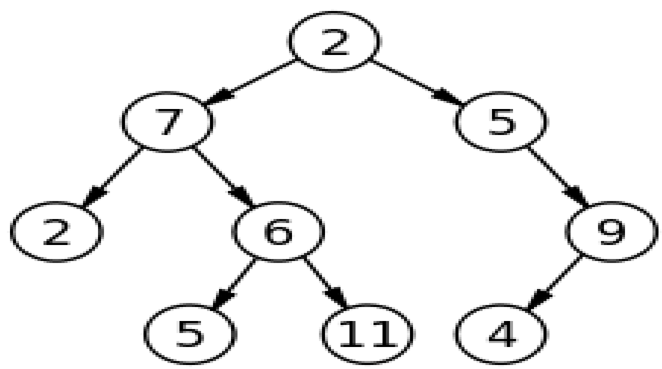
Tree Data structure

One of the most striking and widely used feature in data structures is Tree. In this note you are going learn about tree. And I am sure that by the end of the tutorial you will be able to clearly figure out the concepts of trees and I will discuss some of the classical problems on treesSo lets start with our discussion on trees.

How typically a tree looks like in data structure. **Here is a sample**-



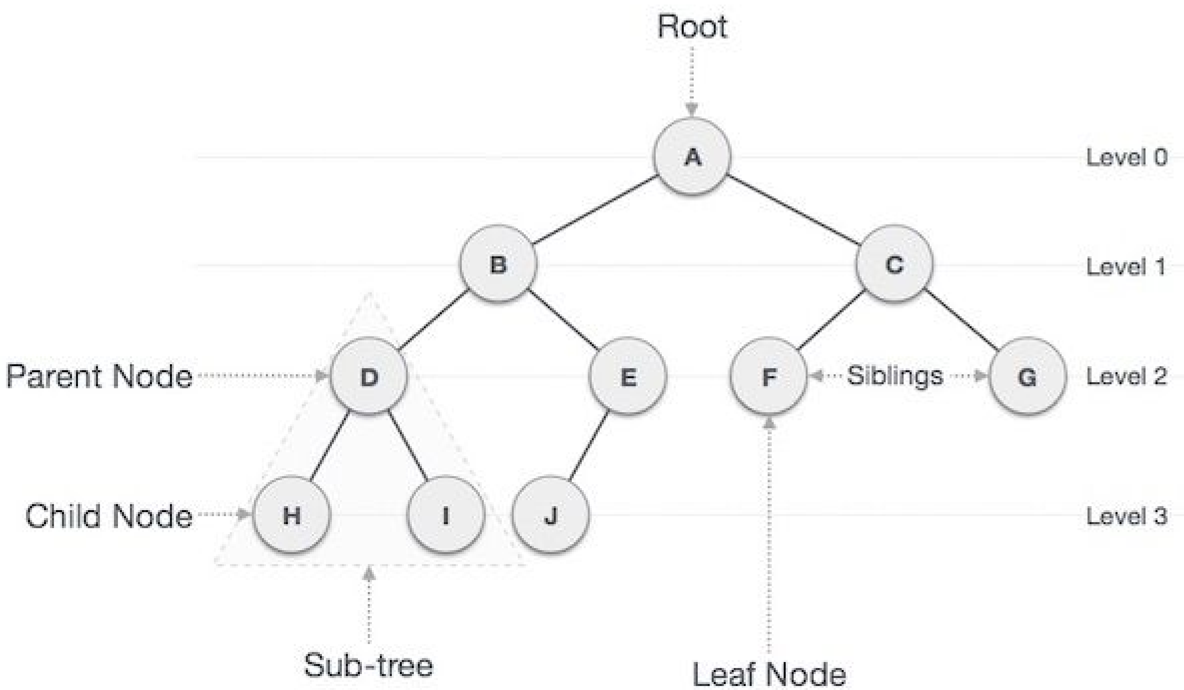
Important Terms

Following are the important terms with respect to tree.

- **Path** − Path refers to the sequence of nodes along the edges of a tree.

- **Root** – The node at the top of the tree is called root. There is only one root per tree and one path from the root node to any node.
- **Parent** – Any node except the root node has one edge upward to a node called parent.
- **Child** – The node below a given node connected by its edge downward is called its child node.
- **Leaf** – The node which does not have any child node is called the leaf node.
- **Subtree** – Subtree represents the descendants of a node.
- **Visiting** – Visiting refers to checking the value of a node when control is on the node.
- **Traversing** – Traversing means passing through nodes in a specific order.
- **Levels** – Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.
- **keys** – Key represents a value of a node based on which a search operation is to be carried out for a node.
-

You can see in the diagram……



Main applications of trees include:

1. Manipulate hierarchical data.
2. Make information easy to search (see tree traversal).
3. Manipulate sorted lists of data.
4. As a workflow for compositing digital images for visual effects.
5. Router algorithms
6. Form of a multi-stage decision-making (see business chess).

Binary Tree: A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

In the above tree you can see that it is also **binary tree**…..

Binary Tree Representation in C:

```
Struct tree
{
int data;            //the element
Struct node   *left.      //pointer to left node
Struct node  *right.       //pointer to right node
};
```

Binary Tree Representation in "Java":

```
Class Node
{

Int    key;
Node left, right;

Public Node(int item)
{
 Key =item;
left=right=null;
}
};
```

BST Basic Operations:

The basic operations that can be performed on a binary search tree data structure, are the following −

- **Insert** − Inserts an element in a tree/create a tree.
- **Search** − Searches an element in a tree.
- **Preorder Traversal** − Traverses a tree in a pre-order manner.
- **Inorder Traversal** − Traverses a tree in an in-order manner.
- **Postorder Traversal** − Traverses a tree in a post-order manner.

