

A ■ P ■ I ■ I ■ T

ASIA PACIFIC INSTITUTE OF
INFORMATION TECHNOLOGY

TREES

Topics Covered

Introduction to Trees

Binary Trees

Primitive Operations on Binary Trees

Application of binary trees

Tree Traversal

- * Inorder Traversal
- * Preorder Traversal
- * Postorder Traversal
- * Binary Search Tree

TREES

PREREQUISITE

Graph :- A graph $G=(V,E)$ consist of two possibly empty subsets V and E . V is the set of **vertices** (also none as nodes) and E is the set of **edges**.

$E_i = (v_j, v_k)$ denotes that there is an edge between the two vertices v_j and v_k .

Graph

- directed graph
- undirected graph
- acyclic

Tree - a special case of an undirected acyclic graph



TREES

Principle of Mathematical Induction.

(To prove the validity of a result)

- 1) Assume the result is true for a particular value of the variable.
- 2) Establish the fact that if the result is true for a particular value of the variable, then it is true for the next value of the variable.

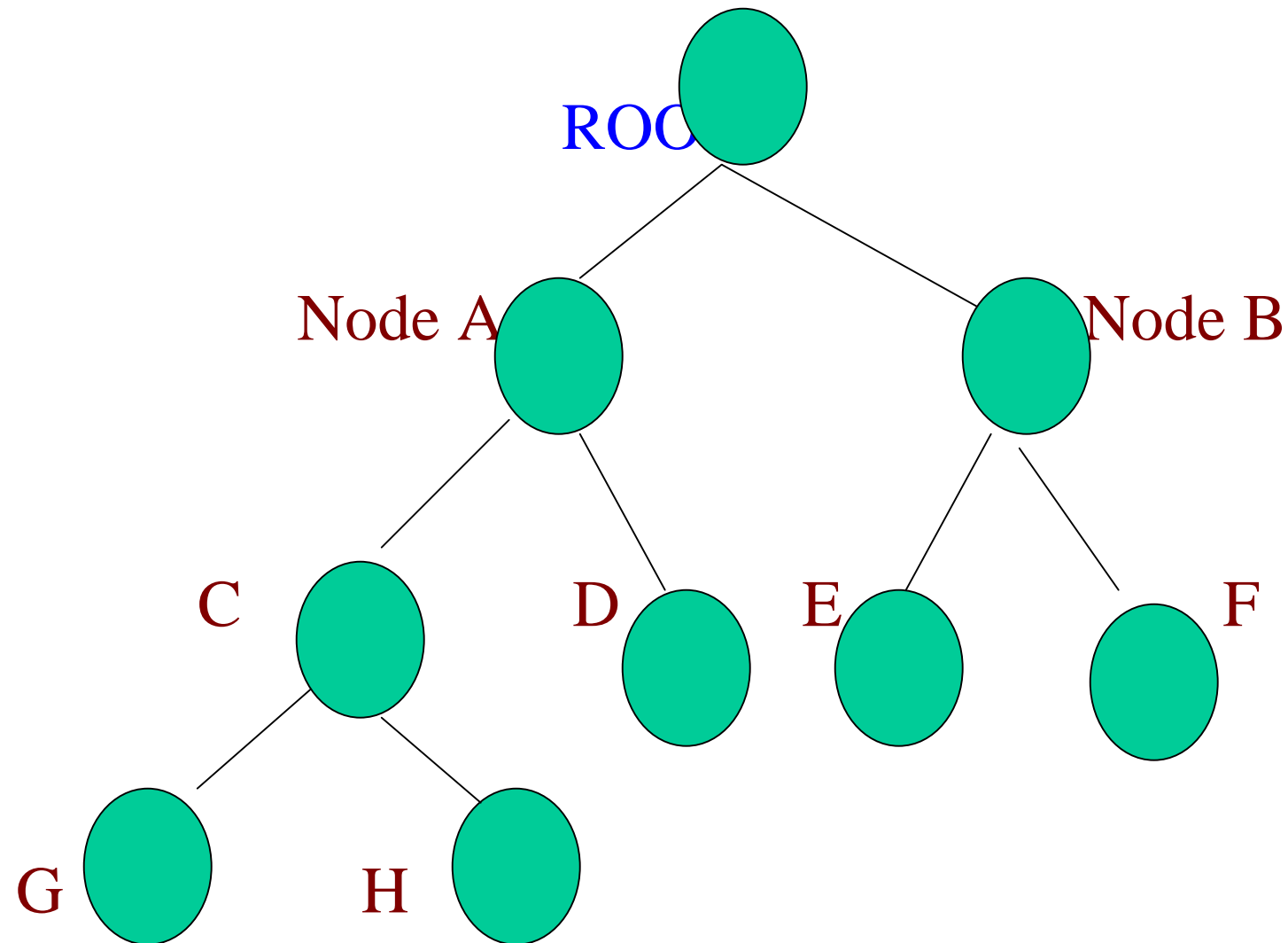
BINARY TREE

A tree in which each parent can have utmost two children.

A **Binary Tree** is the finite set of elements that is either **empty** of its 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 SUBTREE** and **RIGHT SUBTREE** of the original tree.

BINARY TREE

Identify the root, left and right subtrees of the following binary binary trees.



BINARY TREE

Definitions

Father :

If A is the root of a Binary Tree and B is the Root of the left or right subtree of A then A is said to be the father of B

Left Son :

If A is the root of a Binary Tree and B is the root of the left sub-tree of A then B is said to be the left son of A.

BINARY TREE

Definitions

Ancestor :

Node n_1 is an ancestor of node n_2 if n_1 is either the father of n_2 or n_1 is the father of some ancestor of n_2

Descendant :

Node n_2 is a descendant of n_1 if n_2 is either the son of n_1 or n_2 is the son of some descendant of n_1 .

Left Descendant :

A node n_2 is the left descendant of node n_1 if n_2 is either the left son of n_1 or the descendant of the left son of n_1 .

BINARY TREE

Definitions

Right Descendant :

A node n_2 is the right descendant of node n_1 if n_2 is either the right son of n_1 or the descendant of the son of n_1 .

Strictly Binary Tree:

A Binary Tree in which every nonleaf node has nonempty left and right subtrees is called a strictly binary tree.

