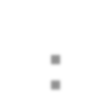
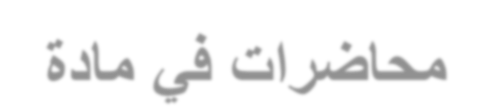
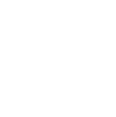
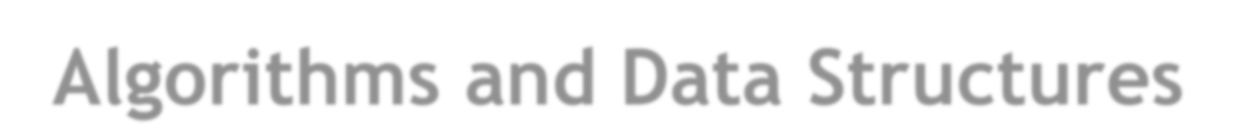
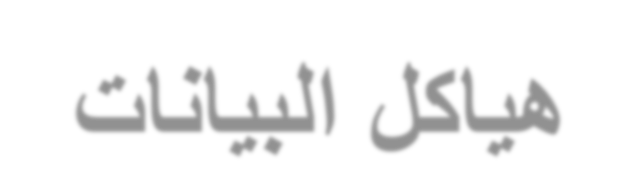
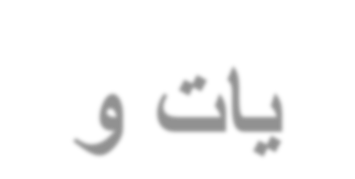
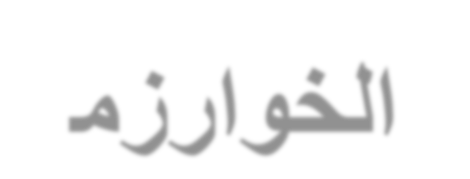
165



**Lecture # 8**

Binary Trees

Outline

• Binary Trees

 Strictly Binary Trees.

 Complete Binary Trees.

 Traversing Binary Trees:

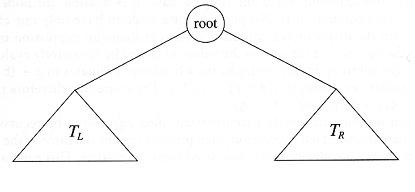
 Inorder traversal.

 Postorder traversal.

 Preorder traversal.

Binary Trees

• **Binary Tree** is a tree in which no node can have more than two children.



Generic binary tree

Binary Trees (cont…)

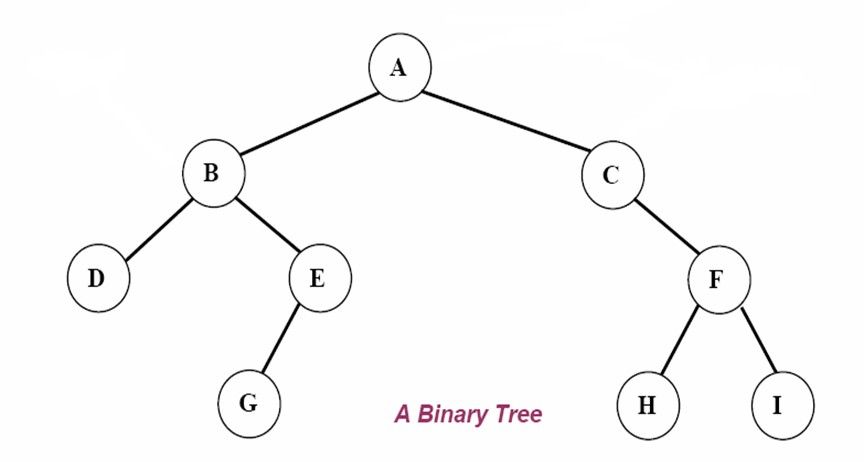
• The two subsets of the root are themselves binary trees called the

**left** and **right subtrees** of the original tree.

• A left or right subtree can be empty.

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

root



left subtree right subtree

Node Struct of Binary Tree

• Possible operations on the Binary Tree ADT

 insert, find, delete, print\_tree, etc.