Creation of (tree/ insert) a node in the tree:

Code in ‘C’:

```
struct node
{
    int data;              //the element
    struct node*left;          //pointer to left node
    struct node*right;         //pointer to right node
};
```

```
struct node* insert(struct node* root, int data)
{
   if (root == NULL)    //If the tree is empty, return a new,single node
       return newNode(data);
   else
   {
       //Otherwise, recur down the tree
       if (data <= root->data)
           root->left  = insert(root->left, data);
       else
           root->right = insert(root->right, data);
       //return the (unchanged) root pointer
       return root;
   }
}
```

Code in “Java”:

```
/*
 *  Java Program to Implement Binary Tree
    In the below code you can perform::::

    1. Insertion in the tree.
    2. Search a node in the tree.
    3. Count the nodes in the tree.
    4. If tree is empty or not…..

 */
```

```java
import java.util.Scanner;

/* Class BTNode */
class BTNode
{
    BTNode left, right;
    int data;

    /* Constructor */
    public BTNode()
    {
        left = null;
        right = null;
        data = 0;
    }
    /* Constructor */
    public BTNode(int n)
    {
        left = null;
        right = null;
        data = n;
    }
    /* Function to set left node */
    public void setLeft(BTNode n)
    {
        left = n;
```

```
    }
    /* Function to set right node */
    public void setRight(BTNode n)
    {
        right = n;
    }
    /* Function to get left node */
    public BTNode getLeft()
    {
        return left;
    }
    /* Function to get right node */
    public BTNode getRight()
    {
        return right;
    }
    /* Function to set data to node */
    public void setData(int d)
    {
        data = d;
    }
    /* Function to get data from node */
    public int getData()
    {
        return data;
    }
}
```

```java
/* Class BT */
class BT
{
    private BTNode root;

    /* Constructor */
    public BT()
    {
        root = null;
    }
    /* Function to check if tree is empty */
    public boolean isEmpty()
    {
        return root == null;
    }
    /* Functions to insert data */
    public void insert(int data)
    {
        root = insert(root, data);
    }
    /* Function to insert data recursively */
    private BTNode insert(BTNode node, int data)
    {
        if (node == null)
            node = new BTNode(data);
        else
```

```
        {
            if (node.getRight() == null)
                node.right = insert(node.right, data);
            else
                node.left = insert(node.left, data);
        }
        return node;
    }
    /* Function to count number of nodes */
    public int countNodes()
    {
        return countNodes(root);
    }
    /* Function to count number of nodes recursively */
    private int countNodes(BTNode r)
    {
        if (r == null)
            return 0;
        else
        {
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }
    /* Function to search for an element */
```

```java
public boolean search(int val)
{
    return search(root, val);
}
/* Function to search for an element recursively */
private boolean search(BTNode r, int val)
{
    if (r.getData() == val)
        return true;
    if (r.getLeft() != null)
        if (search(r.getLeft(), val))
            return true;
    if (r.getRight() != null)
        if (search(r.getRight(), val))
            return true;
    return false;
}
/* Function for inorder traversal */
public void inorder()
{
    inorder(root);
}
private void inorder(BTNode r)
{
    if (r != null)
    {
        inorder(r.getLeft());
```

```
            System.out.print(r.getData() +" ");
            inorder(r.getRight());
        }
    }
    /* Function for preorder traversal */
    public void preorder()
    {
        preorder(root);
    }
    private void preorder(BTNode r)
    {
        if (r != null)
        {
            System.out.print(r.getData() +" ");
            preorder(r.getLeft());
            preorder(r.getRight());
        }
    }
    /* Function for postorder traversal */
    public void postorder()
    {
        postorder(root);
    }
    private void postorder(BTNode r)
    {
        if (r != null)
        {
```

```
            postorder(r.getLeft());
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }
}

/* Class BinaryTree */
public class BinaryTree
{
    public static void main(String[] args)
   {
       Scanner scan = new Scanner(System.in);
       /* Creating object of BT */
       BT bt = new BT();
       /*  Perform tree operations  */
       System.out.println("Binary Tree Test\n");
       char ch;
       do
       {
           System.out.println("\nBinary Tree Operations\n");
           System.out.println("1. insert ");
           System.out.println("2. search");
           System.out.println("3. count nodes");
           System.out.println("4. check empty");

           int choice = scan.nextInt();
```

```
            switch (choice)
            {
            case 1 :
                System.out.println("Enter integer element to insert");
                bt.insert( scan.nextInt() );
                break;
            case 2 :
                System.out.println("Enter integer element to search");
                        System.out.println("Search result : "+ bt.search(
scan.nextInt() ));
                break;
            case 3 :
                System.out.println("Nodes = "+ bt.countNodes());
                break;
            case 4 :
                System.out.println("Empty status = "+ bt.isEmpty());
                break;
            default :
                System.out.println("Wrong Entry \n ");
                break;
            }
            /*  Display tree  */
            System.out.print("\nPost order : ");
            bt.postorder();
            System.out.print("\nPre order : ");
            bt.preorder();
            System.out.print("\nIn order : ");
```

```
            bt.inorder();

              System.out.println("\n\nDo you want to continue (Type
y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```
