Tree Data Structure

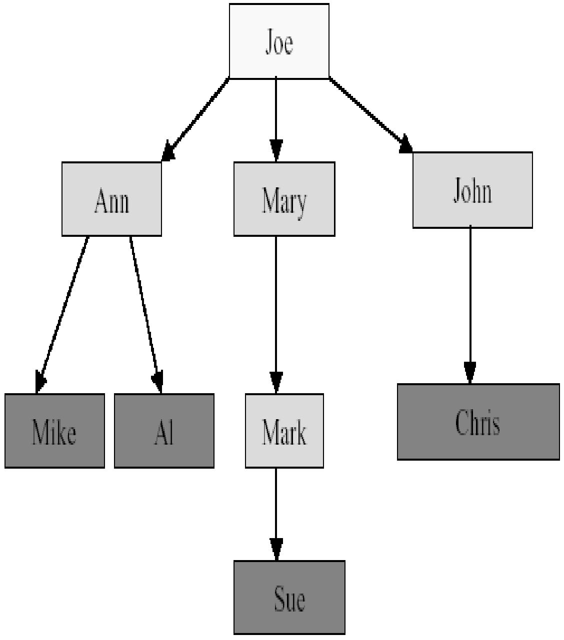
Mohammad Asad Abbasi

Lecture 10

Linear Lists and Trees

- Linear lists are useful for serially ordered data
  - $(e_1, e_2, e_3, \ldots, e_n)$
  - Days of week
  - Months in a year
  - Students in a class

- Trees are useful for hierarchically ordered data
  - Joe's descendants
  - Corporate structure
  - Government Subdivisions
  - Software structure

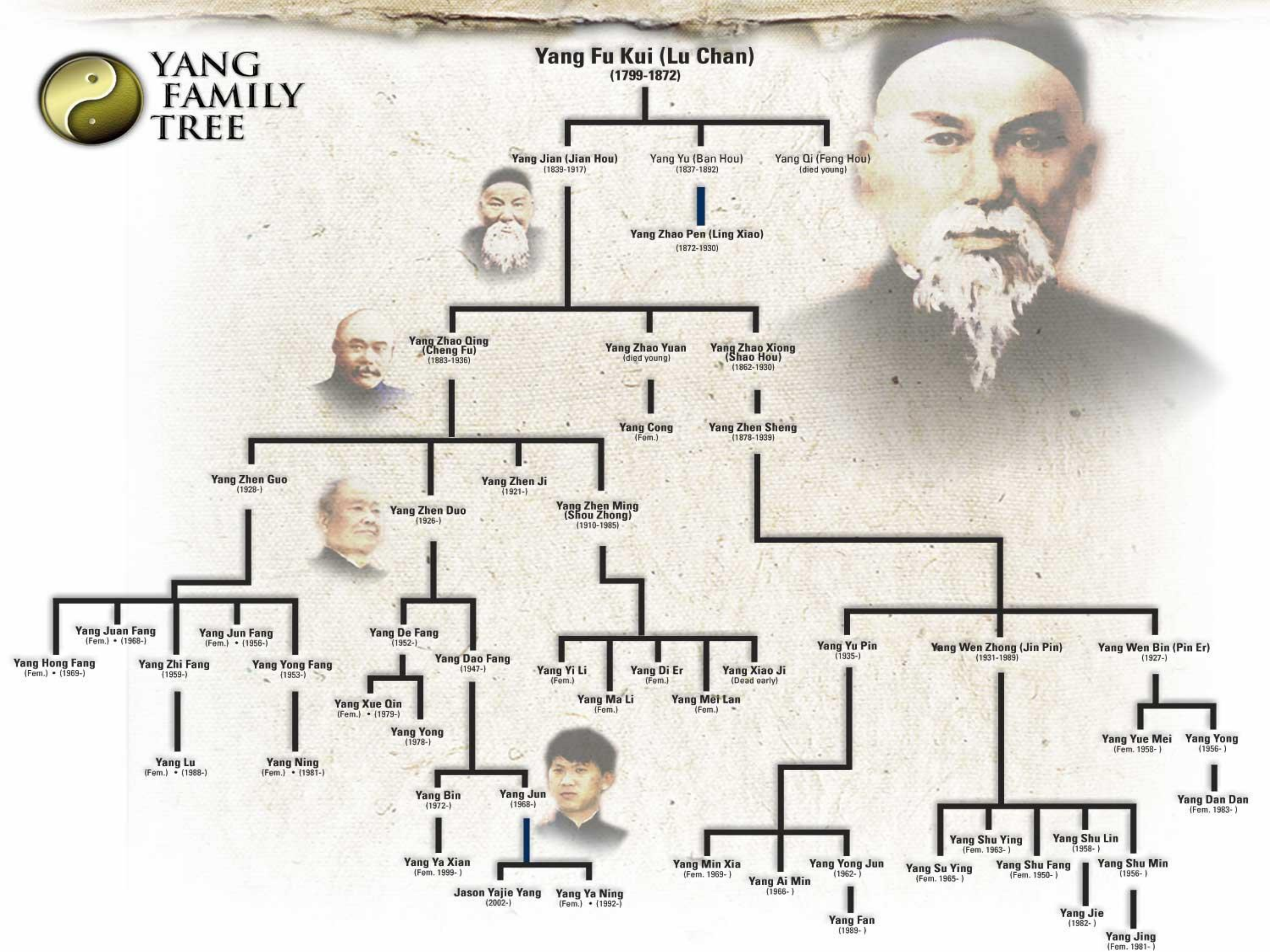


Trees

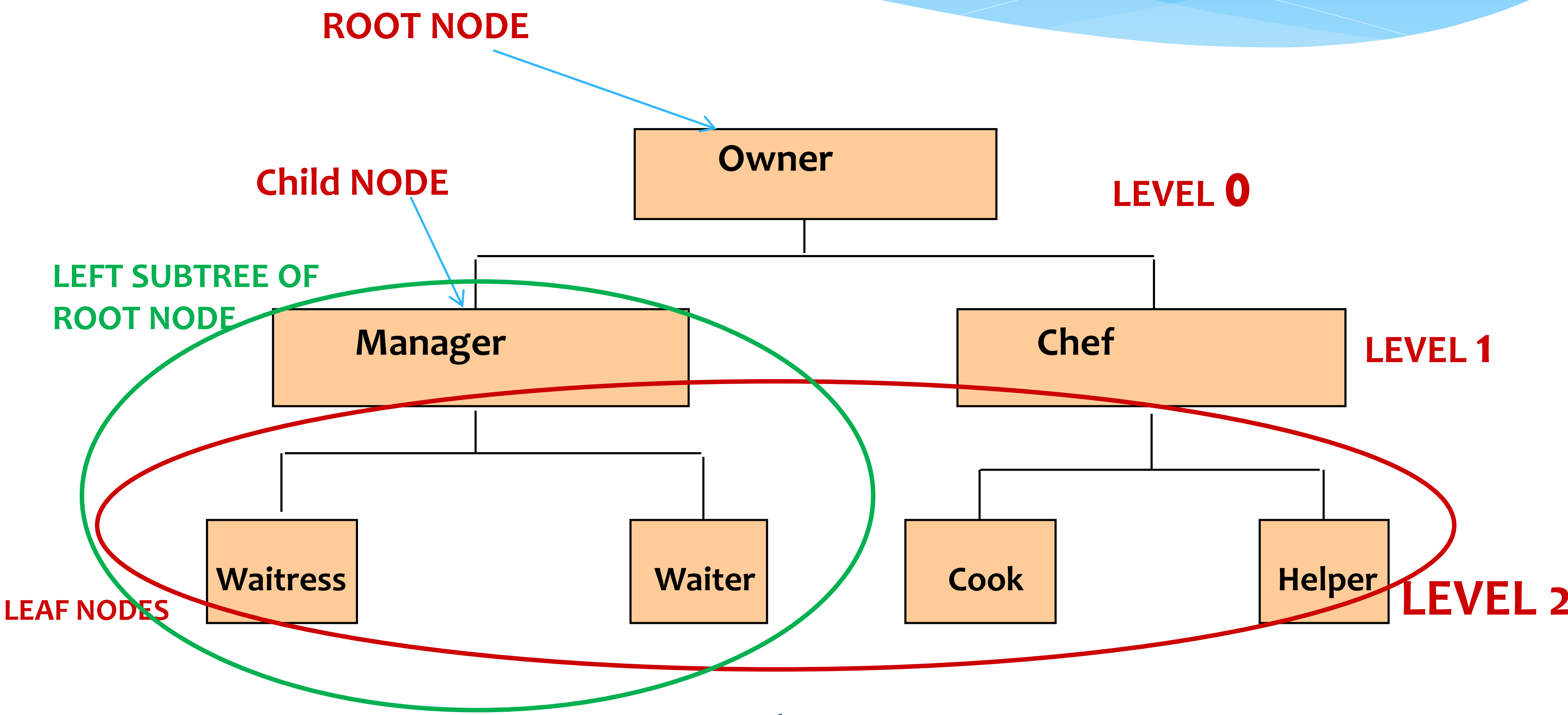
- Compare to linked lists, **trees are non-linear data structures**
  - In linked list, each node points other node(s)
- In a tree structure, each node may point to several nodes, which may in turn point to several other nodes
  - Flexible and powerful data structure that can be used for a variety of applications

