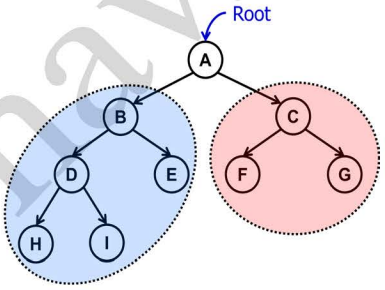
CHAPTER 6
DATA STRUCTURES: TREE

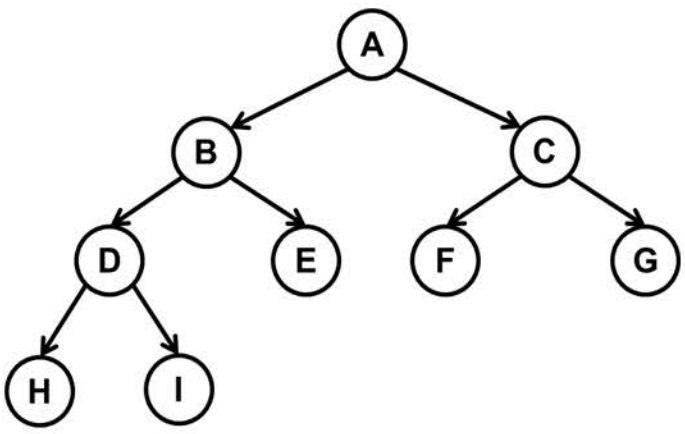
6.1 Tree

- **Tree:**
  - "A data structure that represents the **hierarchical relationship** among individual data items."
  - **Examples**:
    - Classification of books in a library
    - Hierarchy of structure in an organization
    - Mathematical expression
    - Family tree
- **Binary Tree:**
  - "A tree in which each & every node has **at most** two sub-trees."
  - **3 disjoint subsets:**
    - Root
    - Left sub-tree
    - Right sub-tree

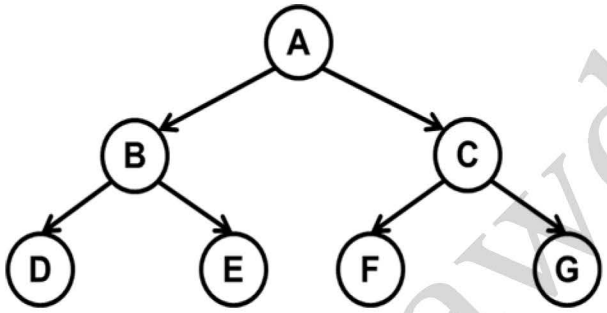


- **Root:**
  - "The root of the tree is a node which is the **forefather** of all children."
- **Leaf:**
  - "A node which has **no** any child node."
  - Examples: H, I, E, F, G (in the above figure)
- **Brother:**
  - "Two nodes are brothers if they are children of the **same father**."
- **Edge/Branch:**
  - "The **link** between **parent** and **child** node."
- **Directed Edge:**
  - "If the edge has a direction from one node to another, it is called **directed** edge."
- **Level of a node:**
  - "**Distance** from the root node."
  - The level of the **root** node is **zero** (**0**).
- **Depth of a Tree:**
  - "The **maximum** level of any **leaf** node in a tree."
  - Equals the length of the longest path from the root to any leaf.