BINARY TREE

Definitions

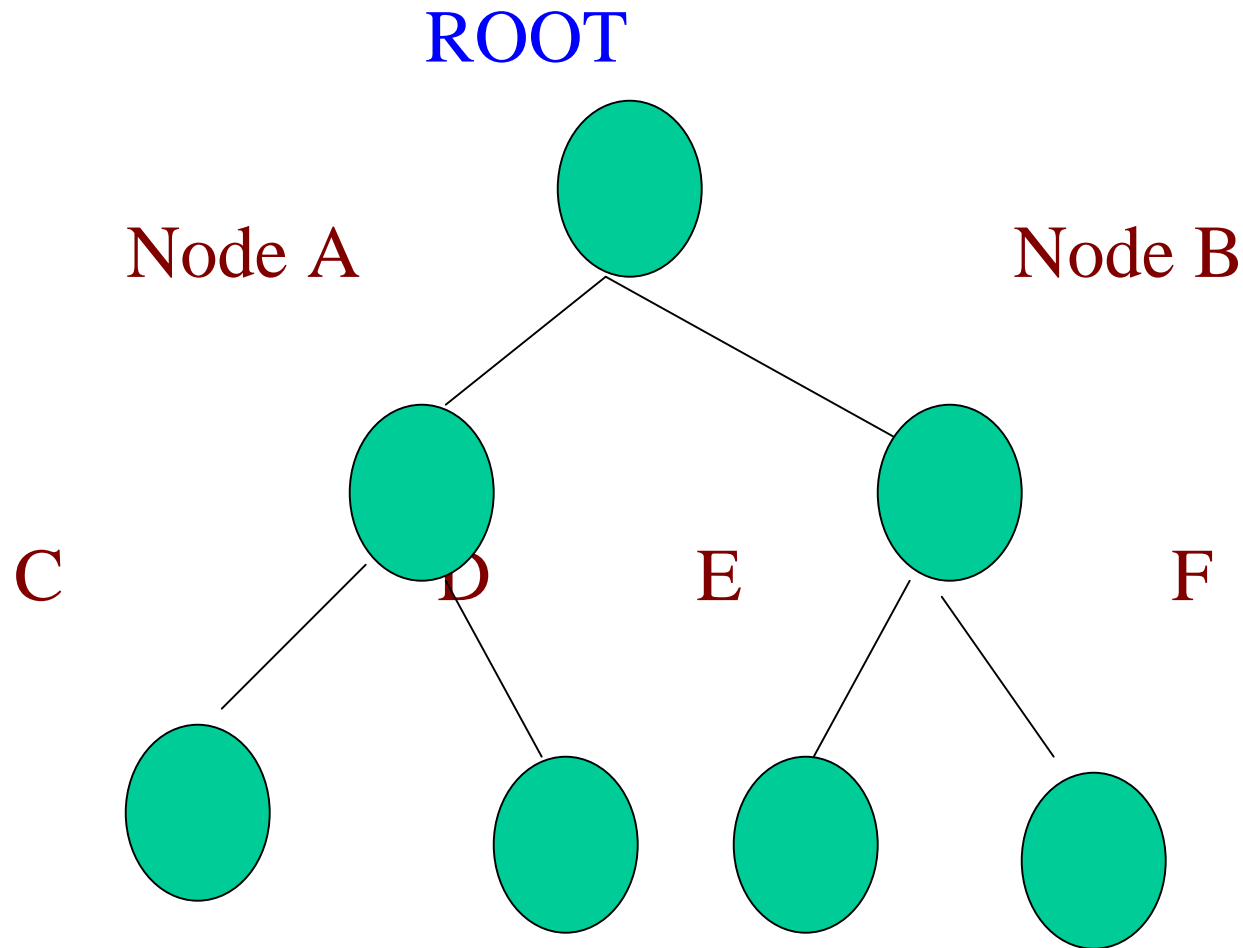
Right Descendant :

A node n_2 is the right descendant of node n_1 if n_2 is either the right son of n_1 or the descendant of the son of n_1 .

Strictly Binary Tree:

A Binary Tree in which every nonleaf node has nonempty left and right subtrees is called a strictly binary tree.

