

Lecture - 10
on
Data Structures

TREES

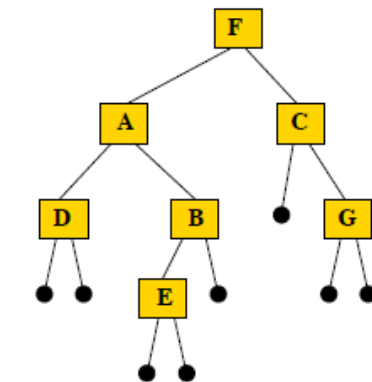
A **binary tree T** is defined as **a finite set of elements, called nodes**, such that:

- a) T is empty (called the null tree or empty tree), or
- b) T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T1 and T2.

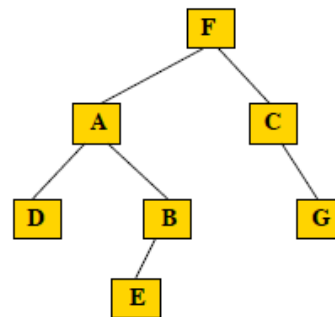
If T does contain a root R, then the two trees T1 and T2 are called, respectively, the left and right subtrees of R. If T1 is nonempty, then its root is called the left successor of R; similarly, if T2 is nonempty, then its root is called the right successor of R.



A Sample Binary Tree



Values are often shown in the nodes



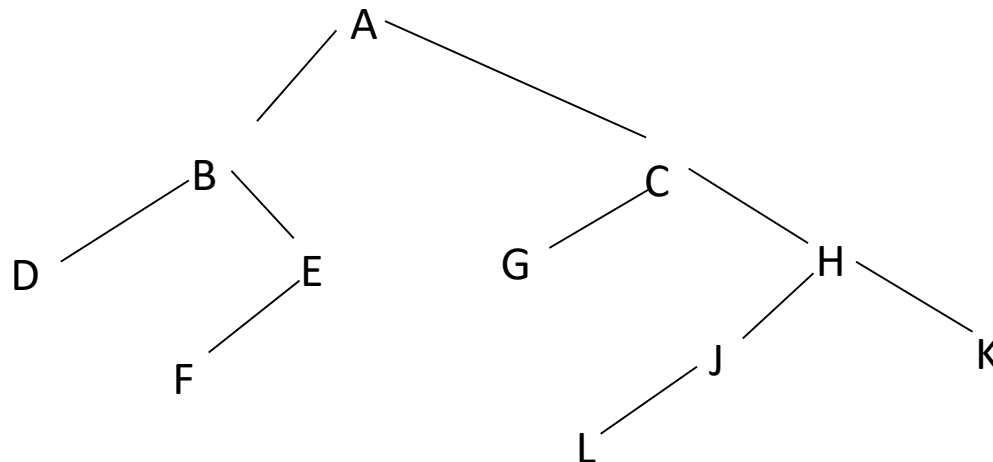
Often, the leaves are not shown,
but they're still there!

17-4

TREES

Any node N in a binary tree T has either 0, 1 or 2 successors. The nodes A , B , C and H have two successors, the nodes E and J have only one successor, and the nodes D , F , G , L and K have no successors. The nodes with no successors are called *terminal nodes*.

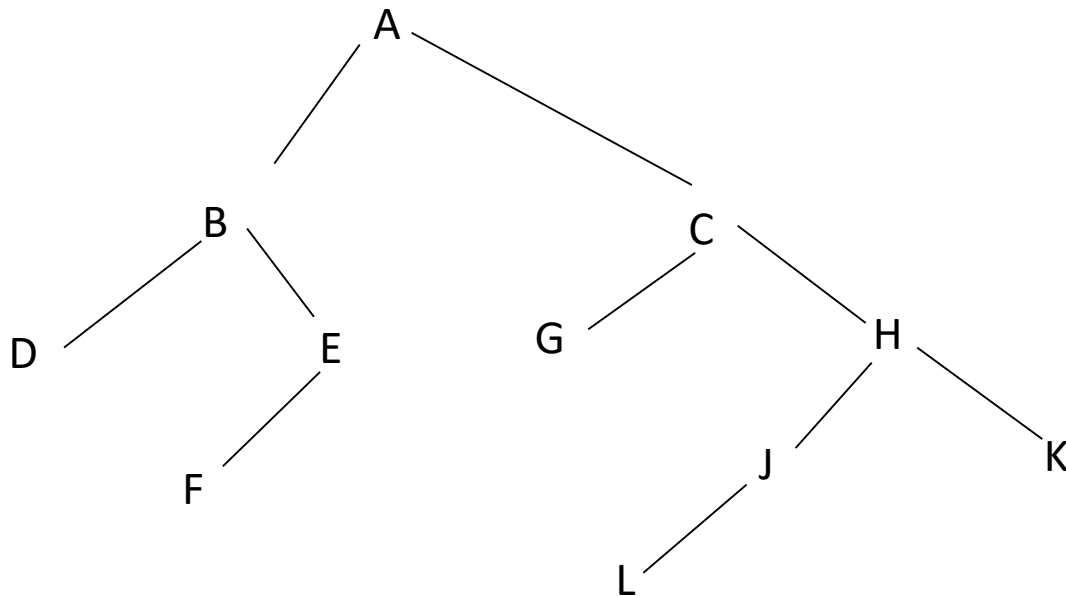
- The above definition of the binary tree T is recursive since T is defined in terms of the binary subtrees T_1 and T_2 . This means, in particular, that every node N of T contains a left and a right subtree. Moreover, if N is a terminal node, then both its left and right subtree are empty.
- Binary tree T and T' are said to be similar if they have the same structure or, in other words, if they have the same shape. The trees are said to be copies if they are similar and if they have the same contents at corresponding nodes.