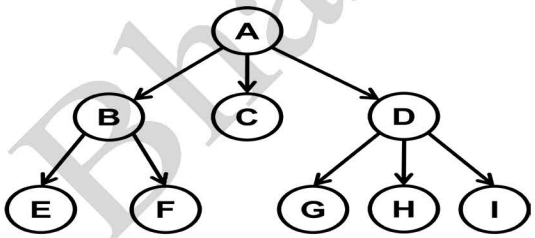
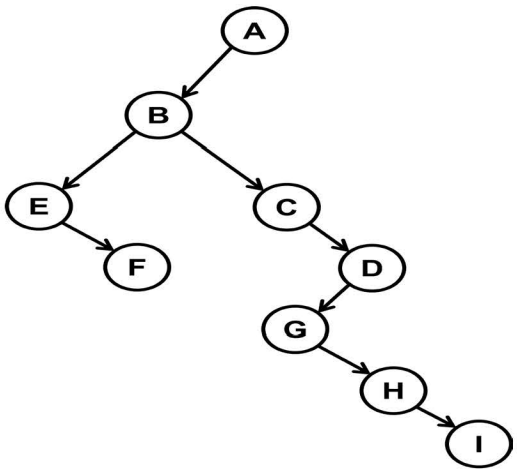
- **Height of a Tree:**
  - "**Total** number of available **levels** in a tree."
  - **Height** = **Depth + 1**.
- **Weight of a Tree:**
  - "**Total** number of **leaf** nodes available in a tree."
- **Degree of a Node:**
  - "Total number of child nodes (sub-trees) of a node."
- **In-degree of a node:**
  - "Number of branches **terminated** at a node."
  - In-degree of the **root** node is **zero**.
- **Out-degree of a node:**
  - "Number of branches **emerging** from a node."
  - Out-degree of a **leaf** node is **zero**.
- **Degree of a Tree:**
  - "Maximum degree of a node in a given tree."
  - The degree of the binary tree is 2.
- **Turnery Tree:**
  - "A tree in which each & every node has **at most three** sub-trees".
- **M-ary Tree:**
  - "A tree in which each & every node has **at most** '**m**' sub-trees".
- **Path of a tree:**
  - "A sequence of **continuous** edges."
- **Strictly Binary Tree:**
  - "A binary tree in which every **non-leaf** node has **non-empty** left & right sub-tree."
  - "A binary tree in which every **non-leaf** node has two child nodes (sub-trees)."
  - Total nodes = **2 n – 1,** where, **n** = no. of leaf nodes.
  - **Example:**



- **Complete Binary Tree:**
  - "A strictly binary tree in which all **leaf** nodes are at the **same** level."
  - Total nodes = $\mathbf{2^{d+1} - 1}$
  - Total leaf nodes = $\mathbf{2^d}$
  - Total non-leaf nodes = $\mathbf{2^d - 1}$, where, **d** = depth of tree.
  - **Example:**



- **Similar Binary Trees:**
  - "Two binary trees are similar if they have **similar** left and right sub-trees."
- **General Tree:**
  - "A tree in which any node can have any number of child nodes (sub-trees)."
- **Conversion of General Tree into Binary Tree:**
  - Explanation: (As per covered in class)

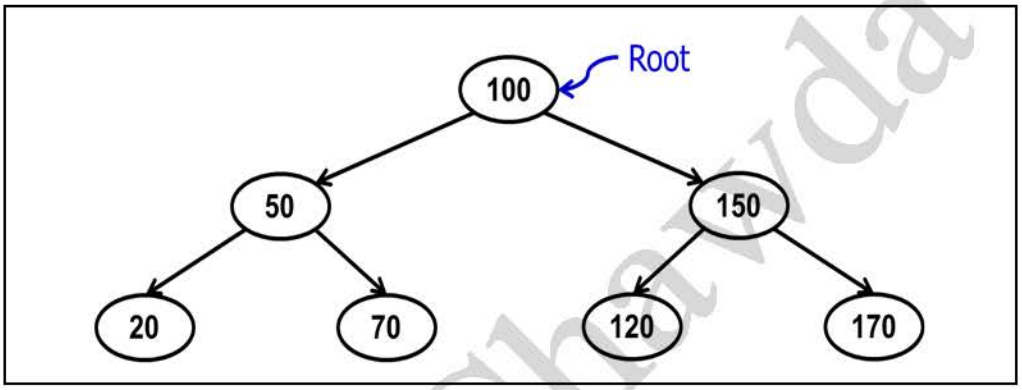
General Tree	Binary Tree
	

- **Forest:**
  - "A forest is a **disjoint** union of trees."

6.2 Binary Search Tree (BST)

6.2.1 Concept

- **Definition:**
  - A binary tree in which for any given node 'nd' –
    - Value of each node in left sub-tree < value of 'nd'
    - Value of each node in right sub-tree ≥ value of 'nd'.
- **Example:**



- **Operations:**
  - Insert
  - Traverse (In-order, Pre-order, Post-order)
  - Search

6.2.2 Tree Traversal

In-order Traversal (L V R)	Pre-order Traversal (V L R)	Post-order Traversal (L R V)
- Visit Left Sub-tree - Visit Vertex - Visit Right Sub-tree	- Visit Vertex - Visit Left Sub-tree - Visit Right Sub-tree	- Visit Left Sub-tree - Visit Right Sub-tree - Visit Vertex

- Explanation: (As per covered in class)