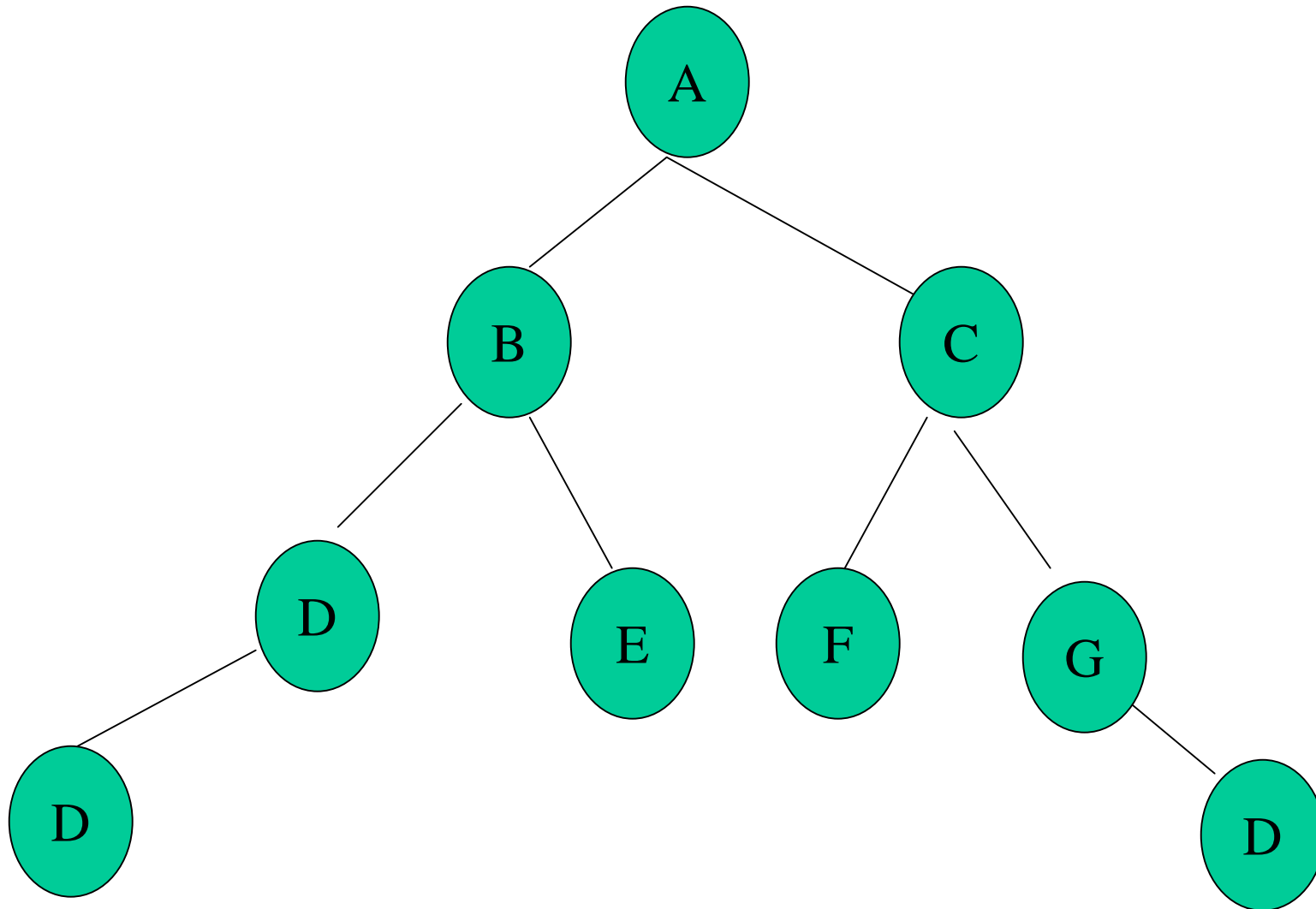
Strictly BiNaRy TeRe



Note : A strictly binary tree with n leaves always contains $2*n-1$ nodes (Proof ?)

Is it a Strictly BiNaRy TeRe ?

ROOT



Definition Continue

Level :

The level of a non-root in a binary tree is one more than the level of its father. The level of the **root = 0**. A binary tree can have utmost 2^l nodes at level l .

Depth :

The **depth of a binary tree** is the **maximum of the levels of the nodes in the tree** (also maximum of the levels of leaf nodes). This is the **same as the length of the longest path** from the root to any leaf.

Definition Continue

Complete Binary Tree :

A complete binary tree of depth d is the strictly binary tree all of whose leaves are at level d . A complete binary tree contains exactly 2^l nodes at each level l

where

$$0 \leq l \leq d.$$

A complete binary tree of depth d is the binary tree of depth d that contains exactly 2^l nodes at each level l where $0 \leq l \leq d$.

Definition Continue

The total number of nodes in a complete binary tree of depth d , say tn , is the sum of the number of nodes at each level between 0 and d .

d

$$tn = \sum_{0}^{d} 2^j = 2^{(d+1)} - 1$$

Note :

if the number of nodes in a complete binary tree, tn , is known we can compute its depth, d , from the equation

$$tn = 2^{(d + 1)} - 1$$

$$d = \log_2(tn+1) - 1$$

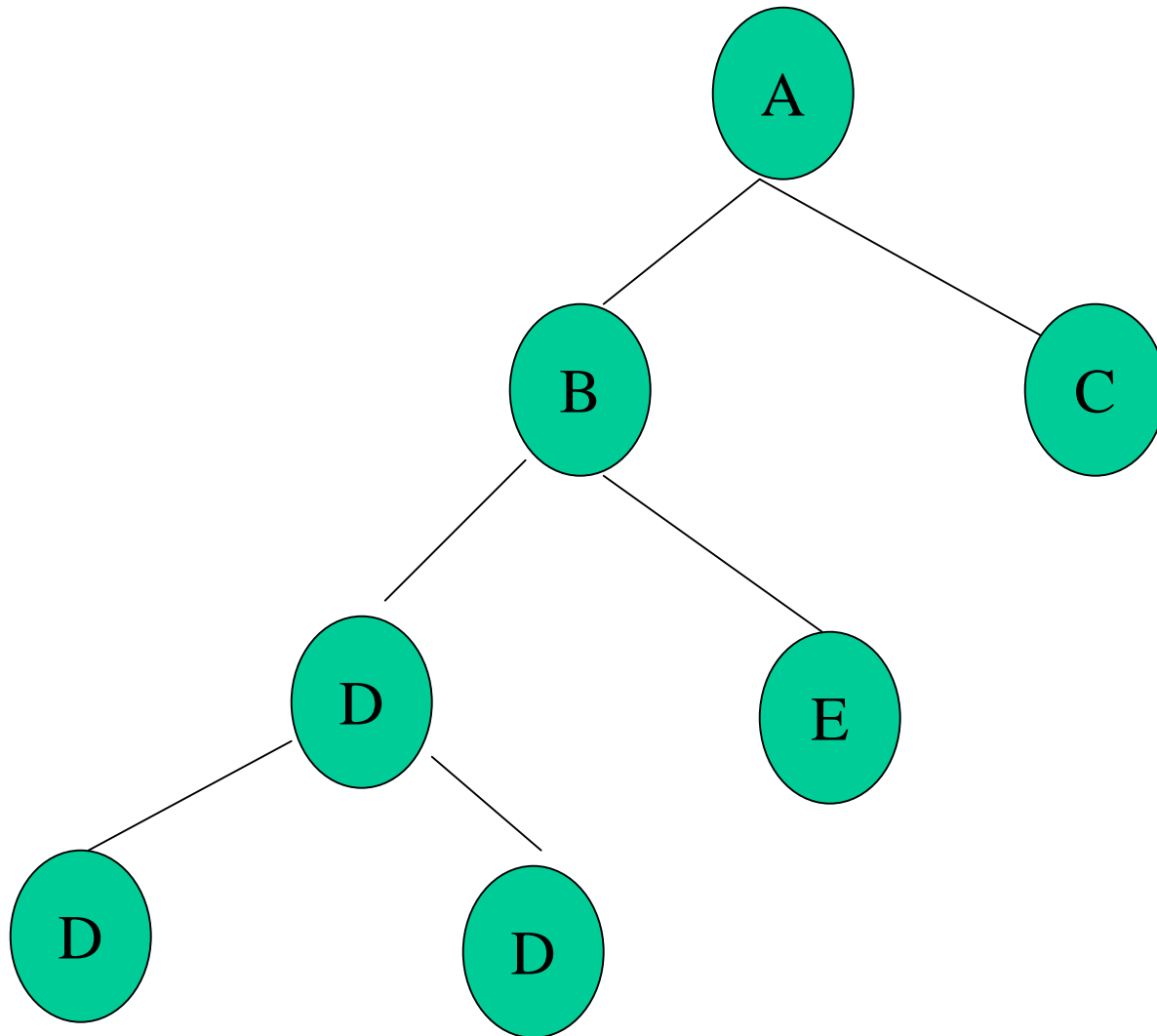
Almost Complete Binary Tree :

A binary tree of depth d is an almost complete binary tree if

- 1) Each leaf in the tree is either at level d or at level $d-1$
- 2) For any node nd in the tree with right descendant at level d , nd must have a left son and every left descendant of nd is either a leaf at d or has two sons.

