();
tree.root = new Node(1);
tree.root.left = new Node(12);
tree.root.right = new Node(9);
tree.root.left.left = new Node(5);
tree.root.left.right = new Node(6);

System.out.println("Inorder traversal");
tree.inorder(tree.root);

System.out.println("\nPreorder traversal ");
tree.preorder(tree.root);

System.out.println("\nPostorder traversal");
tree.postorder(tree.root);
}
}
```