Department of Software Engineering
Mehran University of Engineering and Technology, Jamshoro

Course: SWE224 – Data Structure & Algorithms			
Instructor	Mariam Memon	**Practical/Lab No.**	12 & 13
Date		**CLOs**	3
Signature		**Assessment Score**	02

Topic	**Implementation of Tress & Binary Tree Traversals**
Objectives	Implement the Tree data structure in java and explore various binary tree traversal algorithms

Lab Discussion: Theoretical concepts and Procedural steps

Tree Data Structure

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges. Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size. But, it is not acceptable in today's computational world.

Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.

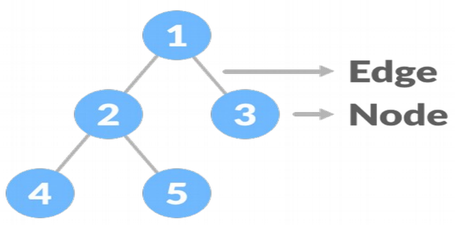
Tree Terminologies

Node

A node is an entity that contains a key or value and pointers to its child nodes.
The last nodes of each path are called leaf nodes or external nodes that do not contain a link/pointer to child nodes.
The node having at least a child node is called an internal node.

Edge

It is the link between any two nodes.



Root

It is the topmost node of a tree.

Height of a Node

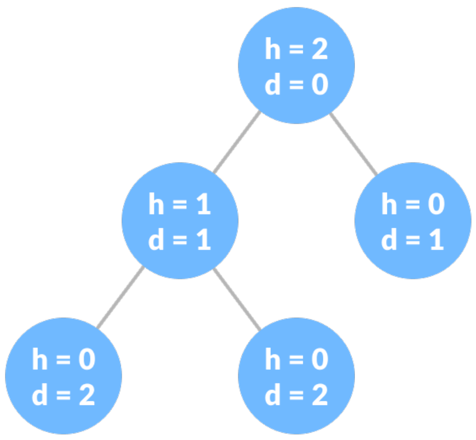
The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

Depth of a Node

The depth of a node is the number of edges from the root to the node.

Height of a Tree

The height of a Tree is the height of the root node or the depth of the deepest node.



Degree of a Node

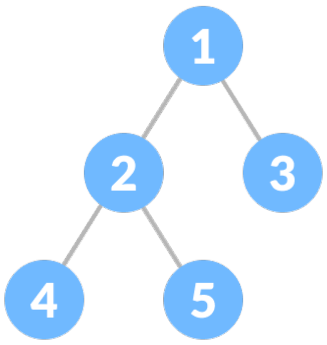
The degree of a node is the total number of branches of that node.

<u>Tree Applications</u>

- Binary Search Trees (BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

Binary Trees

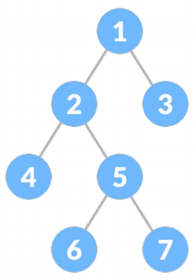
A binary tree is a tree data structure in which each parent node can have at most two children. For example,



Types of Binary Tree

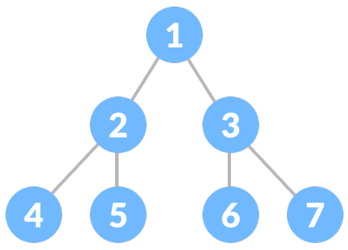
Full Binary Tree

A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.



Perfect Binary Tree

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.



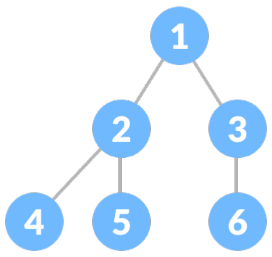
Complete Binary Tree

A complete binary tree is just like a full binary tree, but with two major differences.

Every level must be completely filled

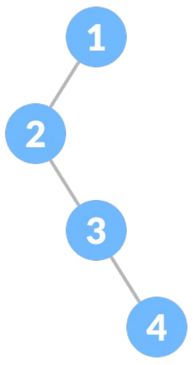
All the leaf elements must lean towards the left.

The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.



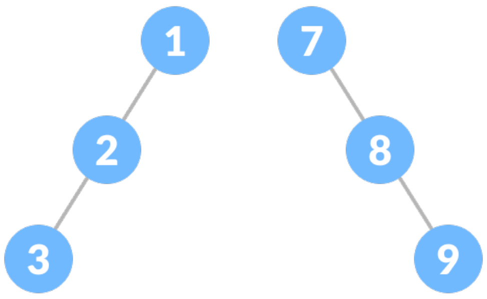
Degenerate or Pathological Tree

A degenerate or pathological tree is the tree having a single child either left or right.



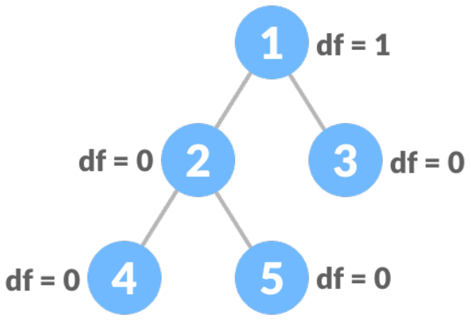
Skewed Binary Tree

A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree.



