Introduction to trees

- So far we have discussed mainly linear data structures – strings, arrays, lists, stacks and queues
- Now we will discuss a non-linear data structure called **tree**.
- Trees are mainly used to represent data containing a hierarchical relationship between elements, for example, records, family trees and table of contents.
- Consider a parent-child relationship

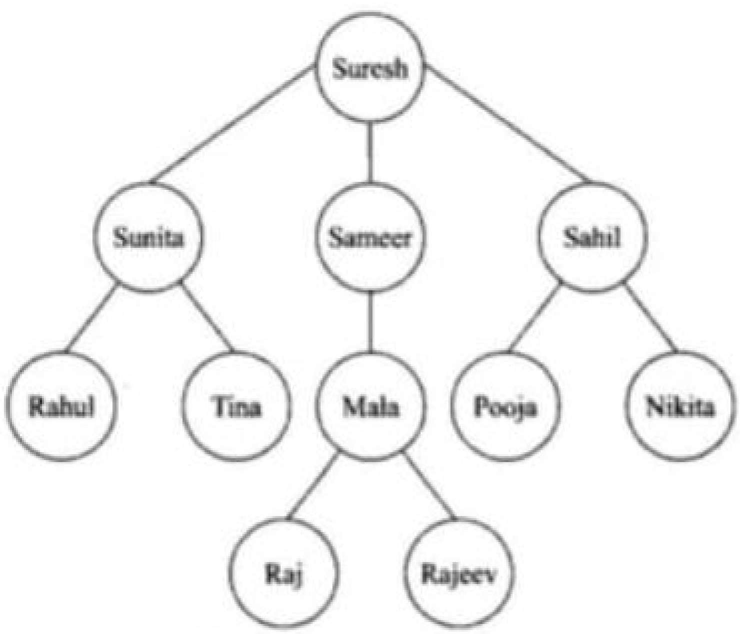
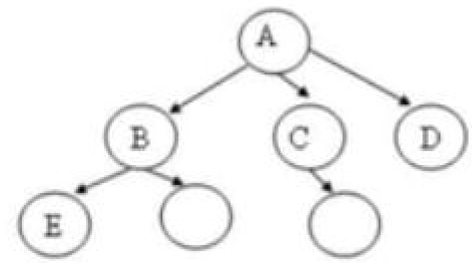