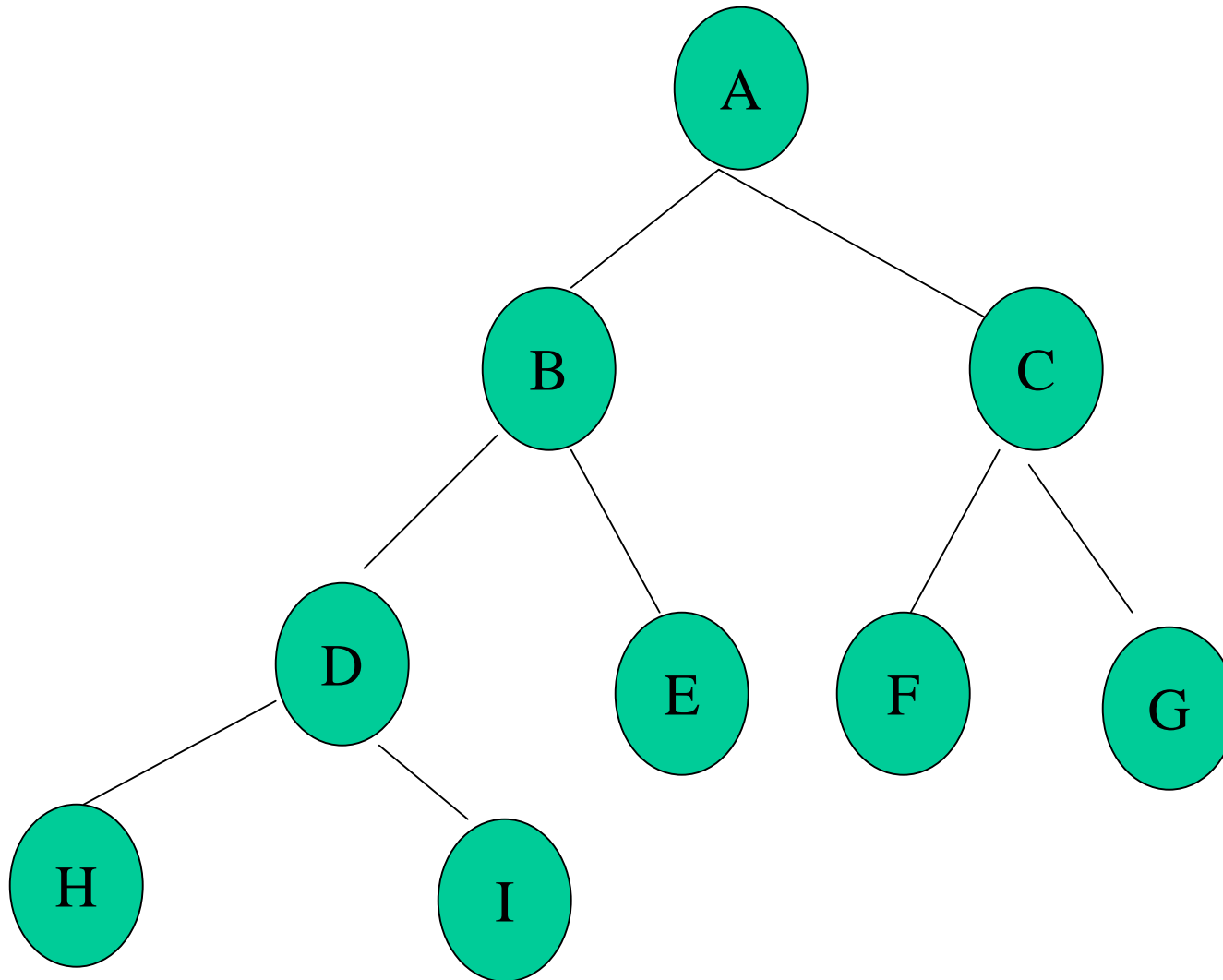
Not an Almost Complete Binary Tree

ROOT



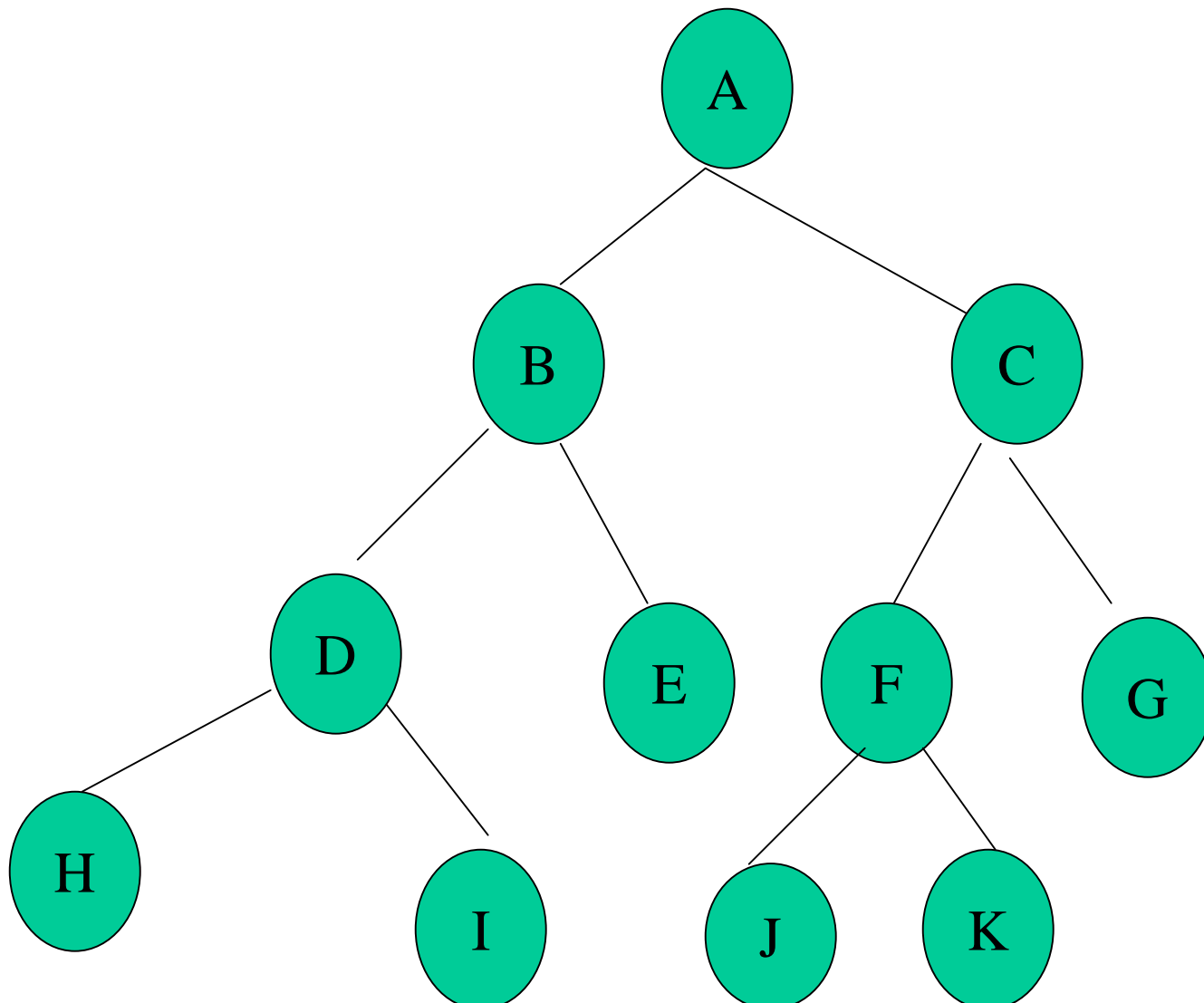
An Almost Complete Binary Tree

ROOT



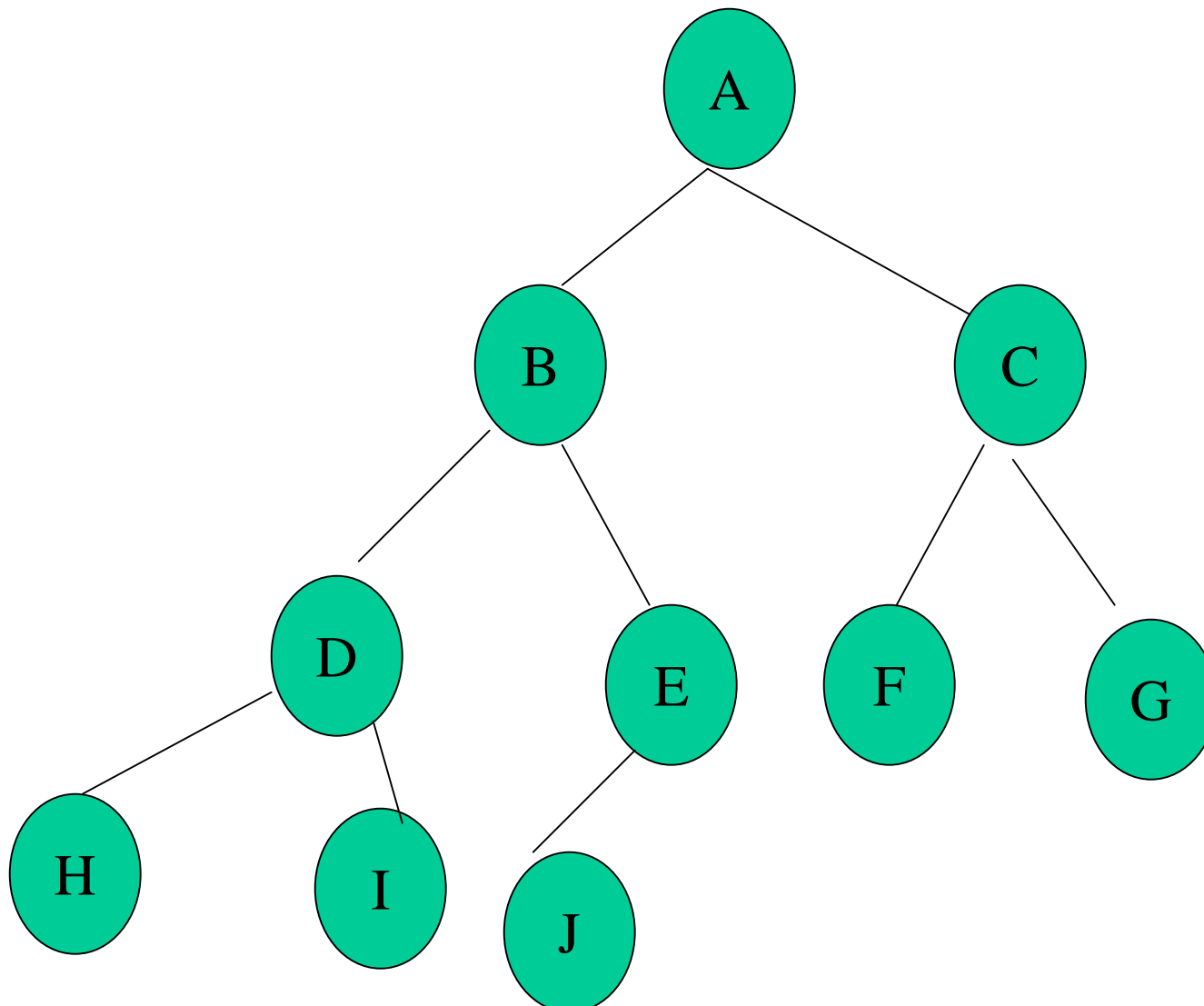
Not an almost complete binary tree

ROOT



An almost complete binary tree

ROOT



OPERATIONS ON BINARY TREE :

Notations :-

Let p be the pointer to the node nd of a binary tree

- info(p) - returns the contents of node p
- left(p) - returns the pointer to the left son of node p
- right(p) - returns the pointer to the right son of node p
- father(p) - returns the pointer to the father of node p
- brother(p) - returns the pointer to the brother of node p

```
Isleft(p)
{
NODEPTR q;
q = father(p);
if( q == NULL)
    return(FALSE);
if(left(q) == p )
    return (TRUE);
return(FALSE);
}
```

This is same as $\text{father}(p) \ \&\& \ (\ p == \text{left}(\text{father}(p)))$
 $\text{isright}(p) \ ?$

P = maketree(x) :-

which creates a binary tree with a single node pointed to by p and stores x in that node. Maketree returns a pointer p.

setleft(p,x) :-

Creates a new left son of node(p) with information field x.

setright(p,x) :-

Creates a new right son of node(p) with information field x.

APPLICATION OF BINARY TREES

To eliminate the duplicate from a series of objects.

Consider the series :

14,15,4,9,7,18,3,5,16,4,20,17,9,14,5

The first number in the list is placed in a node that is created as the root of a binary tree.

Each successive number in the list is compared with the number in the root.

If it matches, we have a duplicate.