Trees

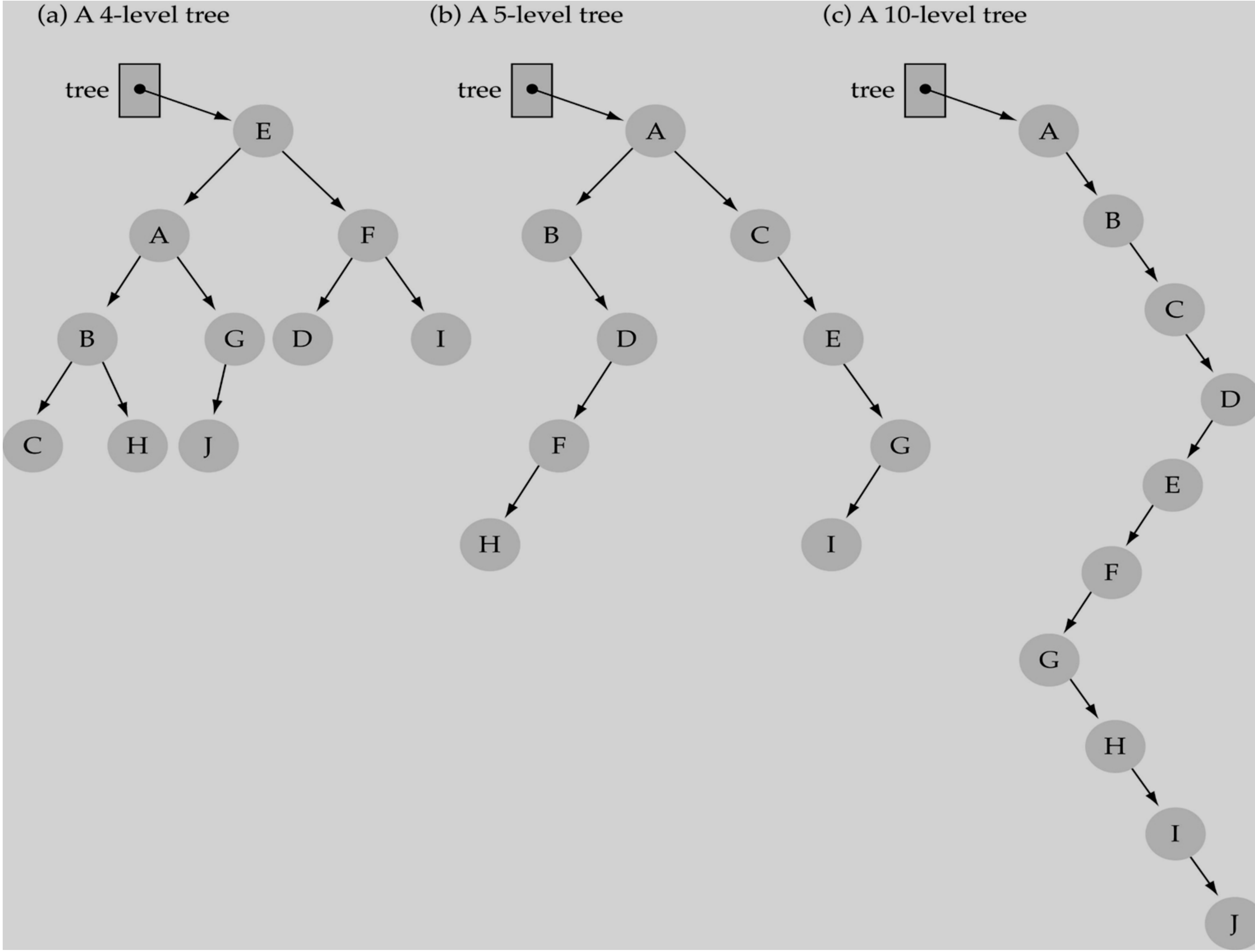
- Tree *t* is finite nonempty set of elements
- One of these elements is called the root node
- The remaining elements, if any, known as child nodes are partitioned into trees, called sub trees of a tree


YANG FAMILY TREE
Yang Fu Kui (Lu Chan)
(1799-1872)
Yang Jian (Jian Hou)
(1839-1917)
Yang Yu (Ban Hou)
(1837-1892)
Yang Qi (Feng Hou)
(died young)
Yang Zhao Pen (Ling Xiao)
(1872-1930)
Yang Zhao Qing (Cheng Fu)
(1883-1936)
Yang Zhao Yuan
(died young)
Yang Zhao Xiong (Shao Hou)
(1862-1930)
Yang Cong
(Fem.)
Yang Zhen Sheng
(1878-1939)
Yang Zhen Guo
(1928-)
Yang Zhen Duo
(1926-)
Yang Zhen Ji
(1921-)
Yang Zhen Ming (Shou Zhong)
(1910-1985)
Yang Hong Fang
(Fem.) • (1969-)
Yang Juan Fang
(Fem.) • (1968-)
Yang Zhi Fang
(1959-)
Yang Jun Fang
(Fem.) • (1956-)
Yang Yong Fang
(1953-)
Yang Lu
(Fem.) • (1988-)
Yang Ning
(Fem.) • (1981-)
Yang De Fang
(1952-)
Yang Dao Fang
(1947-)
Yang Xue Qin
(Fem.) • (1979-)
Yang Yong
(1978-)
Yang Bin
(1972-)
Yang Jun
(1968-)
Yang Ya Xian
(Fem. 1999-)
Jason Yajie Yang
(2002-)
Yang Ya Ning
(Fem.) • (1992-)
Yang Yi Li
(Fem.)
Yang Ma Li
(Fem.)
Yang Di Er
(Fem.)
Yang Mei Lan
(Fem.)
Yang Xiao Ji
(Dead early)
Yang Yu Pin
(1935-)
Yang Wen Zhong (Jin Pin)
(1931-1989)
Yang Wen Bin (Pin Er)
(1927-)
Yang Min Xia
(Fem. 1969-)
Yang Ai Min
(1966-)
Yang Yong Jun
(1962-)
Yang Fan
(1989-)
Yang Su Ying
(Fem. 1965-)
Yang Shu Ying
(Fem. 1963-)
Yang Shu Fang
(Fem. 1950-)
Yang Shu Lin
(1958-)
Yang Shu Min
(1956-)
Yang Jie
(1982-)
Yang Jing
(Fem. 1981-)
Yang Yue Mei
(Fem. 1958-)
Yang Yong
(1956-)
Yang Dan Dan
(Fem. 1983-)