TREES

Observe that;

- (a) B is a left successor and C is a right successor of the node A.
- (b) The left subtree of the root A consists of the nodes B, D, E and F, and the right subtree of A consists of the nodes C, G, H, J, K and L.



TREE

Example 7.1:

Consider the four binary trees in fig. 7.2. The three trees (a), (c) and (d) are similar. In particular, the trees (a) and (c) are copies since they also have the same data at corresponding nodes. The tree (b) is neither similar nor a copy of the tree.

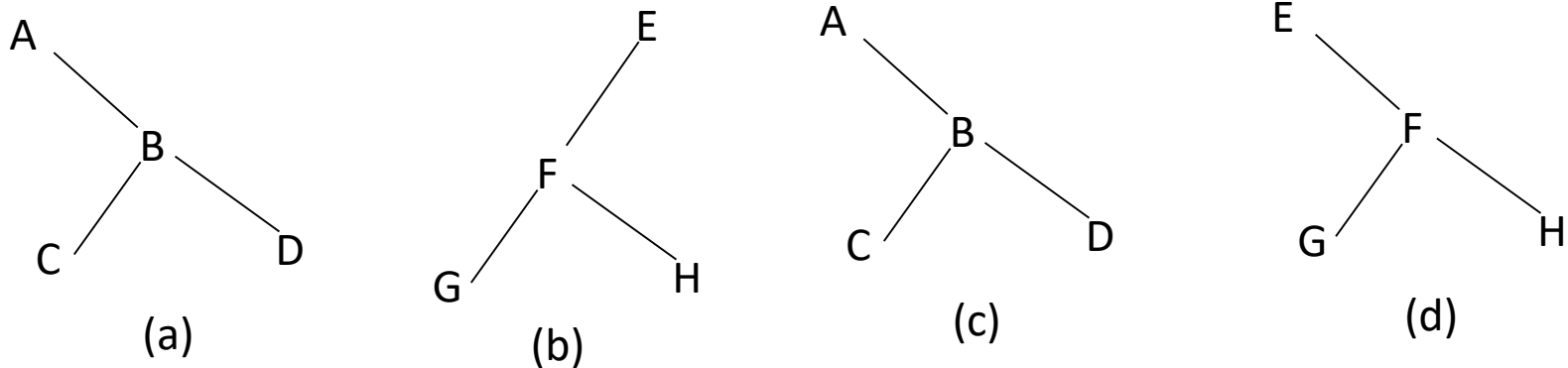
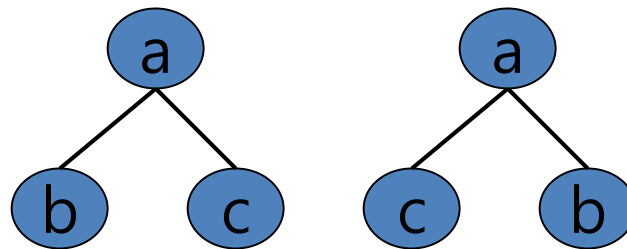


