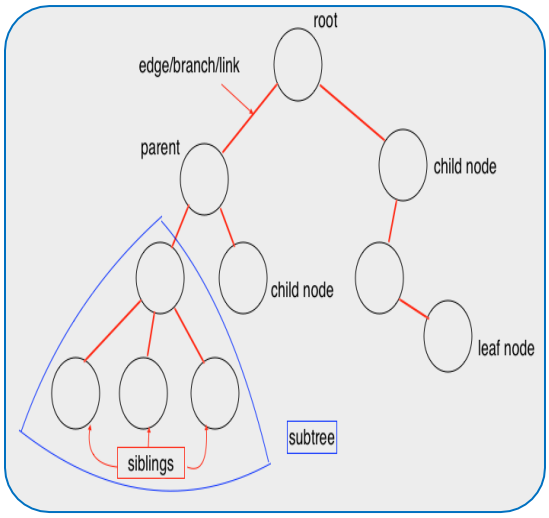
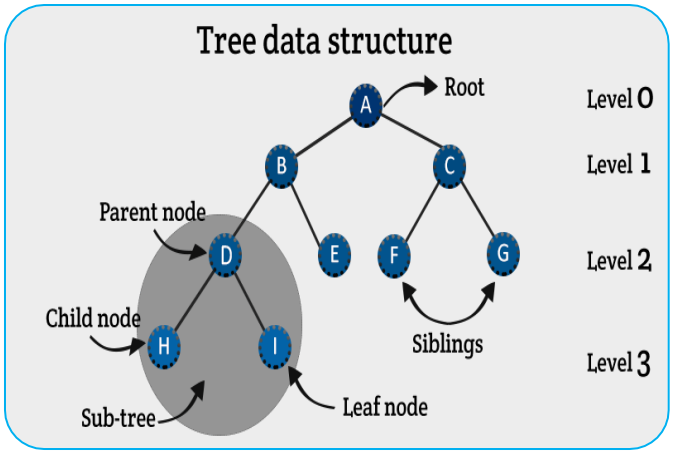
TREE DATA STRUCTURE (LECTURE 4)

Tree Data Structure

- A tree consist of nodes connected by edges, the nodes represent the data items and the edges represent the way the nodes are related
- A tree with nodes which has maximum of two children (0, 1 or 2) is called a binary tree



Binary Tree



- The node at the top of the tree is called root (mention the root when drawing a tree)
- Any node which has exactly one edge running upwards to other node is called a child
- The two children of each node in a binary tree are called left child and right child
- Any node which has one or more lines running downwards to other nodes is called a parent
- A node that has no children is called a leaf node or leaf
- Sequence of nodes from one node to another along the edges is called a path
- Any node which consist of its children and it's children's children and so on is called a sub tree
- Each node in a tree stores objects containing information
- Therefore one data item is usually designated as a key value
- The key value is used to search for an item or to perform other operations on it

Binary Search Tree

- A tree that has at most two children (binary tree)
- A node's left child must have a key less than its parent and node's right child must have a key greater than or equal to its parent

Binary Search Tree Operations/ Functions

- **find()**
  - Find always start at the root
  - Compare the key value with the value at root
  - If the key value is less, then compare with the value at the left child of root
  - If the key value is higher, then compare with the value at the right child of root
  - Repeat this, until the key value is found or reach to a leaf node

- **insert()**
  - Create a new node
  - Find the place (parent) to insert a new node
  - When the parent is found, the new node is connected as its left or right child, depending on whether the new node's key is less than or greater than that of the parent

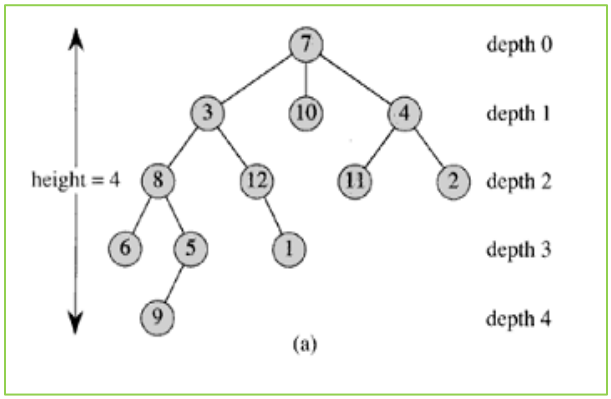
- **delete()**
  - Delete a node with the specified value
  - First find the node to be deleted
  - If the node to be deleted is found there are three cases to be considered. Whether,
    1. The node to be deleted is a leaf
    2. The node to be deleted has one child
    3. The node to be deleted has two children