Tree

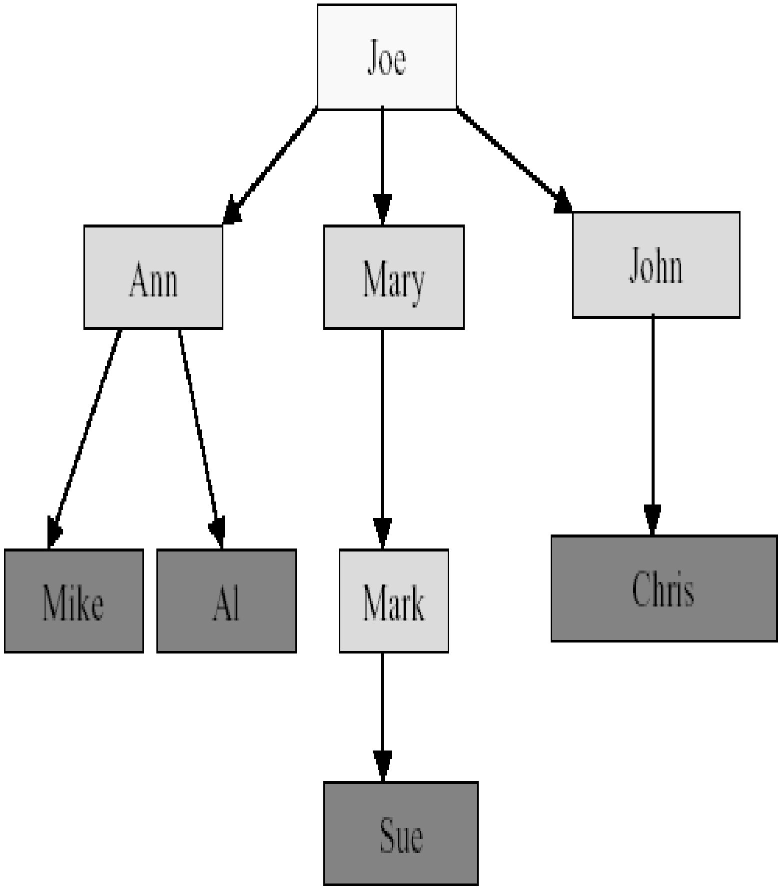


ROOT NODE
Owner
Child NODE
LEVEL 0
LEFT SUBTREE OF
ROOT NODE
Manager
Chef
LEVEL 1
Waitress
Waiter
Cook
Helper
LEAF NODES
LEVEL 2


(a) A 4-level tree
tree
E
A
F
B
G
D
I
C
H
J
(b) A 5-level tree
tree
A
B
C
D
E
F
G
H
I
(c) A 10-level tree
tree
A
B
C
D
E
F
G
H
I
J

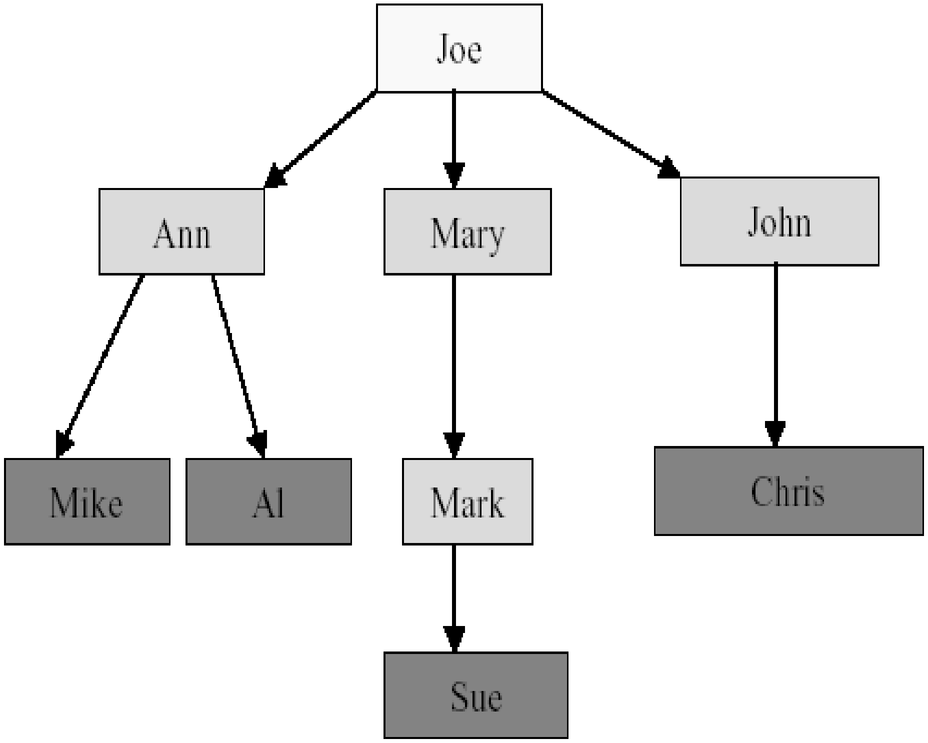
Tree Terminology

- The element at the top of the hierarchy is the **root**
- Elements next in the hierarchy are the **children** of the root
- Elements next in the hierarchy are the **grandchildren** of the root, and so on



Tree Terminology

- Leaves, Parent, Grandparent, Siblings, Ancestors, Descendents



Leaves = {Mike,Al,Sue,Chris}

Parent(Mary) = Joe

Grandparent(Sue) = Mary

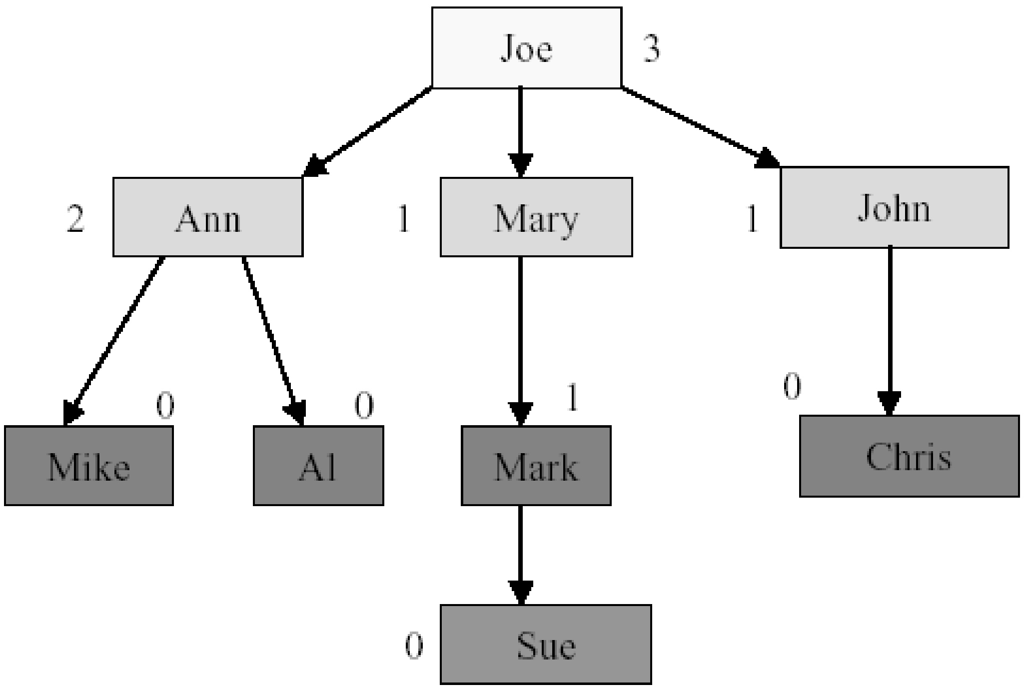
Siblings(Mary) = {Ann,John}

Ancestors(Mike) = {Ann,Joe}

Descendents(Mary)={Mark,Sue}

Node & Tree Degree

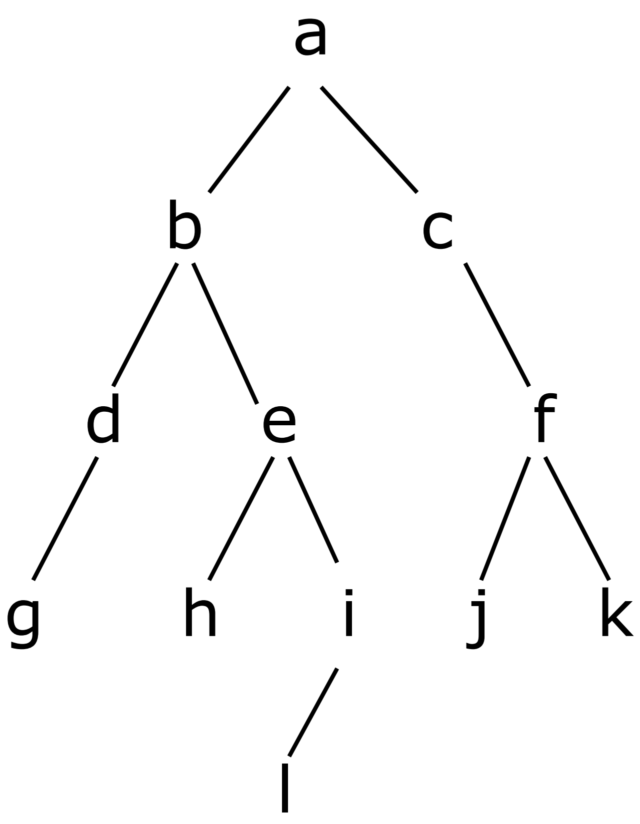
- Node degree is the number of children it has