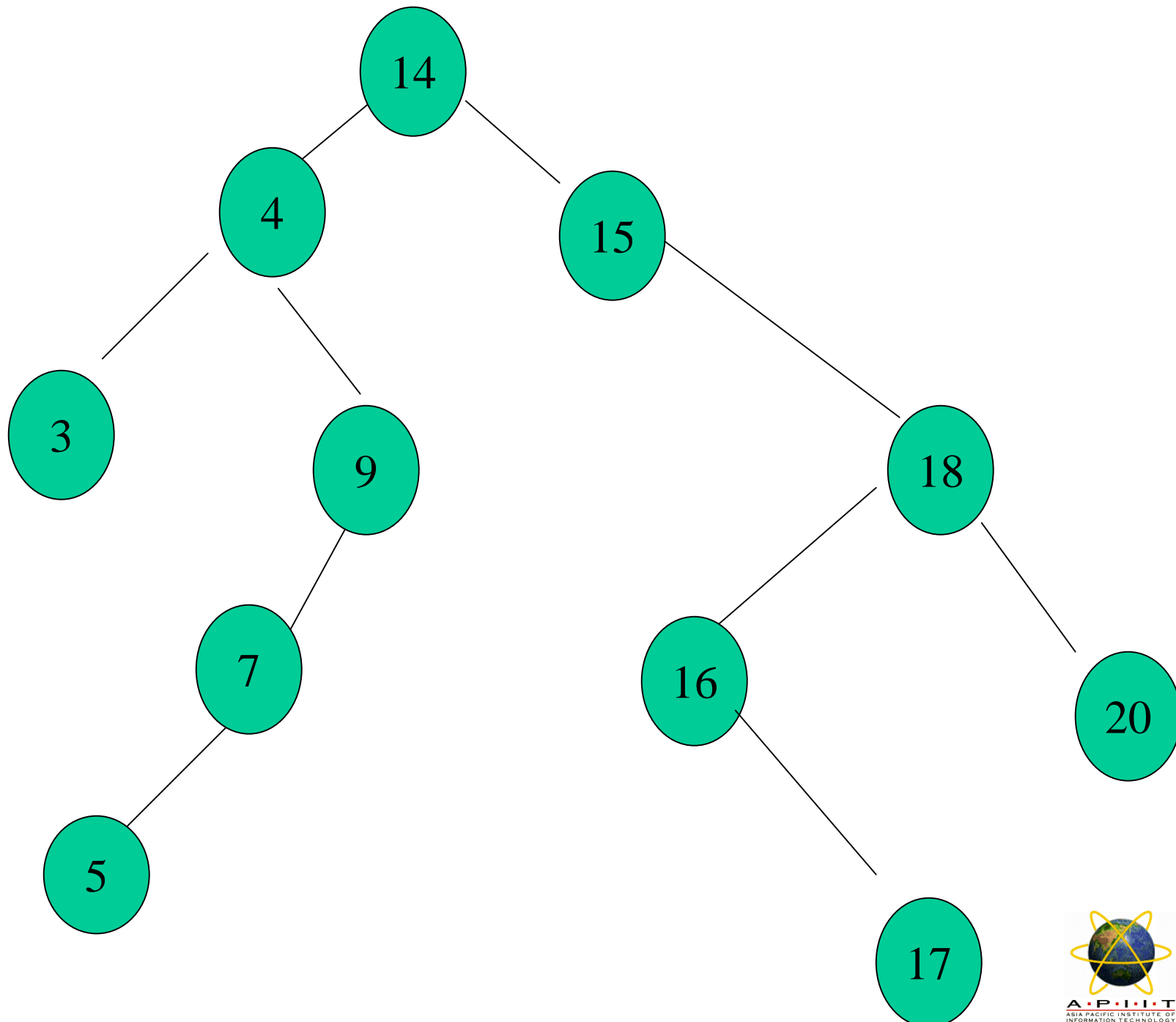
If it is smaller, we examine the left subtree recursively.

If it is larger, we examine the right subtree recursively.

APPLICATION OF BINARY TREES

14,15,4,9,7,18,3,5,16,4,20,17,9,14,5

During this process , if we come across an empty subtree, the number is not a duplicate and is placed into a new node at that position in the tree



```
Scanf(“%d”, &number);  
tree = maketree(number);  
while ( there are numbers left in the input)  
{  
    scanf(“%d”,&number);  
    p = q = tree;  
    while (number != info(p) && q != NULL)  
        {  
            p = q;  
            if(number < info(p))  
                p = left(p);  
            else  
                q = right(p);  
        }  
}
```

```
if(number == info(p))  
    printf(“%d %s \n”,number ,”is a duplicate”);  
else    if(number < info(p))  
        setleft(p,number);  
        else setright(p,number);  
}
```

Tree Traversals

Preorder Traversal

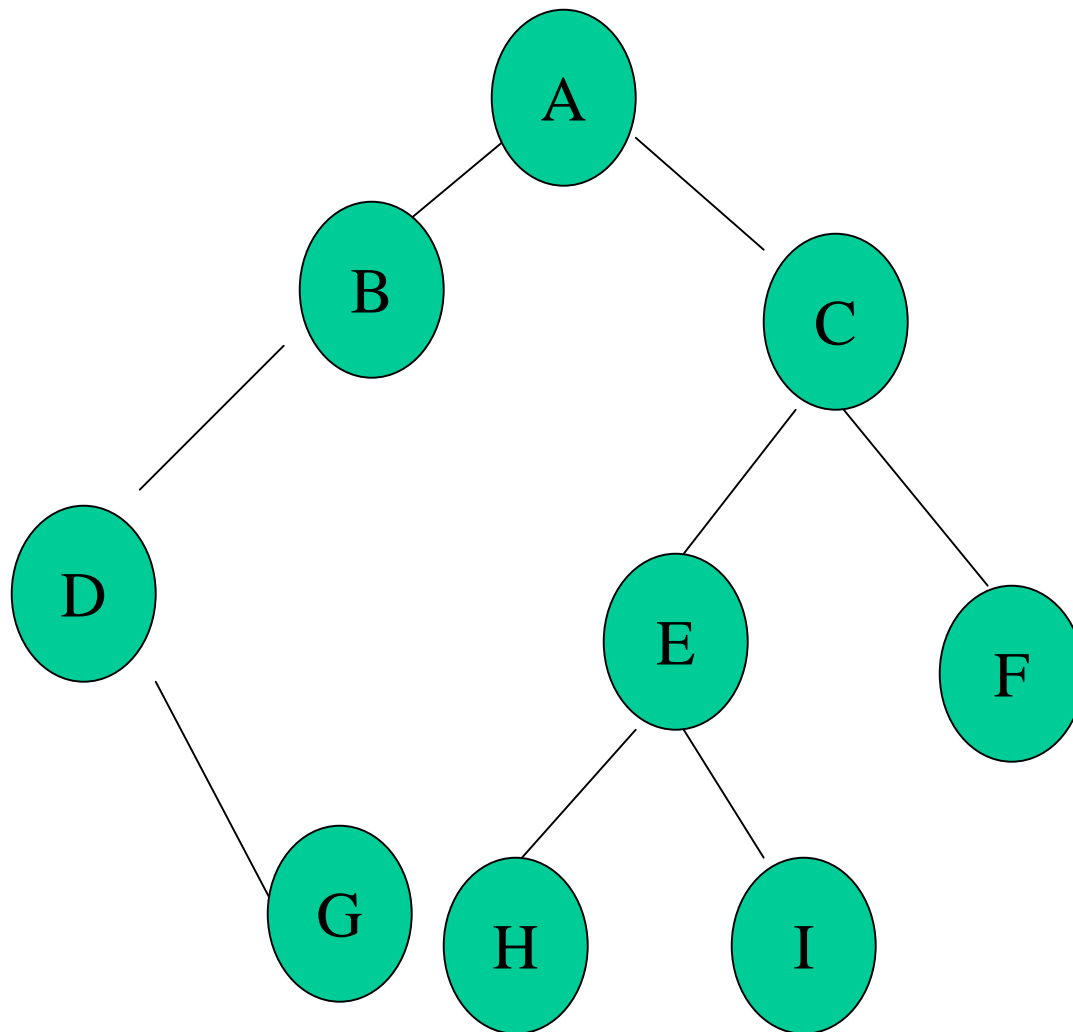
- 1) Visit the root
- 2) traverse the left subtree in preorder
- 3) traverse the right subtree in preorder