6.3 Binary Search Tree – Implementation

```
#include<stdio.h>
#include<stdlib.h>

        // Declare a structure to create a node...

struct node
{
        struct node *left;
        int no;
        struct node *right;
};

        // Declare global variables...
struct node *root=NULL;
struct node *temp=NULL;
struct node *prev=NULL;
struct node *new1=NULL;
struct node *tsucc=NULL;


        // main function starts...
void main()
{
        int choice,n;
        void insert_tree(int);
        void delete_tree(int);
        void trav_in(struct node *);
        void trav_pre(struct node *);
        void trav_post(struct node *);

        while(1)
        {
                printf("\n 1.insert a node into a tree...");
                printf("\n 2.delete a node from a tree...");
                printf("\n 3.inorder traversal...");
                printf("\n 4.preorder traversal...");
                printf("\n 5.postorder traversal...");
                printf("\n 6.exit...");

                printf("\n\n Enter ur choice...");
                scanf("%d", &choice);

                switch(choice)
                {
                        case 1:
                                printf("Enter no. to be inserted...:");
                                scanf("%d",&n);
                                insert_tree(n);
                                break;
```

```
                        case 2:
                                printf("Enter no. to be deleted...:");
                                scanf("%d",&n);
                                delete_tree(n);
                                break;
                        case 3:
                                printf(" Inorder : ");
                                trav_in(root);
                                printf("\n");
                                break;
                        case 4:
                                printf(" Preorder : ");
                                trav_pre(root);
                                printf("\n");
                                break;
                        case 5:
                                printf(" Postorder : ");
                                trav_post(root);
                                printf("\n");
                                break;
                        case 6:
                                printf("\n Program completed successfully...");
                                exit(0);
                                break;
                        default:
                                printf("\n Enter valid choice...\n");
                }
        }//while ends...
}// main() ends...
```

// Insertion of a Node...

```
void insert_tree( int  x )
{
        // prepare new node...
        new1 = (struct node*) malloc ( sizeof ( struct node ) );
        new1→no = x;
        new1→left = NULL;
        new1→right = NULL;

        // if the tree is not available...
        if(root == NULL)
        {
                root = new1;
                return;
        }
```

```
        // if the tree is available, traverse it...
        temp = root;
        while( temp  !=  NULL )
        {
                prev = temp;
                if( temp→no > x )
                        temp = temp→left;
                else
                        temp = temp→right;
        }
        // insert node at proper subtree...
        if( prev→no  >  x )
                prev→left = new1;
        else
                prev→right = new1;
}

// In-order Traversal...
void trav_in( struct node *nd )
{
        if( nd == NULL )
                return;
        trav_in( nd→left );
        printf(" %d ",nd→no);
        trav_in( nd→right );
}

// Pre-order Traversal...
void trav_pre( struct node *nd )
{
        if( nd==NULL )
                return;
        printf(" %d ",nd→no);
        trav_pre( nd→left );
        trav_pre( nd→right );
}

// Post-order Traversal...
void trav_post( struct node *nd )
{
        if( nd==NULL )
                return;
        trav_post( nd→left );
        trav_post( nd→right );
        printf(" %d ",nd→no );
}
```

```
// Deletion of a Node...
void delete_tree( int   x )
{
        // First case: Tree not available...
        if(root == NULL)
        {
                printf("\n Tree is empty...");
                return;
        }

        // find node to be deleted...
        temp = root;
        while( temp != NULL   &&   temp→no != x )
        {
                prev = temp;
                if( temp→no   >   x )
                        temp = temp→left;
                else
                        temp = temp→right;
        }

        // Second case: Node not found...
        if( temp == NULL )
        {
                printf("\n node to be deleted not found...");
                return;
        }

        // Third case: i) Node found with two children...
        if( temp→left != NULL   &&   temp→right != NULL )
        {
                prev = temp;
                tsucc = temp→right;
                while(tsucc→left != NULL)
                {
                        prev = tsucc;
                        tsucc = tsucc→left;
                }
                temp→no = tsucc→no;
                temp = tsucc;
        }
```

```
// Third case: ii) Node found with no any child...
if( temp→left == NULL   &&   temp→right == NULL )
{
        if( prev→left == temp )
                prev→left = NULL;
        else
                prev→right = NULL;
        free(temp);
        return;
}

// Third case: iii) Node found with only left child...
if( temp→left != NULL   &&   temp→right == NULL )
{
        if( prev→left == temp )
                prev→left = temp→left;
        else
                prev→right = temp→left;
        free(temp);
        return;
}

// Third case: iv) Node found with only right child...
if( temp→left == NULL   &&   temp→right != NULL )
{
        if( prev→left == temp )
                prev→left = temp→right;
        else
                prev→right = temp→right;
        free(temp);
        return;
}
}
```


● ● ●