tree degree = 3

- Tree degree is the maximum of node degrees

Size and Depth



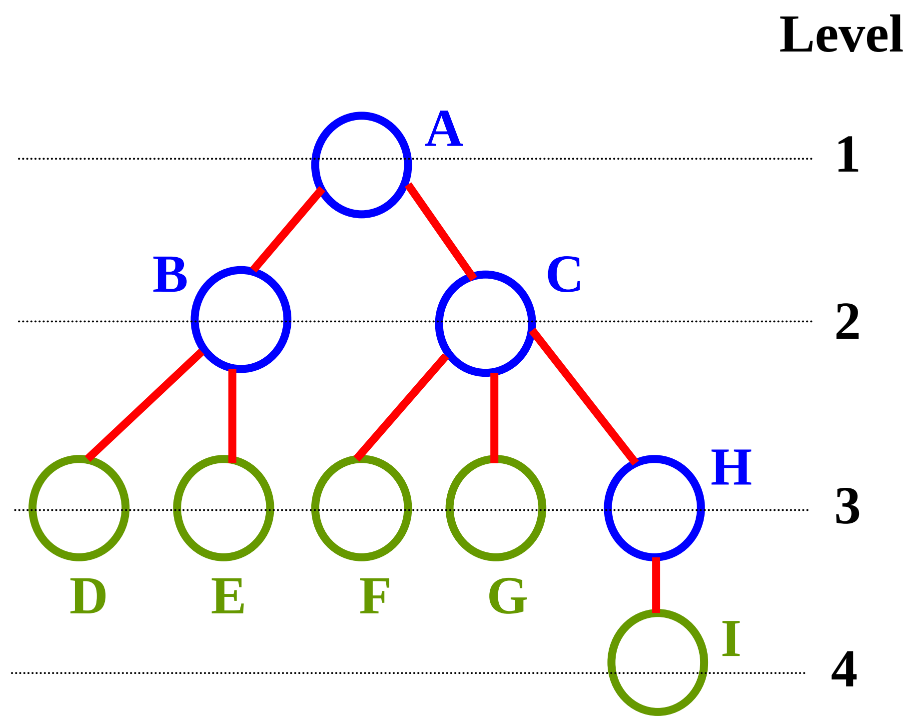
- The size of a binary tree is the number of nodes in it
  - This tree has size 12
- The depth of a node is its distance from the root
  - a is at depth zero
  - e is at depth 2
- The depth of a binary tree is the depth of its deepest node
  - This tree has depth 4

Size and Depth

- **Example**

Property: (*# edges*) = (*#nodes*) - 1

A is the *root* node
B is the *parent* of D and E
C is the *sibling* of B
D and *E* are the *children* of B
D, E, F, G, I are *external nodes*, or *leaves*
A, B, C, H are *internal nodes*
The *level* of *E* is *3*
The *height* of the tree is *4*
The *degree* of node *B* is *2*
The *degree* of the tree is *3*
The *ancestors* of node *I* is *A, C, H*
The *descendants* of node *C* is *F, G, H, I*



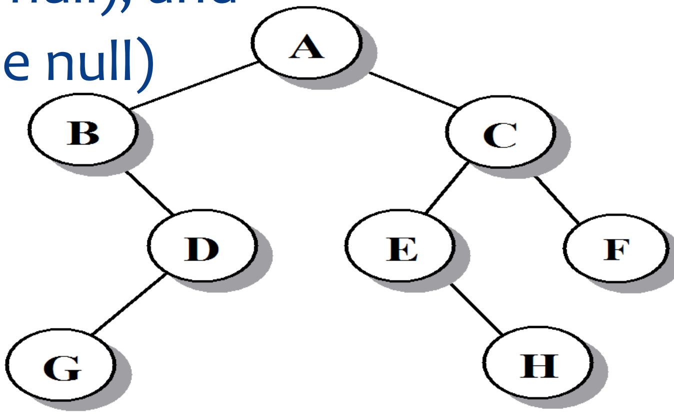
Applications of Trees

- Most decision-making process can be represented as a binary tree. At each node of the tree a yes/no decision is made on some issue.
- Storing naturally hierarchical data: File system
- Computer chess games build a huge tree (training) which they prune at runtime using heuristics to reach an optimal move.
- Syntax Trees Constructed by compilers and (implicitly) calculators to parse expressions.
- Huffman Coding Tree used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.

- Telephone exchanges used a tree hierarchy to find the actual target phone when dialing a phone number, for example. It is again not a binary tree, but a "decimal" tree with 10 nodes coming off each individual node.

Binary Tree

- A binary tree is composed of **zero** or more nodes
- Each node contains:
  - A **value** (data item)
  - A reference or pointer to a **left child** (may be null), and
  - A reference or pointer to a **right child** (may be null)
- A binary tree may be ***empty*** (contain no nodes)
- If not empty, a binary tree has a root node
  - Every node in the binary tree is reachable from the root node by a ***unique* path**

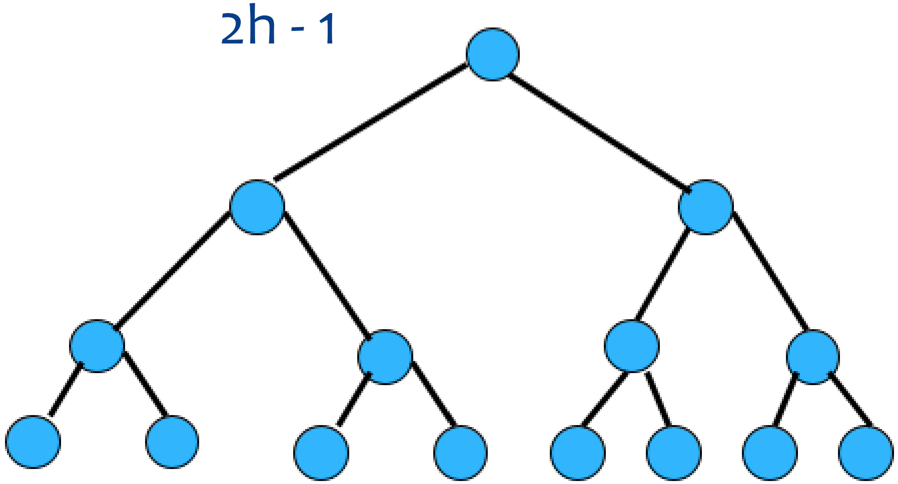


Minimum & Maximum Number Of Nodes

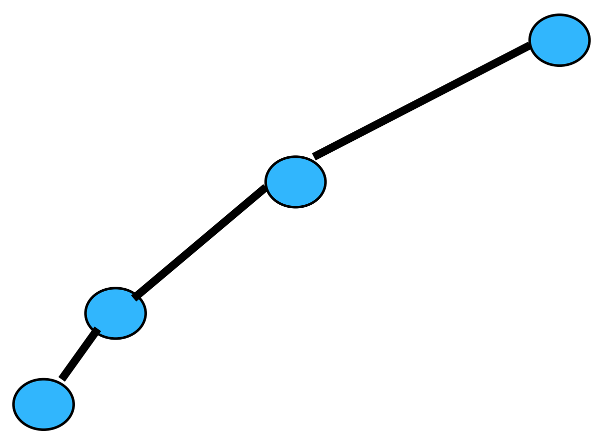
- All possible nodes at first h levels are present
- Maximum number of nodes