• **Implementation**

 Because a binary tree has at most two children, we can keep direct pointers to them.

**struct BinaryNode**

**{**

**int element; // The data in the node**

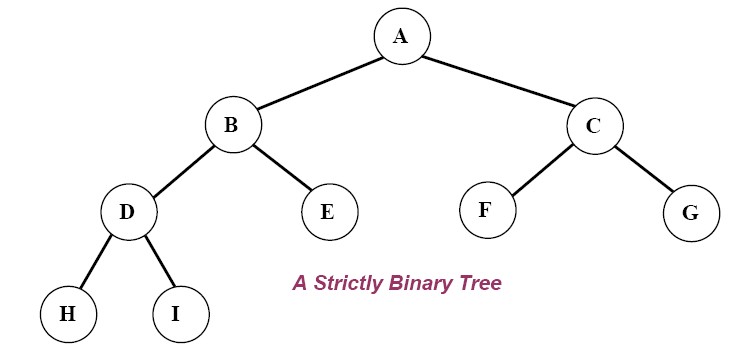
**BinaryNode \*left; // Left child**

**BinaryNode \*right; // Right child**

**};**

Strictly Binary Trees

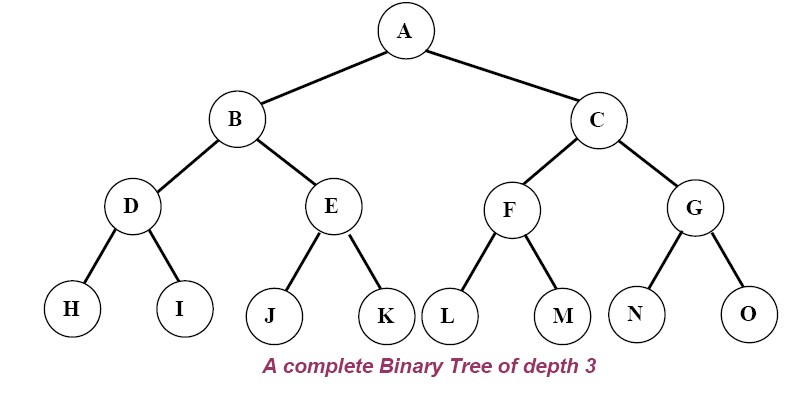
• If every *nonleaf* node (internal node) in a binary tree has *nonempty*



left and right subtrees, the tree is called a ***strictly binary tree***.

Complete Binary Trees

• A ***complete binary tree*** of depth ***d*** is a strictly binary tree in which all its leaves are at level ***d***.



Traversing Binary Trees

• One of the common operations of a binary tree is to ***traverse*** the tree.

• **Traversing** a tree is to pass through all of its nodes once.

 You may want to print the contents of each node or to process

the contents of the nodes.

 In either case each node of the tree is visited.

• Commonly, **tree traversal** is used to print out the data in a tree in a

certain order.

Traversing Binary Trees (cont…)

• There are three main traversal methods where traversing a binary tree involves visiting the root and traversing its left and right subtrees.

 The only difference among these three methods is the order in which these three operations are performed.

Binary Tree Traversal Methods

• The three main traversal methods are:

**(1) Inorder traversal.**

**(2) Postorder traversal. (3) Preorder traversal.**

(1) Inorder Traversal:

• Traversing a binary tree in ***inorder*** (or symmetric order):

**( left - root - right )**

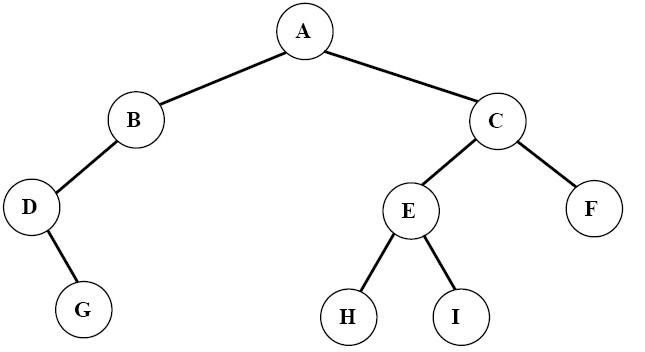
1. Traverse the ***left subtree*** in inorder.