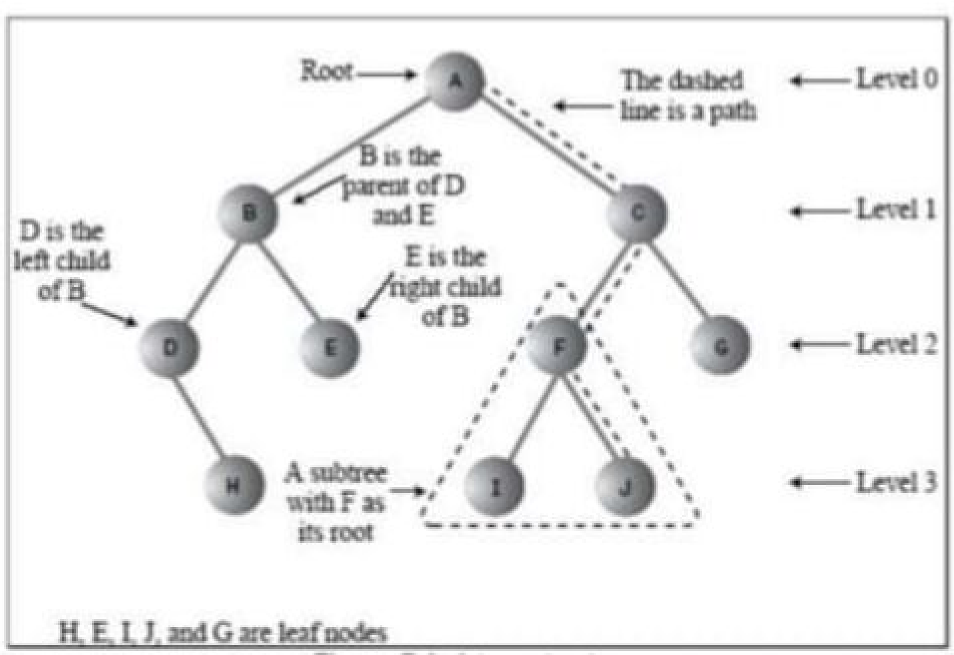
Fig. 8.1 *A Hypothetical Family Tree*

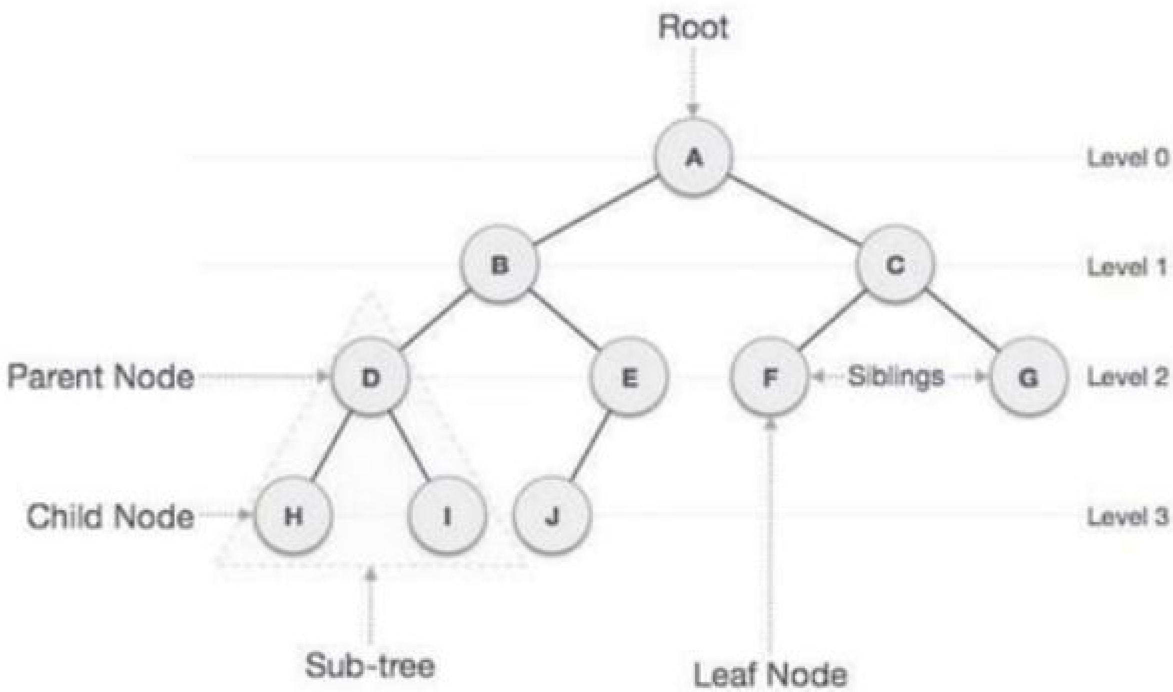
Tree

- A tree is an abstract model of a hierarchical structure that consists of nodes with a parent-child relationship.
  - Tree is a sequence of nodes
  - There is a starting node known as a root node
  - Every node other than the root has a parent node.
  - Nodes may have any number of children



A has 3 children, B, C, D
A is parent of B



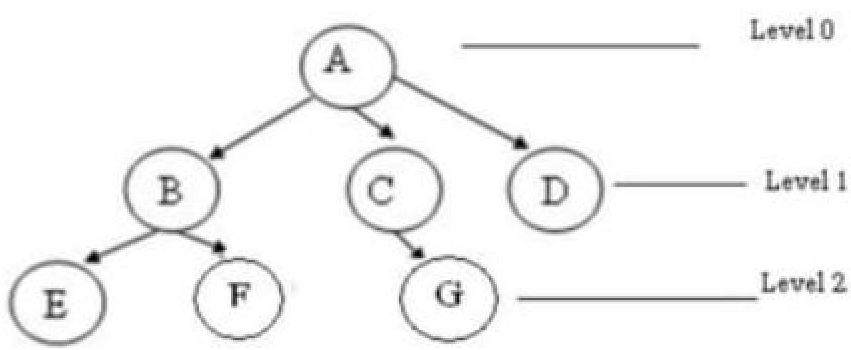


Some Key Terms:

- **Root** – Node at the top of the tree is called root.
- **Parent** – Any node except root node has one edge upward to a node called parent.
- **Child** – Node below a given node connected by its edge downward is called its child node.
- **Sibling** – Child of same node are called siblings
- **Leaf** – Node which does not have any child node is called leaf node.
- **Sub tree** – Sub tree represents descendants of a node.
- **Levels** – Level of a node represents the generation of a node. If 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.

Some Key Terms:

- Degree of a node:
  - The degree of a node is the number of children of that node
- Degree of a Tree:
  - The degree of a tree is the maximum degree of nodes in a given tree
- Path:
  - It is the sequence of consecutive edges from source node to destination node.
- Height of a node:
  - The height of a node is the max path length form that node to a leaf node.
- Height of a tree:
  - The height of a tree is the height of the root
- Depth of a tree:
  - Depth of a tree is the max level of any leaf in the tree



- ✔ A is the root node
- ✔ B is the parent of E and F
- ✔ D is the sibling of B and C
- ✔ E and F are children of B

Characteristics of trees

- Non-linear data structure
- Combines advantages of an ordered array
- Searching as fast as in ordered array
- Insertion and deletion as fast as in linked list
- Simple and fast

Application

- Directory structure of a file store
- Structure of an arithmetic expressions
- Used in almost every 3D video game to determine what objects need to be rendered.
- Used in almost every high-bandwidth router for storing router-tables.
- used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.

Introduction To Binary Trees

- A **binary tree**, is a tree in which no node can have more than two children.

- Consider a binary tree T, here 'A' is the root node of the binary tree T.

- '**B**' is the left child of 'A' and '**C**' is the right child of 'A'
  - i.e A is a father of B and C.
  - The node B and C are called siblings.

- Nodes D,H,I,F,J are leaf node

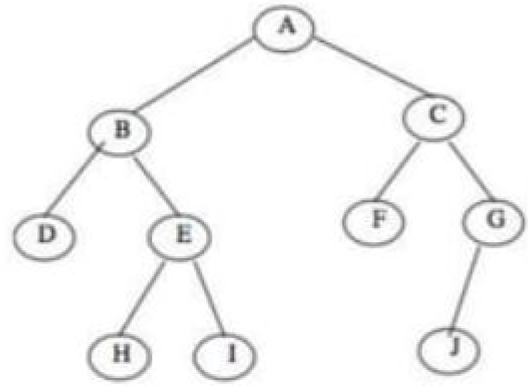
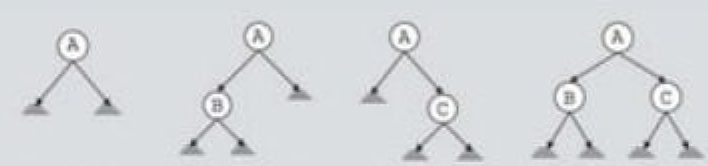


Fig. 8.3. Binary tree

Binary Trees

- A **binary tree**, *T*, is either empty or such that
  - *I.* *T* has a special node called the **root** node
  - *II.* *T* has two sets of nodes $\mathbf{L_T}$ and $\mathbf{R_T}$, called the **left subtree** and **right subtree** of *T*, respectively.
  - *III.* L_T and R_T are binary trees.