  1 + 2 + 4 + 8 + … + 2h-1

  2h - 1

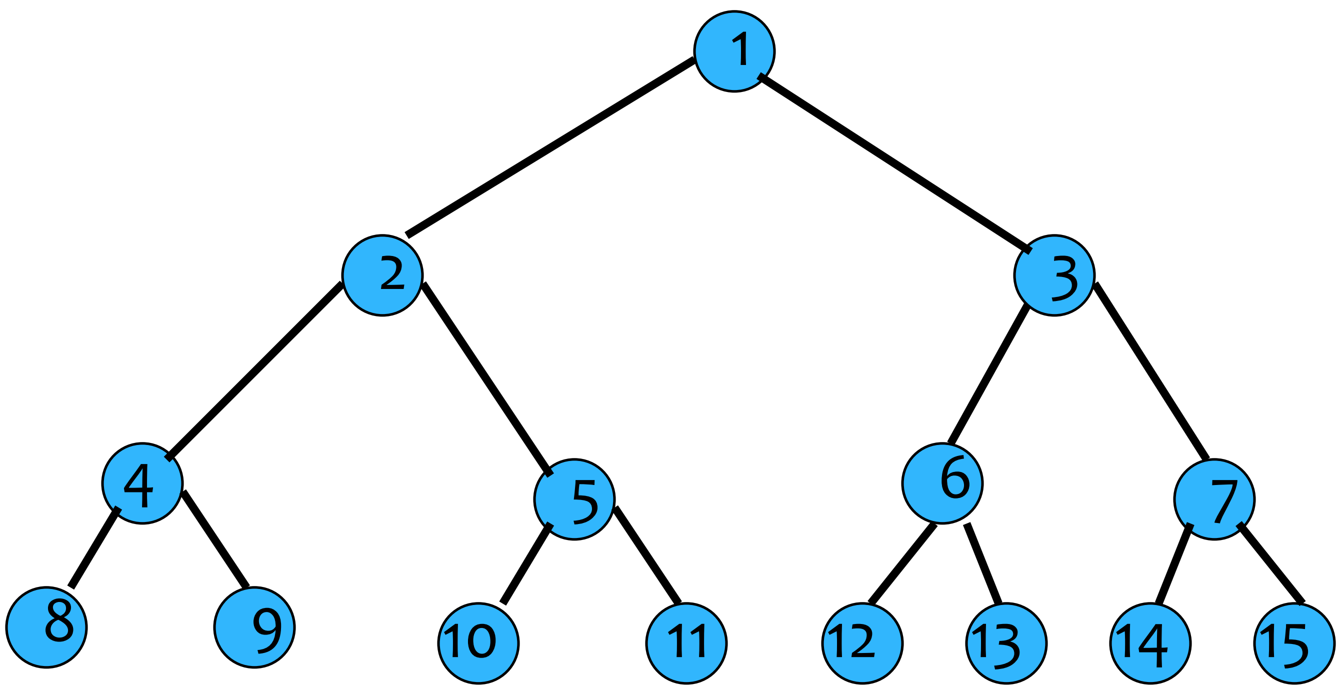


- Minimum number of nodes in a binary tree whose height is **h**
- At least one node at each of first **h** levels

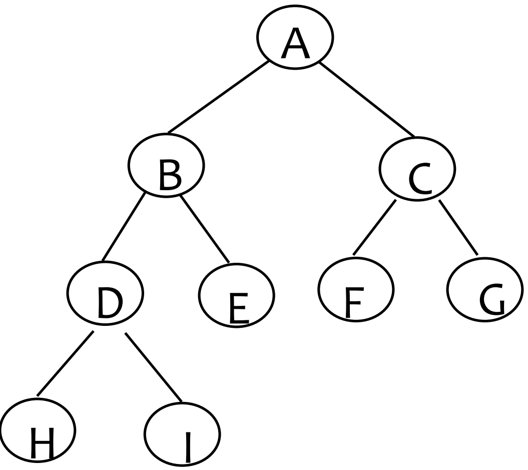


Numbering Nodes In Binary Tree

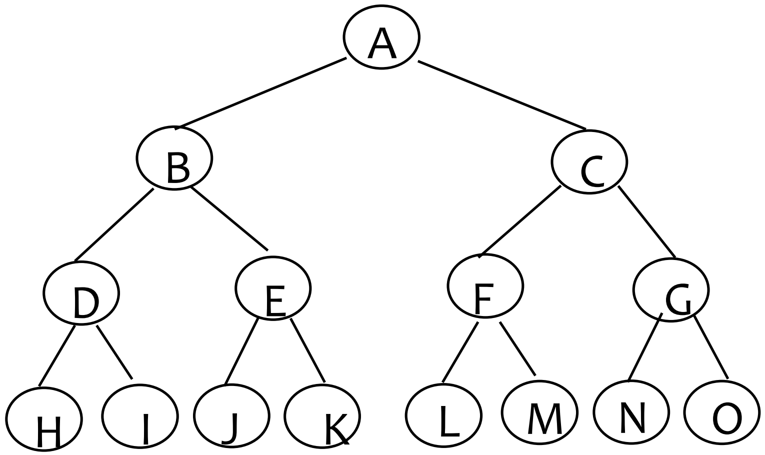
- Number the nodes 1 through $\mathbf{2^h - 1}$
- Number by levels from **top to bottom**
- Within a level number from **left to right**



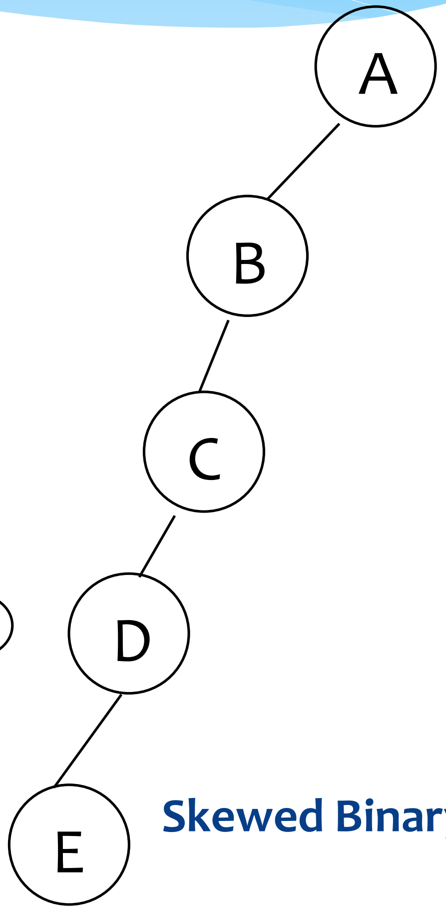
Types of Binary Trees



Complete binary tree



Full binary tree of depth 4



Skewed Binary Tree

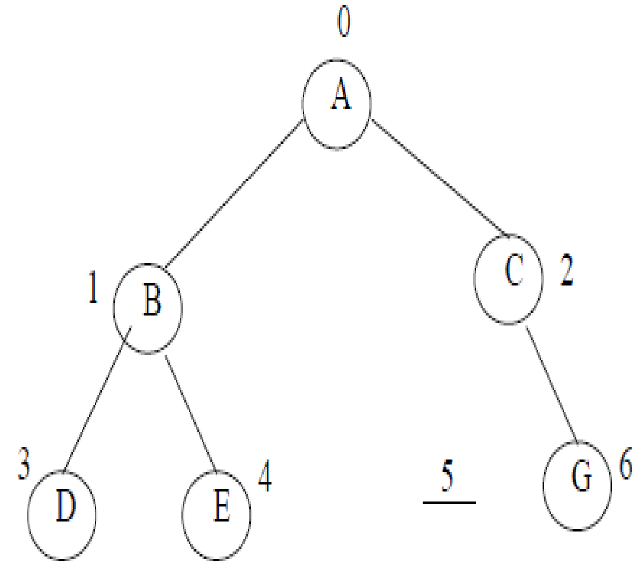
Binary Tree Representation

- There are two ways of representing binary tree in memory :