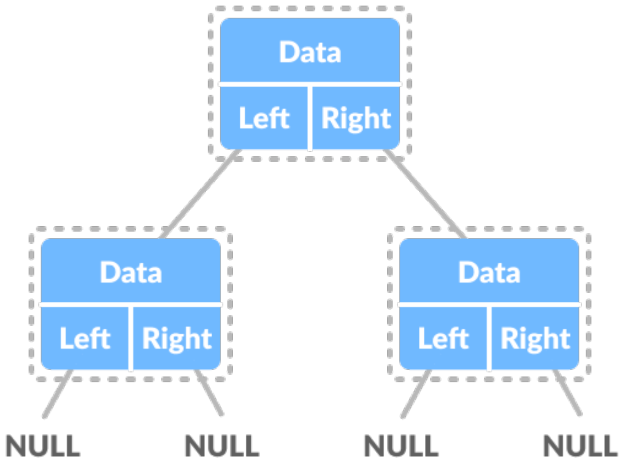
Balanced Binary Tree

It is a type of binary tree in which the difference between the height of the left and the right subtree for each node is either 0 or 1.



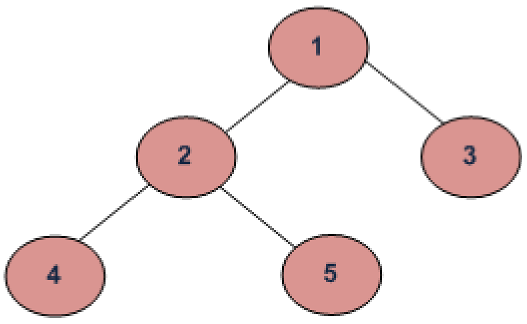
<u>Binary Tree Representation</u>

A node of a binary tree is represented by a structure containing a data part and two pointers to other structures of the same type.



Tree Traversals

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.



- **Depth First Traversals:**

  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

- **Breadth First or Level Order Traversal**: 1 2 3 4 5

Depth First Traversal Algorithms

a) **Inorder Traversal Algorithm**

Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

b) **Preorder Traversal Algorithm**

Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

c) **Postorder Traversal Algorithm**

Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

Breadth First Traversal / Level Order Traversal Algorithm

```
printLevelorder(tree)
• Create an empty queue q
• temp_node = root /*start from root*/
• Loop while temp_node is not NULL
    • print temp_node->data.
    • Enqueue temp_node’s children (first add left then right
      children) to q
    • Dequeue a node from q.
```

<u>Implementation Code for Binary Tree and its traversal</u>

```
import java.util.Queue;
import java.util.LinkedList;

class Node {
    int data;
    Node left, right;

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

class BinaryTree {
    Node root;

    BinaryTree(int data) {
      root = new Node(data);
    }

    BinaryTree() {
      root = null;
    }

    // Traverse Inorder
    public void traverseInOrder(Node node) {
      if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
        traverseInOrder(node.right);
      }
    }

    // Traverse Postorder
    public void traversePostOrder(Node node) {
```

```
    if (node != null) {
      traversePostOrder(node.left);
      traversePostOrder(node.right);
      System.out.print(" " + node.data);
    }
  }

  // Traverse Preorder
  public void traversePreOrder(Node node) {
    if (node != null) {
      System.out.print(" " + node.data);
      traversePreOrder(node.left);
      traversePreOrder(node.right);
    }
  }

  public void printLevelOrder(Node node)
  {
    Queue<Node> queue = new LinkedList<Node>();
    queue.add(node);
    while (!queue.isEmpty())
    {

      // poll() removes the present head.
      Node tempNode = queue.poll();
      System.out.print(tempNode.data + " ");

      /*Enqueue left child */
      if (tempNode.left != null) {
        queue.add(tempNode.left);
      }

      /*Enqueue right child */
      if (tempNode.right != null) {
        queue.add(tempNode.right);
      }
    }
  }

  public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();

    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("Pre order Traversal: ");
    tree.traversePreOrder(tree.root);
```

```
        System.out.print("\nIn order Traversal: ");
        tree.traverseInOrder(tree.root);
        System.out.print("\nPost order Traversal: ");
        tree.traversePostOrder(tree.root);
        System.out.print("\nlevel order Traversal: ");
        tree.printLevelOrder(tree.root);
    }
}
```

Lab Tasks

Task 1: Implement the binary tree with following fields:
Object root; BinaryTree left, BinaryTree right;

- Create the following overloaded constructors:
    - BinaryTree(Object root);
    - BinaryTree(Object root, BinaryTree left, BinaryTree right);
- Create getter & setter methods for root, left and right.

Task 2: Add the following methods to your class:

1. public String toString()
2. public boolean isLeaf()
3. public boolean isFull()
4. public boolean isComplete()
5. public boolean isBalanced()
6. public int size()
7. public int height()
8. public int degree()
9. public boolean contains(Object object)
10. public int numOfLeaves()
11. public int level(Object object)

Task 3: Implement all the traversal algorithms in your class.

Lab Rubrics for Evaluation

Rubrics	**Proficient**	**Adequate**	**Poor**
Programming Algorithms	Completely accurate and efficient design of algorithms **(0.8)**	Accurate but inefficient algorithms **(0.4)**	Unable to design algorithms for the given problems **(0.0)**
Originality	Submitted work shows large amount of original	Submitted work shows some amount of original	Submitted work shows no original thought

	thought **(0.6)**	thought **(0.4)**	**(0.0)**
Troubleshooting	Easily traced and corrected faults **(0.6)**	Traced and corrected faults with some guidance **(0.4)**	No troubleshooting skills **(0.0)**