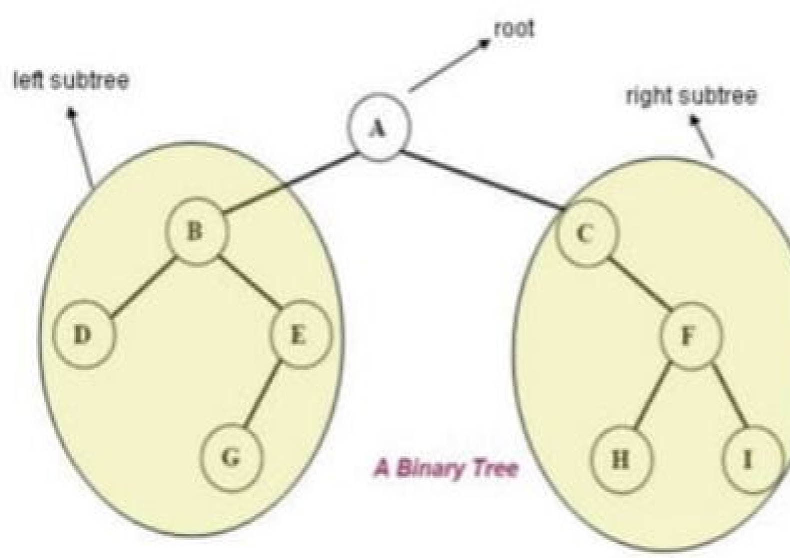


(a) Binary tree with one node | (b) Binary tree with two nodes | (c) Binary tree with two nodes | (d) Binary tree with three nodes

Binary Tree

- A binary tree is a finite set of elements that are either empty or is partitioned into three disjoint subsets.
- The first subset contains a single element called the **root** of the tree.
- The other two subsets are themselves binary trees called the **left** and **right sub-trees** of the original tree.
- A left or right sub-tree can be empty.
- Each element of a binary tree is called a **node** of the tree.

The following figure shows a binary tree with 9 nodes where A is the root

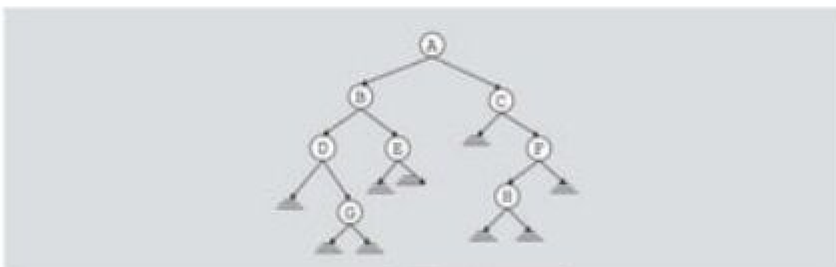


Binary Tree

- The root node of this binary tree is A.

- The left sub tree of the root node, which we denoted by $\mathbf{L_A}$, is the set L_A = {B,D,E,G} and the right sub tree of the root node, $\mathbf{R_A}$ is the set R_A={C,F,H}

- The root node of L_A is node B, the root node of R_A is C and so on



Binary Tree Properties

- If a binary tree contains **m** nodes at level **L**, it contains at most **2m** nodes at level **L+1**

- Since a binary tree can contain at most 1 node at level 0 (the root), it contains at most **2L** nodes at level L.

Types of Binary Tree

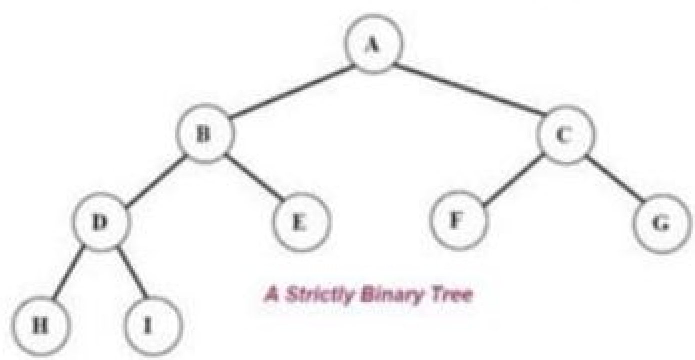
- Complete binary tree
- Strictly binary tree
- Almost complete binary tree

Strictly binary tree

- If every non-leaf node in a binary tree has nonempty left and right sub-trees, then such a tree is called a **strictly binary tree**.