  1. Sequential representation using arrays
  2. Linked list representation

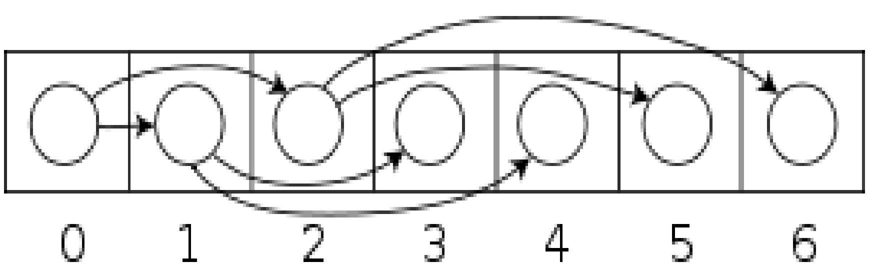
Array Representation

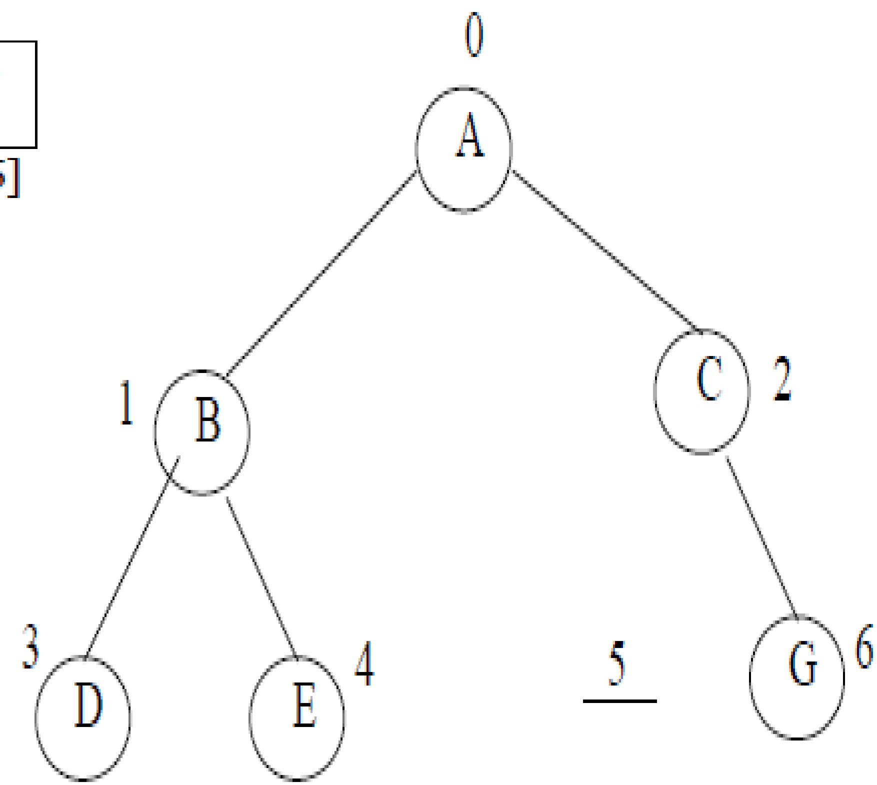
- An array can be used to store the nodes of a binary tree
- The nodes stored in an array of memory can be accessed sequentially
- Suppose a binary tree T of depth *d*
- *Then at most* $\mathbf{2^d - 1}$ *nodes can be there in T (i.e* **SIZE =** $\mathbf{2^d - 1}$*), so the array of size "SIZE" to represent the binary tree*
- *Consider a binary tree* of depth 3
- Then SIZE = $2^3 - 1 = 7$
- Then the array A[7] is declared to hold the nodes



Array Representation

A[ ]	A	B	C	D	E		F
	[0]	[1]	[2]	[3]	[4]	[5]	[6]





Array Representation

- To perform any operation often we have to identify the father, the left child and right child of an arbitrary node

1. The **father of a node** having index n can be obtained by ***(n – 1)/2***

- *For example to find the* **father of D**, where array index ***n = 3***
- *Then the father nodes index can be obtained*

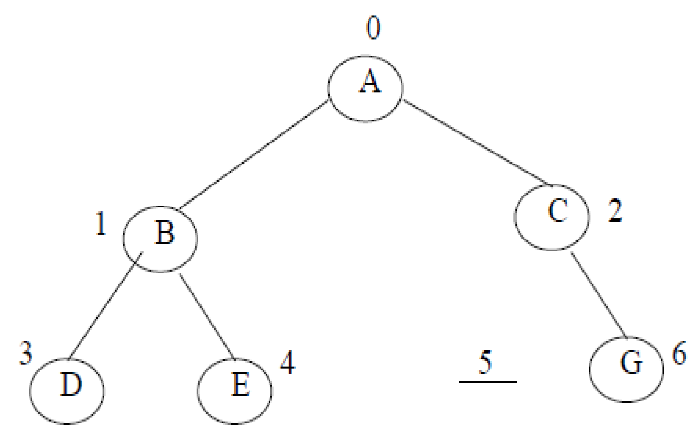
  = (*n* – 1)/2

  = 3 – 1/2

  = 2/2

  = 1

  i.e., A[1] is the father of D, which is B

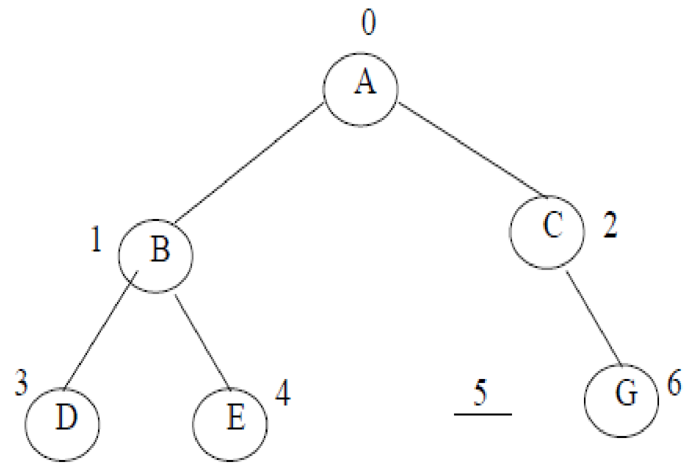


Array Representation

2. The **left child** of a node having index *n can be obtained by* ***(2n+1)***

- *For example* *to* find the **left child of C**, where array index ***n* = 2.** *Then it can be obtained by*

  $= (2n + 1)$
  $= 2*2 + 1$
  $= 4 + 1$
  $= 5$



- *i.e., A[5] is the left child of C, which is NULL. So no left child for C*

Array Representation

3. The **right child** of a node having array index *n can be obtained by* **(2n+ 2)**

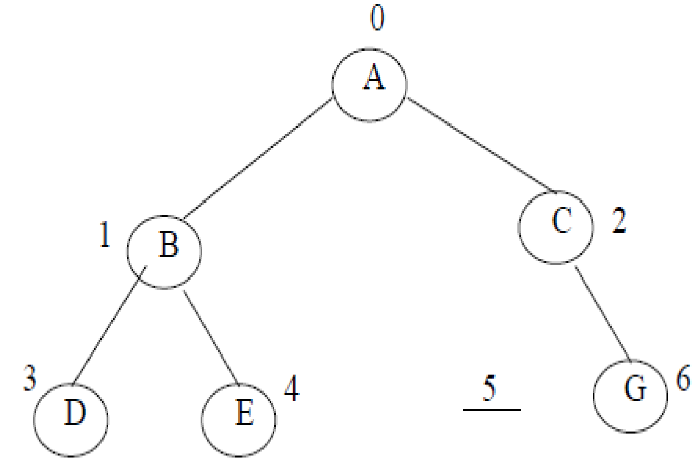
- For example to find the **right child of B,** where the array index ***n = 1.*** *Then*