2. Visit the ***root***.

3. Traverse the ***right subtree*** in inorder.

Traversing a binary tree in *Inorder*

**Inorder*: left-root-right***



**Inorder: DGBAHEICF**



Traversing a binary tree in *Inorder*

**void BinarySearchTree::inorder(BinaryNode \*t)**

**{**

**if(t != NULL)**

**{**

**inorder(t->left); cout << t->element; inorder(t->right);**

**}**

**}**

(2) Postorder Traversal:

• Traversing a binary tree in ***postorder***:

**( left - right - root )**

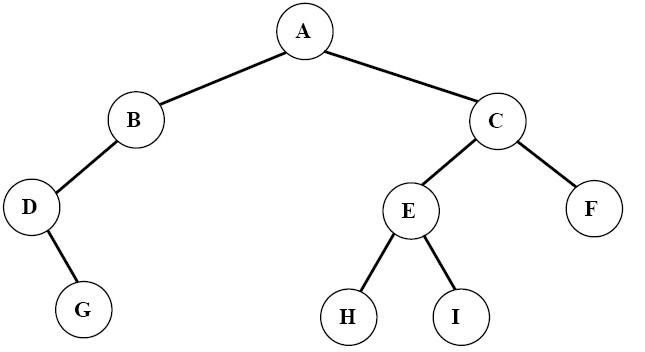
1. Traverse the ***left subtree*** in postorder.

2. Traverse the ***right subtree*** in postorder.

3. Visit the ***root***.

Traversing a binary tree in *Postorder*

**Postorder*: left-right-root***



**Postorder: GDBHIEFCA**



Traversing a binary tree in *Postorder*

**void BinarySearchTree::postorder(BinaryNode \*t)**

**{**

**if(t != NULL)**

**{**

**postorder(t->left); postorder(t->right); cout << t->element;**

**}**

**}**

(3) Preorder Traversal:

• Traversing a binary tree in ***preorder*** (or depth-first order):

**( root - left - right )**

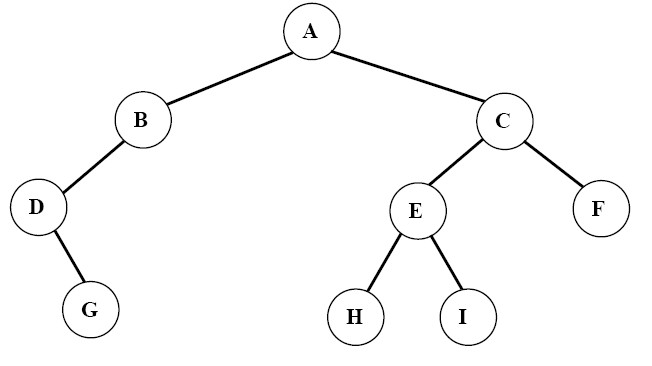
1. Visit the ***root***.