- **Or, to put it another way,** all of the nodes in a strictly binary tree are of degree zero or two, never degree one.

- A strictly binary tree with N leaves always contains 2N – 1 nodes.

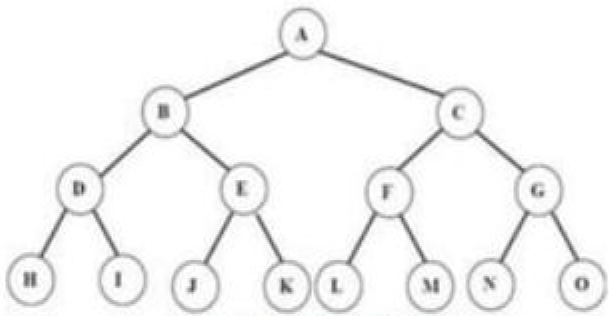


Complete binary tree

- A **complete binary tree** is a **binary tree** in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

- A ***complete binary tree*** of depth **d** is called strictly binary tree if all of whose leaves are at level **d**.

- A complete binary tree has $2^{\mathbf{d}}$ nodes at every depth **d** and $2^{\mathbf{d}}$ -1 non leaf nodes



A complete Binary Tree of depth 3

Almost complete binary tree

- An almost complete binary tree is a tree where for a right child, there is always a left child, but for a left child there may not be a right child.

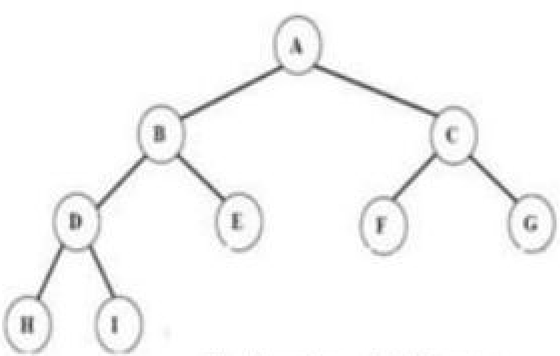


Fig Almost complete binary tree.

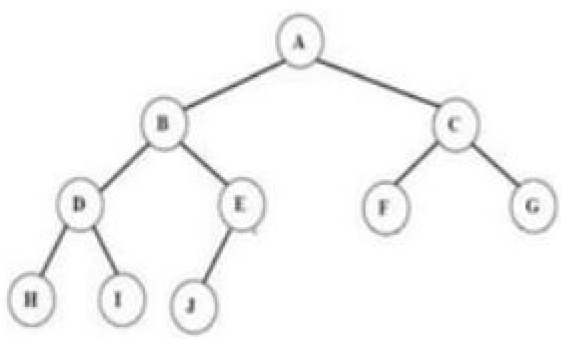


Fig Almost complete binary tree but not strictly binary tree. Since node E has a left son but not a right son.

Operations on Binary tree:

- ✔ **father(n,T):**Return the parent node of the node n in tree T. If n is the root, NULL is returned.
- ✔ **LeftChild(n,T):**Return the left child of node n in tree T. Return NULL if n does not have a left child.
- ✔ **RightChild(n,T)**:Return the right child of node n in tree T. Return NULL if n does not have a right child.
- ✔ **Info(n,T):** Return information stored in node n of tree T (ie. Content of a node).
- ✔ **Sibling(n,T):** return the sibling node of node **n** in tree T. Return NULL if n has no sibling.
- ✔ **Root(T):** Return root node of a tree if and only if the tree is nonempty.
- ✔ **Size(T):** Return the number of nodes in tree T
- ✔ **MakeEmpty(T):** Create an empty tree T
- ✔ **SetLeft(S,T):** Attach the tree S as the left sub-tree of tree T
- ✔ **SetRight(S,T):** Attach the tree S as the right sub-tree of tree T.
- ✔ **Preorder(T):** Traverses all the nodes of tree T in preorder.
- ✔ **postorder(T):** Traverses all the nodes of tree T in postorder
- ✔ **Inorder(T):** Traverses all the nodes of tree T in inorder.