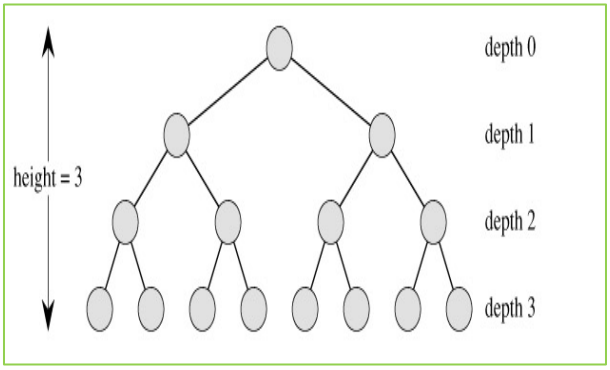
❖ **traverse()**
- Traverse a tree means to visit all the nodes in some specified order
- There are three common ways to traverse a tree

1. Pre order
   - Visit the node
   - Call itself to traverse the node's left subtree
   - Call itself to traverse the node's right subtree
2. In order
   - Call itself to traverse the node's left subtree
   - Visit the node
   - Call itself to traverse the node's right subtree
3. Post order
   - Call itself to traverse the node's left subtree
   - Call itself to traverse the node's right subtree
   - Visit the node

Tree Terminology

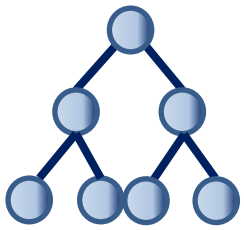
Degree of a node: The number of children it has
Depth: The depth of x in a tree is the length of the path from the root to a node x
Height: The largest depth of any node in a tree





Full Binary Tree

❖ A Full binary tree of height *h* contains exactly $2^{h+1} - 1$ nodes
❖ Height, $h = 2$, nodes = $2^{2+1} - 1 = 7$



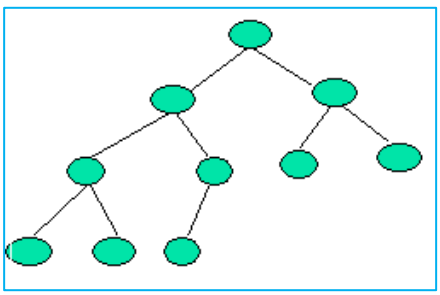
Complete Binary Tree

❖ It is a Binary tree where each node is either a leaf or has degree ≤ **2**
❖ Completely filled, except possibly for the bottom level
❖ Each level is filled from **left to right**
❖ All nodes at the lowest level are as far to the left as possible
❖ Full binary tree is also a complete binary tree
❖ All FBTs are CBTs, but, all CBTs are not FBTs.

Height of a complete binary tree

Height of a complete binary tree that contains *n* elements is $\log_2 n$

Ex:
This is a Complete Binary Tree with height = 3
No of nodes: $n = 10$
Height = $\log_2 n = \log_2 10 = 3$
(Take floor value)



TREE DATA STRUCTURE (LAB 4)

Question

a) Implement a Node class with suitable attributes to store employee number and name of employees
b) Implement displayNode() method to display the details stored in a Node
c) Implement the Tree class with the following data members and methods

Tree
Node root
Node find(int emp) void insert(in emp, String name) void inOrder() void preOrder() void postOrder() Node findRecursive() void deleteAll()

d) Implement a new method called findRecursive(int emp) which perform the find operation recursively
e) Implement a method called deleteAll() to remove all the Nodes from the tree
f) Write an application to do the following
   i) Create a tree of 10 Nodes with the following details

Employee Number	Name
149	Anusha
167	Kosala
047	Dinusha
066	Mihiri
159	Jayani
118	Nimal
195	Nishantha
034	Ayodya
105	Bimali
133	Sampath

   ii) Display the employee data using inorder, preorder and postorder traversing
   iii) Allow the user to input any employee number from the keyboard and display the employee details if the employee exists in the tree
   iv) Delete all the nodes from the binary search tree
   v) Display the tree after deleting nodes

Tree implementation

❖ Node Class

- Node contains the information about an object
- Each node should have a key, data and reference to left and right child

(Lab sheet 4 answer)

```
public class Node {
	public int emp;
	public String name;
	public Node leftChild, rightChild;
	public Node(int emp, String name) {
		this.emp = emp;
		this.name = name;
		leftChild = null;
		rightChild = null;
	}
	public void display() {
		System.out.print("[" + emp + ", " + name + "]");
		System.out.println(" ");
	}
}
```