  $= (2n + 2)$

  $= 2*1 + 2$

  $= 4$

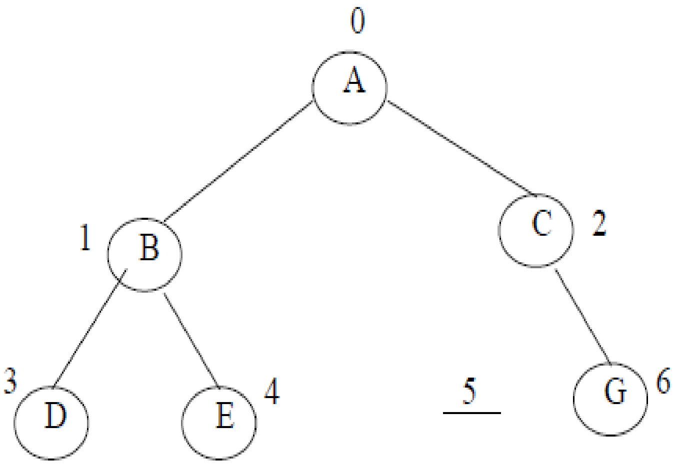
- *i.e., A[4] is the right child of B, which is E*



Array Representation

4. If the **left child** is at array index *n, then its right brother is at* ***(n + 1)***

Similarly, if the **right child** is at index *n, then its left brother is at* ***(n – 1)***



Binary Trees

- Binary tree representations (using array)
  - **Waste spaces**: in the worst case, a skewed tree of depth *k* **requires 2^k-1** spaces. Of these, only ***k*** spaces will be occupied

[1]	A
[2]	B
[3]	—
[4]	C
[5]	—
[6]	—
[7]	—
[8]	D
[9]	—
. . .	. . .
[16]	E

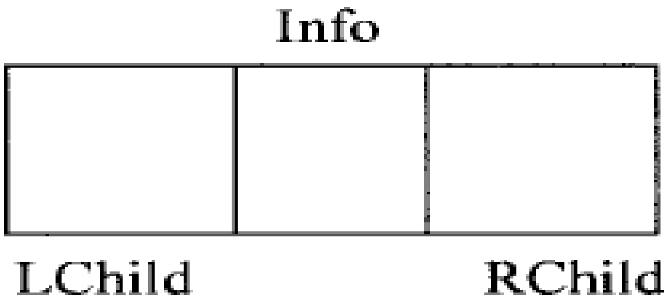
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Linked List Representation

- The most popular and practical way of representing a binary tree is using linked list (or pointers)

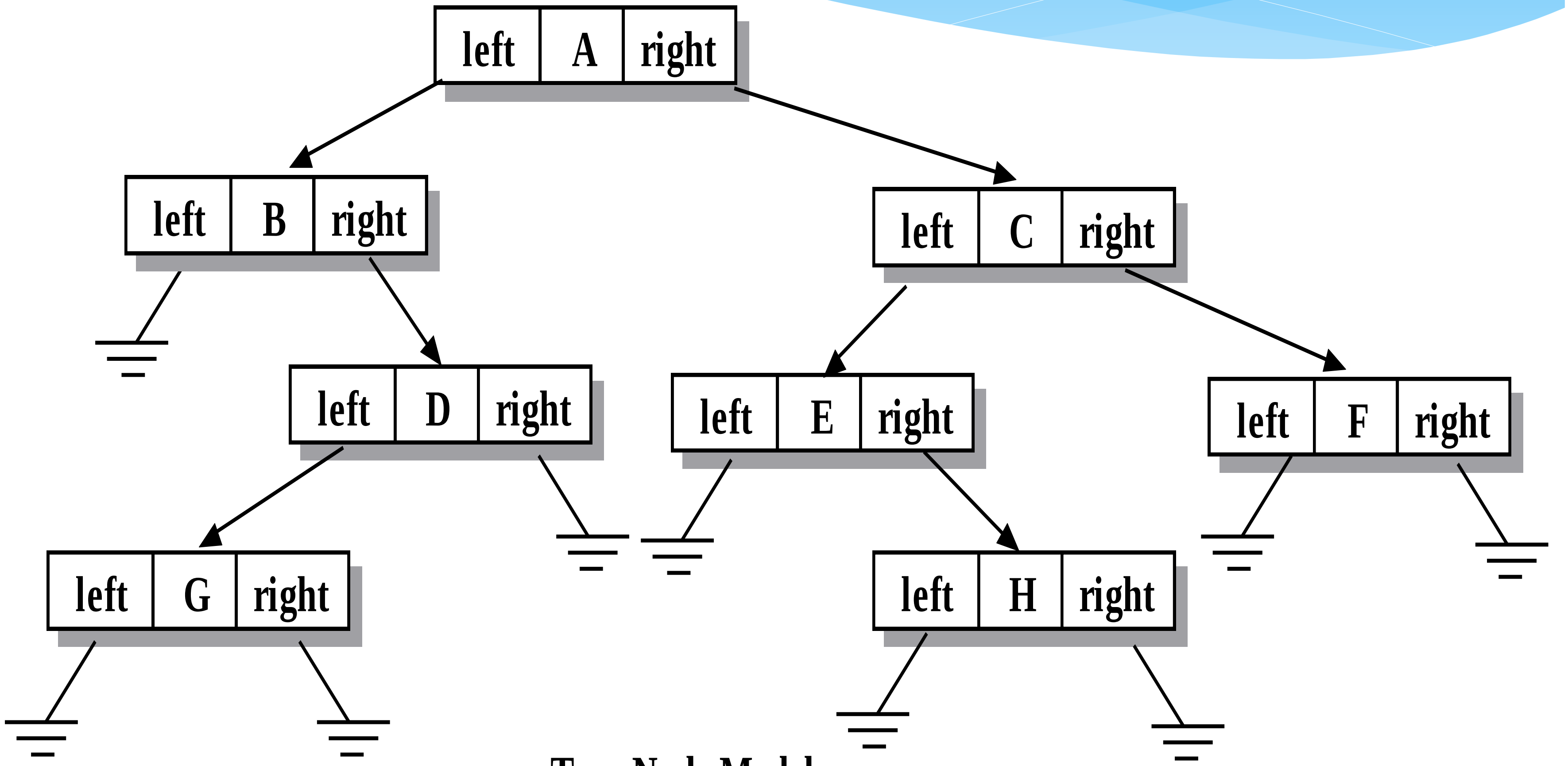
- In linked list, every element is represented as nodes. A node consists of **three fields** such as :

1. **Left Child (LChild)**
2. **Information of the Node (Info)**
3. **Right Child (RChild)**




```
struct Node
{
    int Info;
    struct Node *Lchild;
    struct Node *Rchild;
};
```

Linked List Representation



Tree Node Model

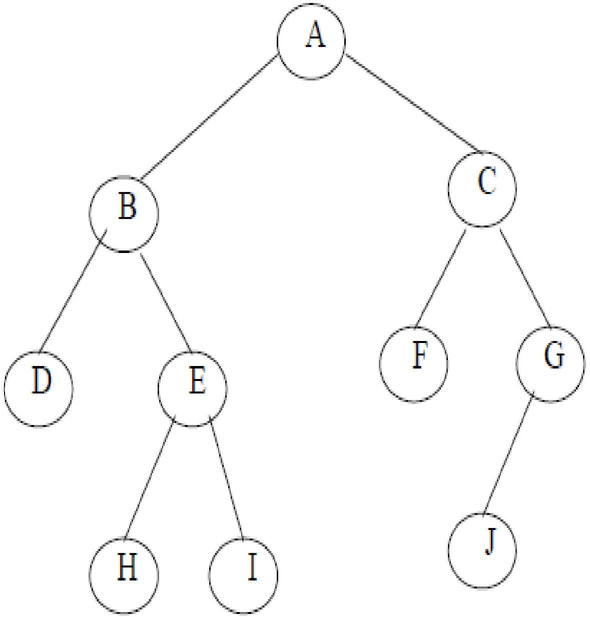
Traversing a Binary Tree

- At a given node, there are **three** things to do in some order:
  - **To visit the node itself**
  - **To traverse its left subtree**
  - **To traverse its right subtree**

- We can traverse the node **before** traversing either subtree

- Or, we can traverse the node **between** the subtrees

- Or, we can traverse the node **after** traversing both subtrees



- If we designate the task of visiting the root as R', traversing the left subtree as L and traversing the right subtree as R, then the three modes of tree traversal would be represented as:
  - **R'LR – Preorder**
  - **LRR' – Postorder**
  - **LR'R – Inorder**