C representation for Binary tree:

```
struct bnode
{
        int info;
        struct bnode  *left;
        struct bnode  *right;
};
struct bnode *root=NUI
```

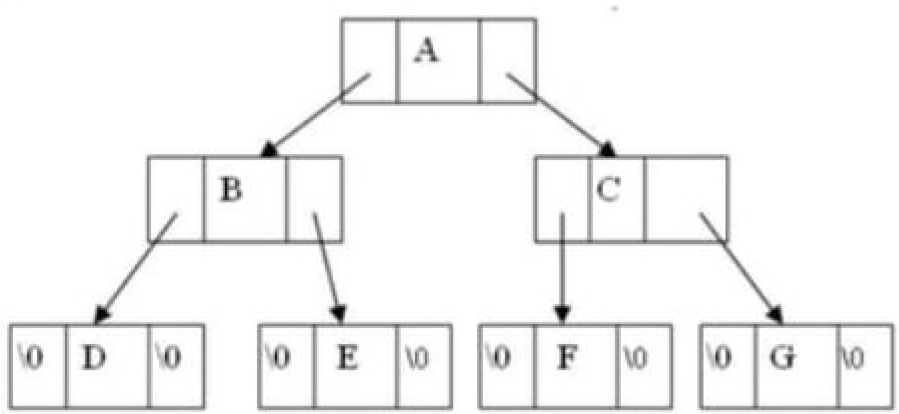


Fig: Structure of Binary tree

Tree traversal

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- All nodes are connected via edges (links) we always start from the root (head) node.
- There are three ways which we use to traverse a tree
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal
- Generally we traverse a tree to search or locate given item or key in the tree or to print all the values it contains.

Pre-order, In-order, Post-order

- Pre-order

<root><left><right>

- In-order

<left><root><right>

- Post-order

<left><right><root>

Pre-order Traversal

- The preorder traversal of a nonempty binary tree is defined as follows:
  - Visit the root node
  - Traverse the left sub-tree in preorder
  - Traverse the right sub-tree in preorder