❖ Tree Class

```
public class Tree {
	public Node root;

	public Tree() {
		this.root = null;
	}
```

```
    public Node find(int emp) {
         root = findRecursive(root, emp);
         return root;
    }//find

    public Node findRecursive(Node root, int emp) {
           // Base Cases: root is null or key is present at root
        if (root==null || root.emp==emp)
            return root;
        // value is greater than root's key
        if (root.emp > emp)
            return findRecursive(root.leftChild, emp);
        // value is less than root's key
        return findRecursive(root.rightChild, emp);
    }//findRecursive

    public void insert(int emp, String name) {
           root = insertRecursive(root, emp, name);
    }//insert

    public Node insertRecursive(Node root, int emp, String name) {
           // If the tree is empty, return a new node
        if (root == null) {
           root = new Node(emp, name);
            return root;
        }
        // Otherwise, recur down the tree
        if (emp < root.emp)
            root.leftChild = insertRecursive(root.leftChild, emp, name);
        else if (emp > root.emp)
            root.rightChild = insertRecursive(root.rightChild, emp, name);
        // return the (unchanged) node pointer
        return root;
    }//insertRecursive

    public void inOrder(Node root) {
           if (root != null) {
                  inOrder(root.leftChild);
            root.display();
            inOrder(root.rightChild);
        }
    }//inOrder

    public void preOrder(Node root) {
      if (root != null) {
          root.display();
           preOrder(root.leftChild);
          preOrder(root.rightChild);
      }
   }//preOrder

    public void postOrder(Node root) {
      if (root != null) {
           preOrder(root.leftChild);
          preOrder(root.rightChild);
          root.display();
      }
   }//postOrder

   void deleteAll(Node node) {
      // In Java automatic garbage collection happens, so make root null to delete the tree
       root = null;
   }//deleteAll
}
```

❖ Application class

```
import java.util.Scanner;

public class Application {

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

		tree.insert(149, "Anusha");
		tree.insert(167, "Kosala");
		tree.insert(47, "Dinusha");
		tree.insert(66, "Mihiri");
		tree.insert(159, "Jayani");
		tree.insert(118, "Nimal");
		tree.insert(195, "Nishantha");
		tree.insert(34, "Ayodya");
		tree.insert(105, "Bimali");
		tree.insert(133, "Sampath");

		tree.inOrder(tree.root);
		tree.preOrder(tree.root);
		tree.postOrder(tree.root);

		Scanner sc = new Scanner(System.in);
		System.out.println("Enter employee number");
		int emp = sc.nextInt();

		Node crnt = tree.find(emp);
		if (crnt != null) {
			tree.find(emp).display();
		} else {
			System.out.println("Employee number " + emp + " does not exist in the tree");
		}

		tree.deleteAll(tree.root);
		System.out.println("Tree values: " + tree.root);

		}

	}
```

TREE DATA STRUCTURE (TUTORIAL 4)

Question 1

Arrange the following sequence of integers into a binary search tree

280 308 180 416 298 350 156 255 580 275 12

Question 2

Print the elements in the tree built in Question1 using the following traversing methods.

a) inorder b) preorder c) postorder

Question 3

Consider the Node class and Tree class given below.

Node	Tree
int iData double dData Node leftChild Node rightChild	Node root
Void displayNode	Node find(int key) void insert(in id, double dd) boolean delete(int id) void desOrder() Node minimum()

i) Implement a method called minimum( ) to find the minimum node in a tree

ii) Implement a method called descOrder() to display the values in the tree in descending order

Question 4

Draw the tree structures for the binary tree created in Question 1 for each of the following delete commands

a) Delete(255) b) Delete(308) c) Delete (180) d) Delete(280)

Additional Exercises

Question 1

Write a java program to implement the following

a) Implement a Node class to store a height of a child. In the same class implement displayNode ( ) method to display the data stored in a Node

b) Implement the Tree class with the following data members and methods

Tree
Node root
void insert(Char ch) Node minimum() Node maximum() void descendingOrder()

c) In your application, enter the height of 10 children in a class from the key board and store them in a tree. Use the above implemented methods to display the height of the tallest child, shortest child in the class. Also display the height of all ten children in descending order

```
  int minValue(Node root) {
    int minValue = root.emp;
    while (root.leftChild != null) {
         minValue = root.leftChild.emp;
        root = root.leftChild;
    }
    return minValue;
  }//minValue

  int maxValue(Node root) {
    int maxValue = root.emp;
    while (root.rightChild != null) {
         maxValue = root.rightChild.emp;
        root = root.rightChild;
    }
    return maxValue;
  }//maxValue

/descending? delete key?
```