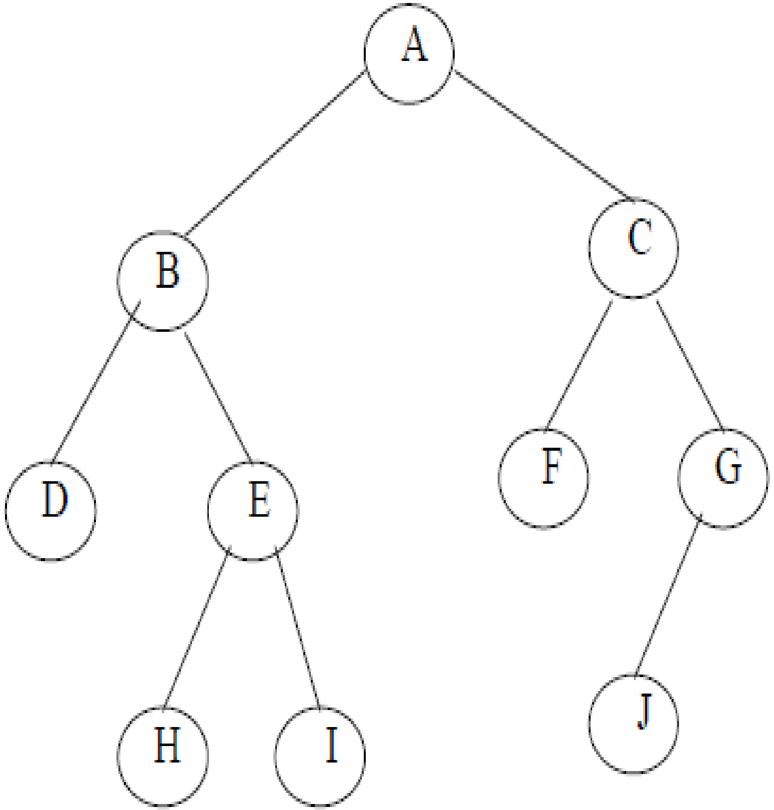
1. Pre Order Traversal (Node-left-right)

- To traverse a non-empty binary tree in pre order :

  1. Visit the root node

  2. Traverse the left sub tree in preorder

  3. Traverse the right sub tree in preorder



The preorder traversal is

A, B, D, E, H, I, C, F, G, J

Preorder Traversal

```
void preorder (p)
struct btreenode *p;
 {

  if ( p != null)                /* Checking for an empty tree */
   {
                                 /* print the value of the root node */
    printf("%d", p->info);
                                 /* traverse its left subtree */
    preorder(p->left);
                                 /* traverse its right subtree */
    preorder(p->right);
   }
 }
```

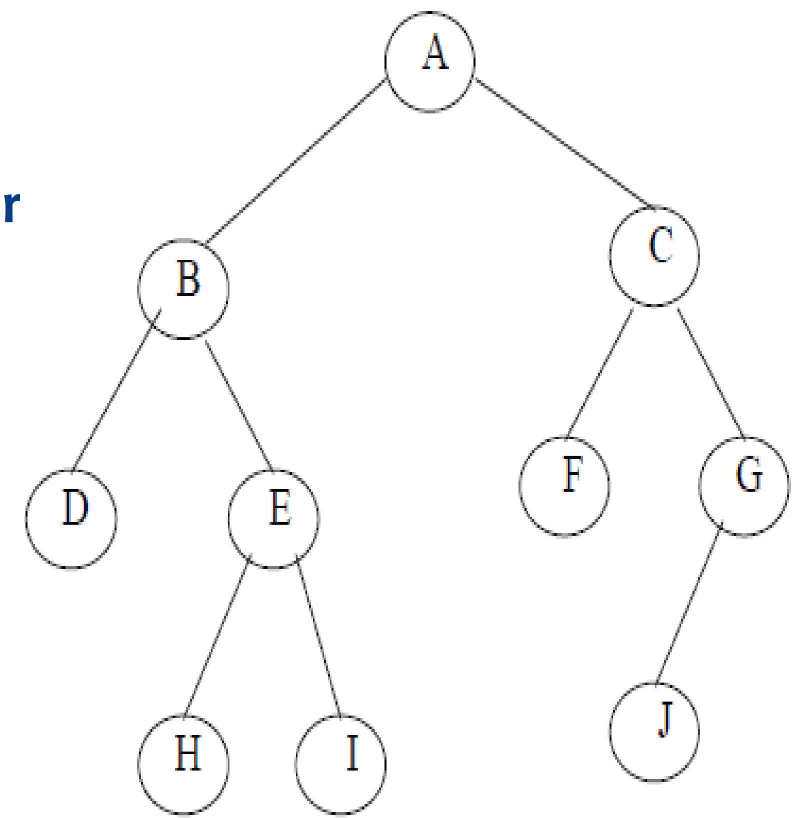
2. Post Order Traversal (Left-right-node)

- The post order traversal of a non-empty binary tree :

1. Traverse the left sub tree in post order

2. Traverse the right sub tree in post order

3. Visit the root node



The postorder traversal is

D, H, I, E, B, F, J, G, C, A

Postorder Traversal

```
void postorder(p)
struct btreenode *p;
 {
                                        /* checking for an empty tree */
  if (p != null)
   {
                                        /* traverse the left subtree */
    postorder(p->left);
                                        /* traverse the right subtree */
    postorder(p->right);
                                        /* print the value of root node */
    printf("%d", p->info);
   }
 }
```

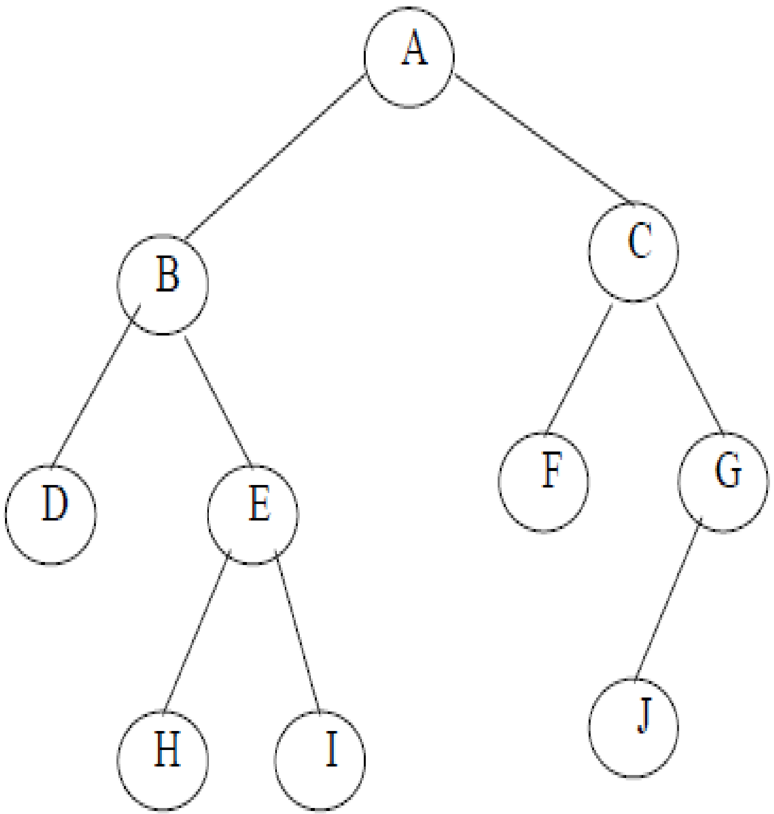
3. In order Traversal (Left-node-right)

- The in order traversal of a non-empty binary tree :

1. Traverse the left sub tree in order

2. Visit the root node

3. Traverse the right sub tree in order



The Inorder traversal is

D, B, H, E, I, A, F, C, J, G.

Inorder Traversal

```
void inorder(p)
struct btreenode *p;
{
                                        /* checking for an empty tree */
 if (p != null)
 {
                                        /* traverse the left subtree inorder */
   inorder(p->left);
                                        /* print the value of the root node */
   printf(“%d”, p->info);
                                        /*traverse right subtree inorder */
   inorder(p->right);
 }
}
```