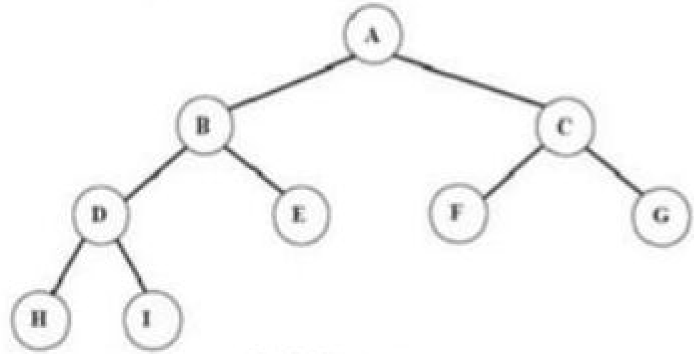


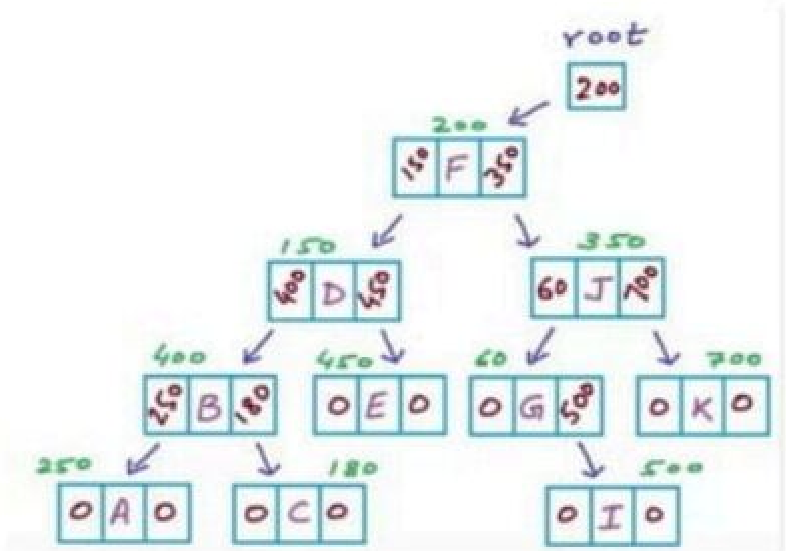
fig Binary tree

The preorder traversal output of the given tree is: A B D H I E C F G

The preorder is also known as depth first order.

Pre-order Pseudocode

```
struct Node{
   char data;
   Node *left;
   Node *right;
}
void Preorder(Node *root)
{
   if (root==NULL) return;
   printf ("%c", root->data);
   Preorder(root->left);
   Preorder(root->right);
}
```



In-order traversal

- The in-order traversal of a nonempty binary tree is defined as follows:
  - Traverse the left sub-tree in in-order
  - Visit the root node
  - Traverse the right sub-tree in inorder

- The in-order traversal outpu of the given tree is H D I B E A F C G

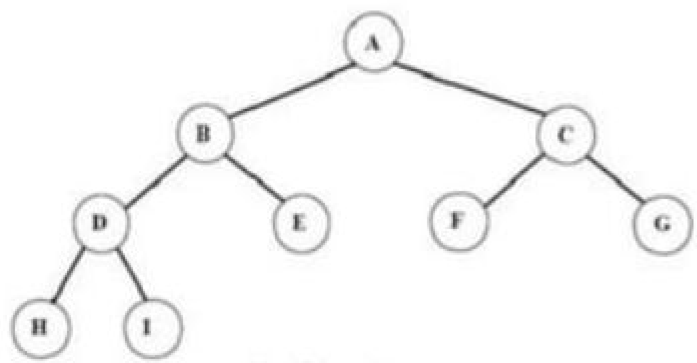
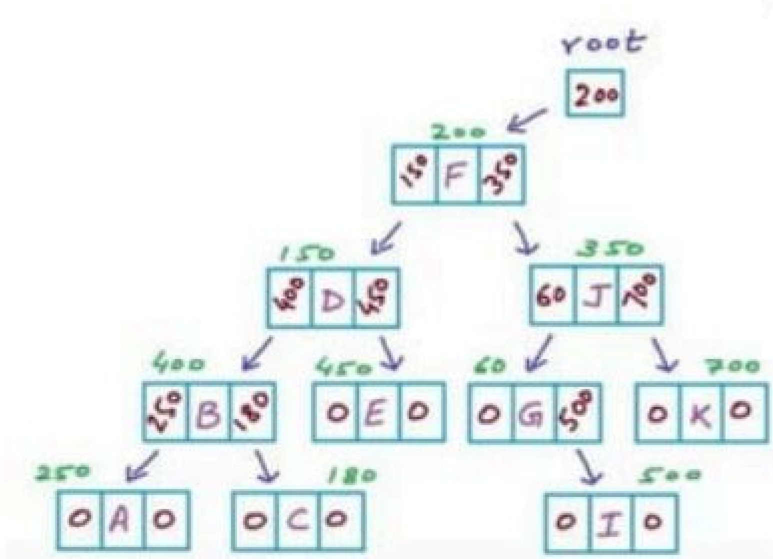


fig Binary tree

In-order Pseudocode

```
struct Node{
   char data;
   Node *left;
   Node *right;
}
void Inorder(Node *root)
{
   if (root==NULL) return;
   Inorder(root->left);
   printf ("%c", root->data);
   Inorder(root->right);
}
```



Post-order traversal

- The in-order traversal of a nonempty binary tree is defined as follows:
  - Traverse the left sub-tree in post-order
  - Traverse the right sub-tree in post-order
  - Visit the root node