Inorder Traversal

- 1) traverse the left subtree in inorder
- 2) Visit the root
- 3) traverse the right subtree in inorder

Postorder Traversal

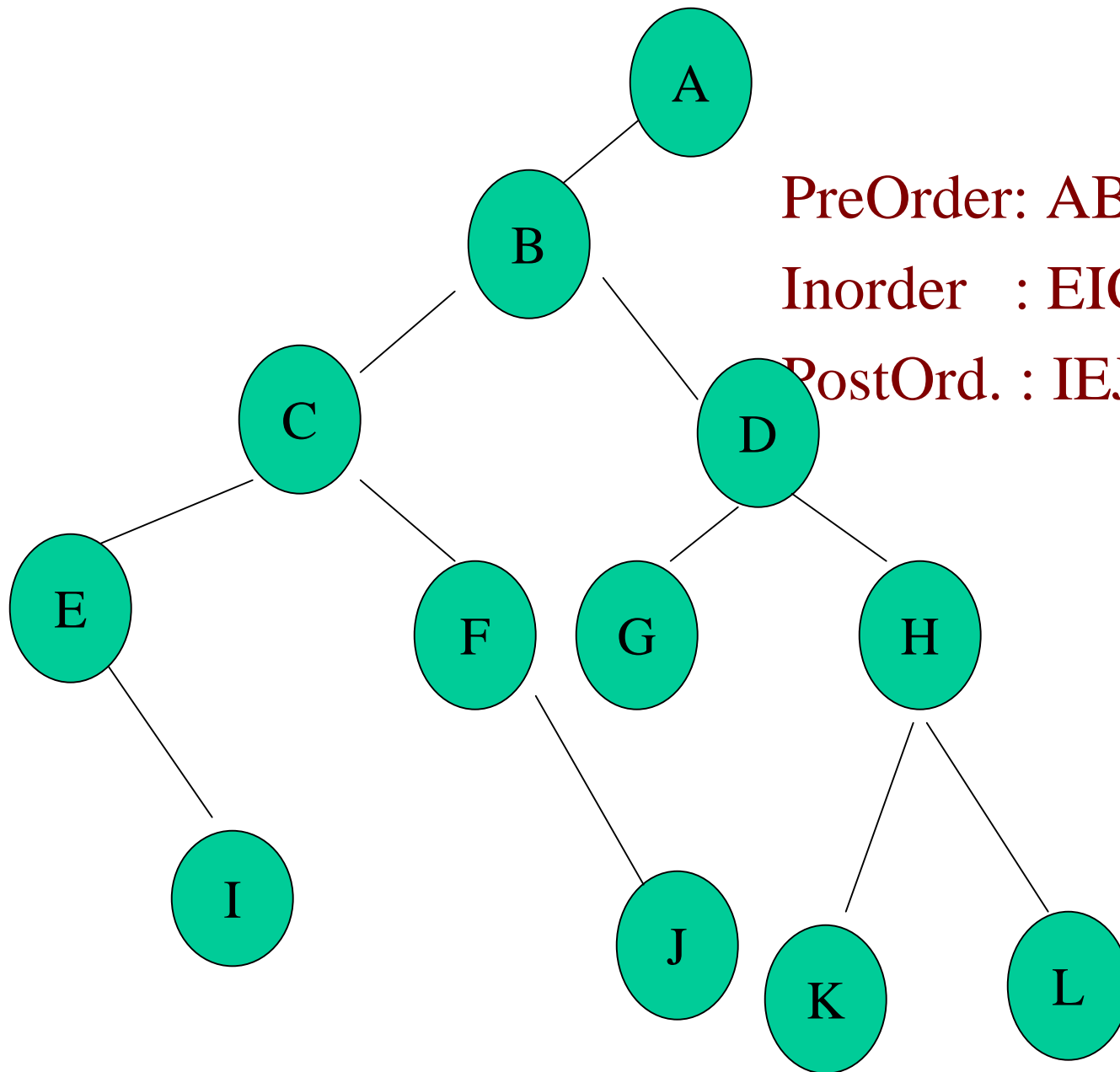
- 1) traverse the left subtree in postorder
- 2) traverse the right subtree in postorder
- 3) Visit the root