2. Traverse the ***left subtree*** in preorder.

3. Traverse the ***right subtree*** in preorder.

Traversing a binary tree in *Preorder*

**Preorder*: root-left-right***



**Preorder: ABDGCEHIF**



Traversing a binary tree in *Preorder*

**void BinarySearchTree::preorder(BinaryNode \*t)**

**{**

**if(t != NULL)**

**{**

**cout << t->element; preorder(t->left); preorder(t->right);**

**}**

**}**

**Traversing Binary Trees**

185

+

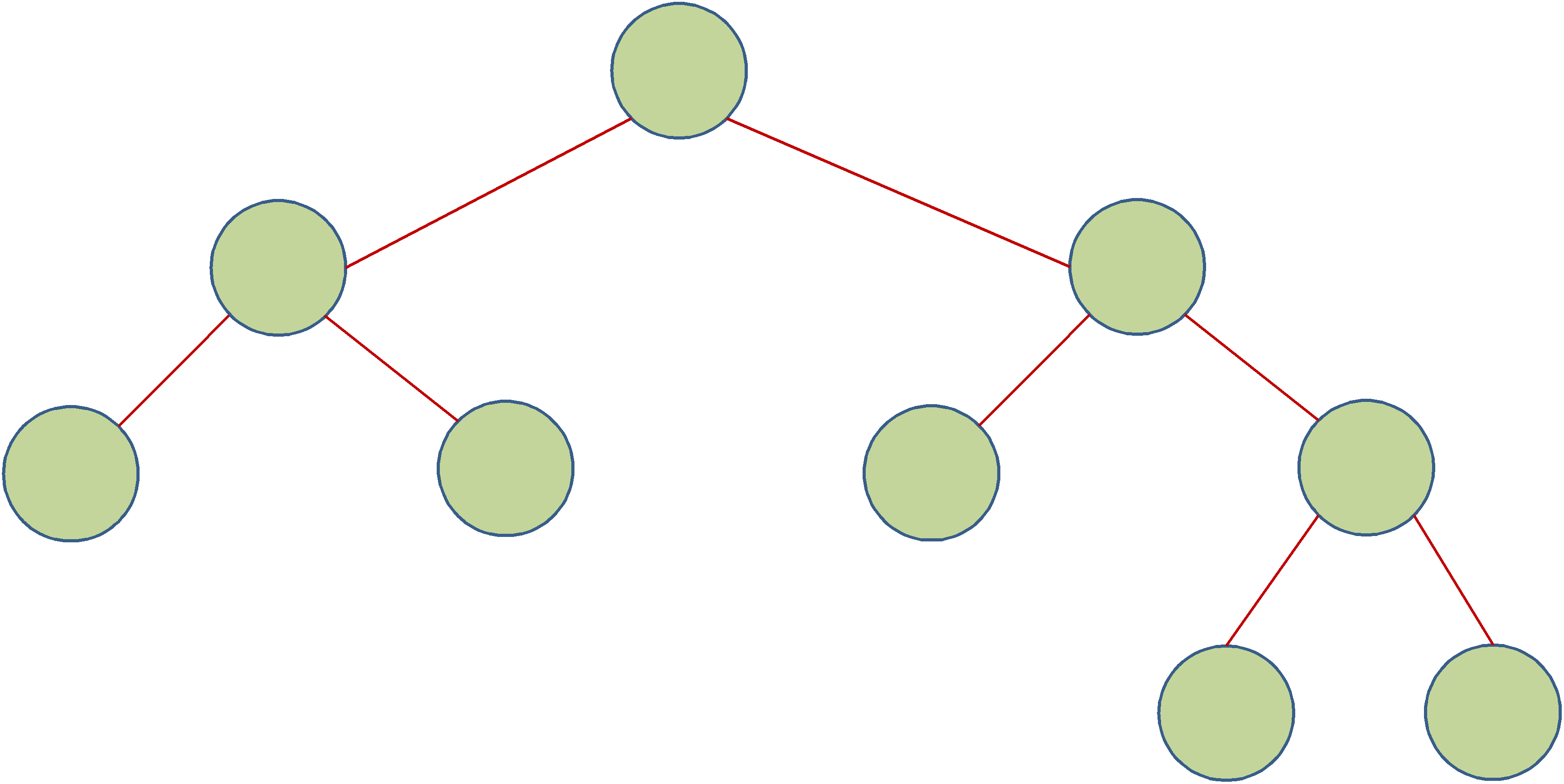
- \*

A B C /



E F

**Inorder: Postorder: Preorder:**



**A-B+C\*E/F**

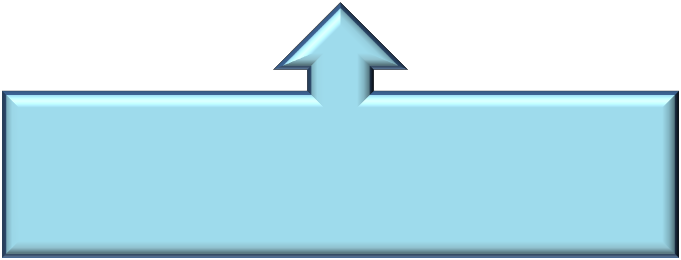
**AB-CEF/\*+**



**+-AB\*C/EF**



**Another Example**



**(Expression Tree)**