- The in-order traversal outpu of the given tree is H I D E B F G C A

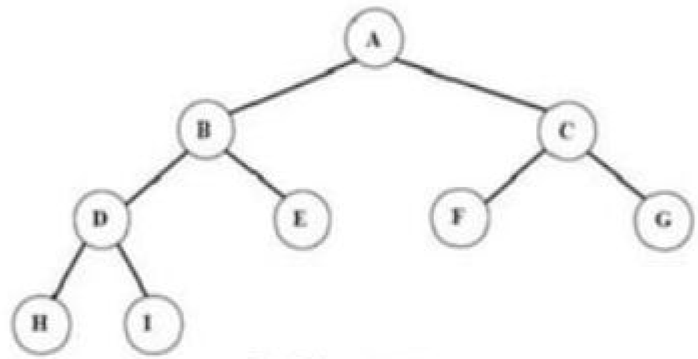
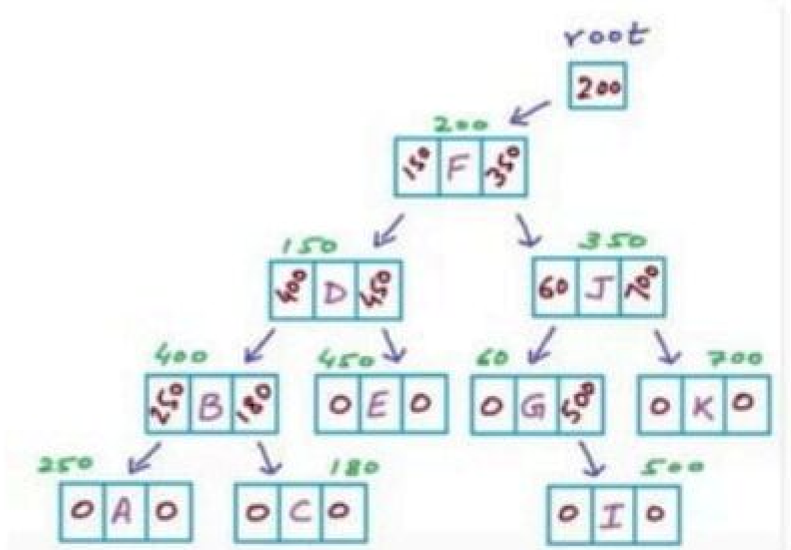


fig Binary tree

Post-order Pseudocode

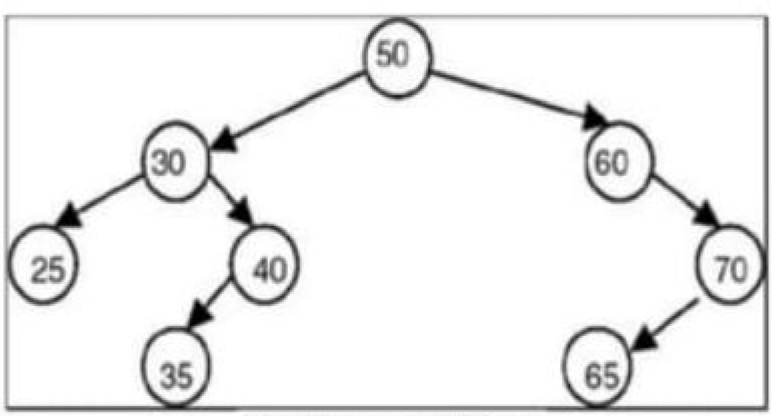
```
struct Node{
   char data;
   Node *left;
   Node *right;
}
void Postorder(Node *root)
{
   if (root==NULL) return;
   Postorder(root->left);
   Postorder(root->right);
   printf ("%c", root->data);
}
```



Binary Search Tree(BST)

- A binary search tree (BST) is a binary tree that is either empty or in which every node contains a key (value) and satisfies the following conditions:
  - All keys in the left sub-tree of the root are smaller than the key in the root node
  - All keys in the right sub-tree of the root are greater than the key in the root node
  - The left and right sub-trees of the root are again binary search trees

Binary Search Tree(BST)



The binary search tree.