PREORDER : A B D G C E H I F

INORDER : D G B A H E I C F

POSTORDER : G D B H I E F C A



PreOrder: ABCEIFJDGHKL

Inorder : EICFJBGDKHLA

PostOrd. : IEJFCGKLHDBA

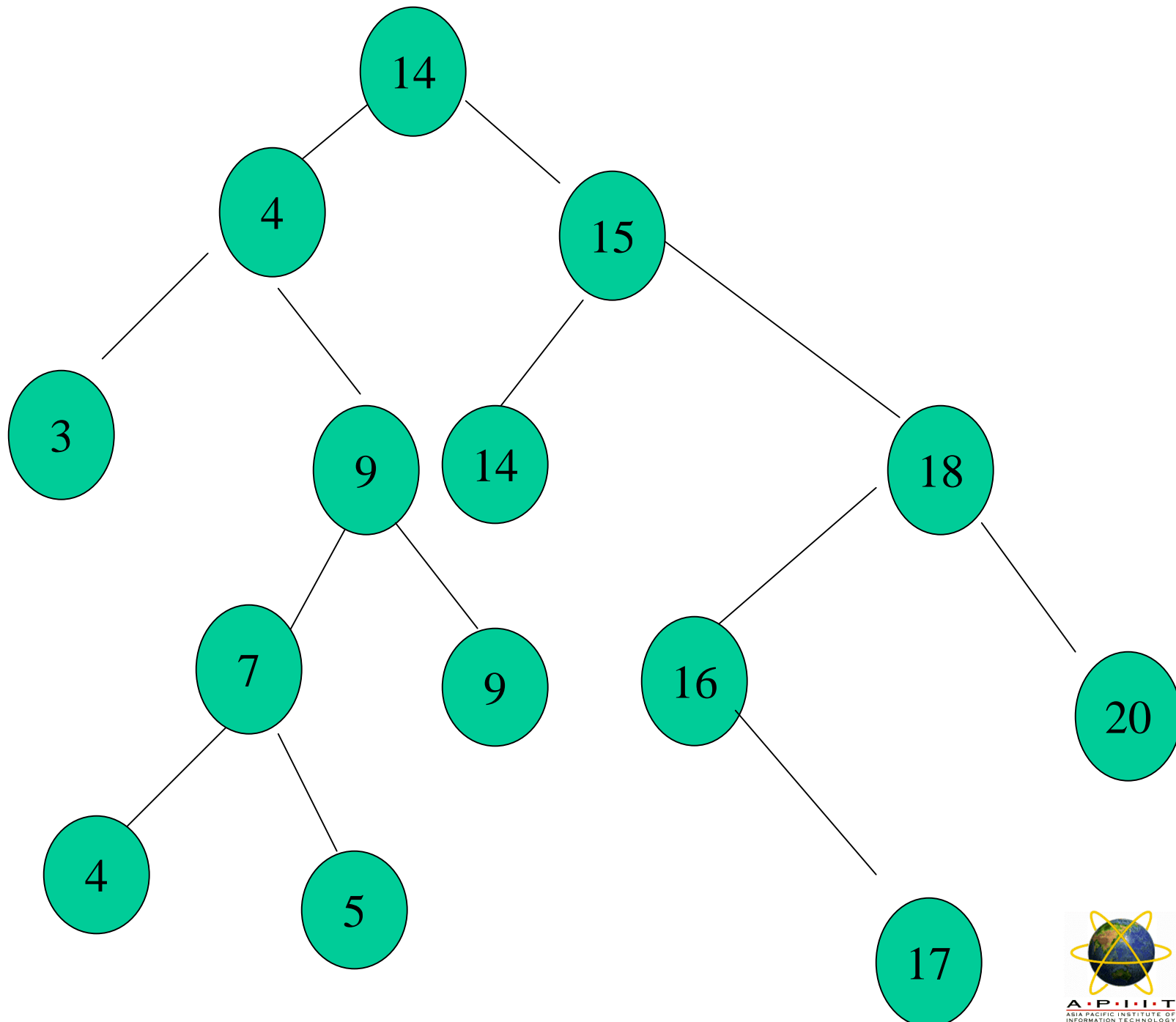
Binary Search Tree :

A **Binary Tree** in which **all elements** in the **left subtree** of a node **n** in the tree **are less than** the contents of **n**, and **all elements in the right subtree** of **n** **are greater than or equal** to the contents of **n** [Duplicate area allowed]

A binary search tree can be used to sort a series of numbers. Consider the following series of numbers :

14,15,4,9,7,18,3,5,16,4,20,17,9,14,5

Construct a Binary Search Tree using these numbers (allow duplicates)



If this tree is traversed in inorder (LEFT, ROOT, RIGHT) and print the contents as we visit each node, then the resulting series will be in sorted order.

Implementation of Binary Trees :

```
struct nodetype
```

```
{
```

```
    int info;
```

```
    struct nodetype *left;
```

```
    struct nodetype *right;
```

```
};
```

```
typedef struct nodetype *NODEPTR;
```

Implementation of Binary Trees :

maketree(x) Function

```
NODEPTR maketree(x);  
int x;  
{  
    NODEPTR p;  
    p = getnode();  
    p --> info = x;  
    p --> left = NULL;  
    p --> right = NULL;  
    return(p);  
}
```

setleft function

__setleft(p,x)

NODEPTR p; int x;

{

if (p == NULL)

printf(“ Invalid Insertion”);

else

if (p --> left != NULL)

printf(“ Invalid Insertion”);

else

p --> left = maketree(x);

}

setright(p,x);

Constructing a Binary Search Tree

```
main() {  
NODEPTR ptree,p,q;  
int number;  
scanf("%d", &number);  
ptree = maketree(number);  
while( scanf("%d",&number) != EOF)  
{  
    p = q = ptree;  
    while( number != p --> info && q != NULL)  
    {  
        p = q;  
        if ( number < p -> info)  
            q = p --> left;  
        else    q = p --> right;  
    }  
}
```

```
if ( number == p --> info)
    printf(“%d is a duplicate \n”, number);
else if ( number < p --> info)
    setleft(p,number);
else
    setright(p,number);
}
}
```


Procedure for Traversing Binary Trees

Preorder Traversal :

```
pretrav(tree)
NODEPTR tree;
{
    if ( tree != NULL )
    {
        printf(“ %d \n”, tree --> info);        // Visit the root
        pretrav(tree --> left);                // traverse left subtree
        pretrav(tree --> right);               // traverse right subtree
    }
}
```

Procedure for Traversing Binary Trees

Inorder Traversal :

```
intrav(tree)
NODEPTR tree;
{
    if ( tree != NULL )
    {
        intrav(tree --> left);           // traverse left subtree
        printf(“ %d \n”, tree --> info); // Visit the root
        intrav(tree --> right);          // traverse right subtree
    }
}
```

Procedure for Traversing Binary Trees

Postorder Traversal :

```
posttrav(tree)
NODEPTR tree;
{
    if ( tree != NULL )
    {
        posttrav(tree --> left);           // traverse left subtree
        posttrav(tree --> right);          // traverse right subtree
        printf(“ %d \n”, tree --> info); // Visit the root
    }
}
```

THE END OF LESSON V