Fig. 7.2

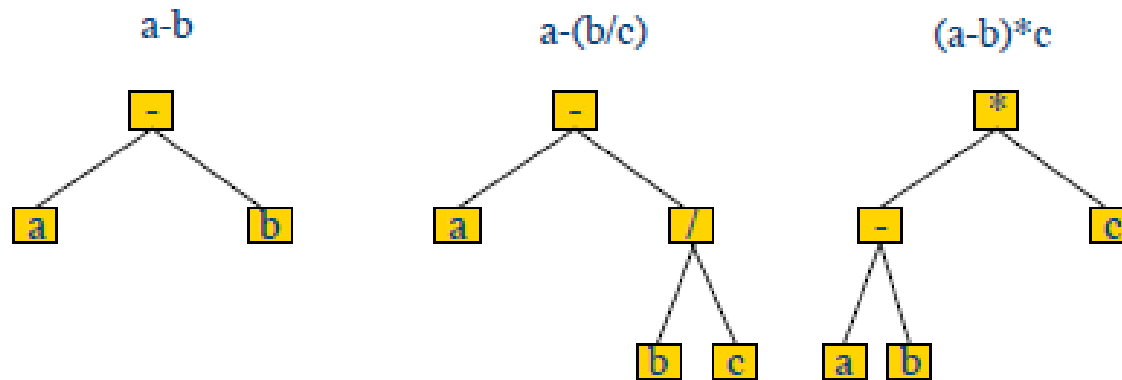
Difference between a tree & a binary tree

- A binary tree may be empty; a tree cannot be empty.
- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree.
- The sub trees of a binary search tree are ordered; those of a tree are not ordered.





Arithmetic Expressions as Trees

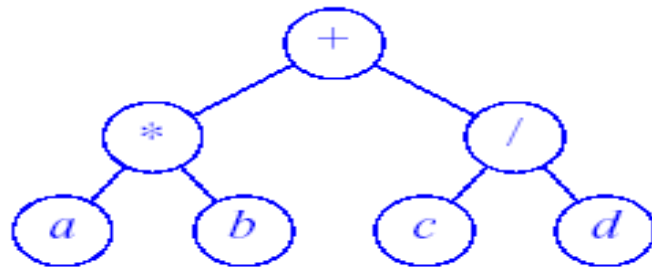


Arithmetic expressions are often represented as binary trees.

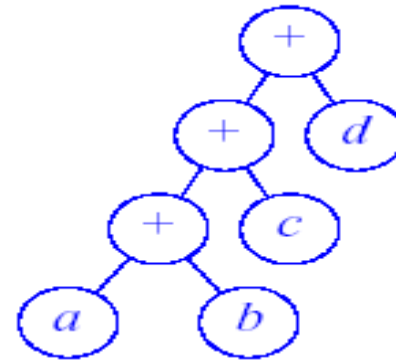
Internal nodes are **operations** - Leaves are **numbers/variables**.

Operator **precedence** is enforced by the tree shape.

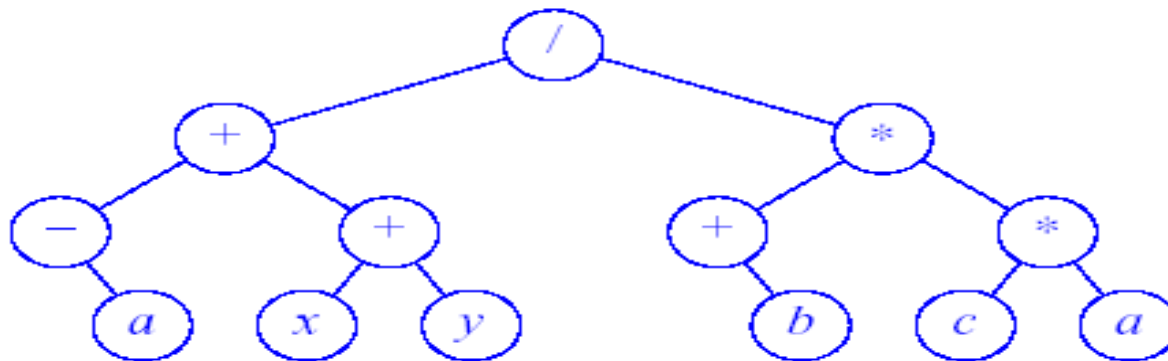
Arithmetic Expressions as trees(cont.)



(a) $(a * b) + (c / d)$



(b) $((a + b) + c) + d$



(c) $((-a) + (x + y)) / ((+b) * (c * a))$

Figure 11.5 Expression Trees

Arithmetic Expressions as trees(cont.)

Example 7.2: Algebraic Expressions

Consider any algebraic expression E involving only binary operations, such as

$$E = (a - b) / ((c * d) + e)$$

E can be represented by means of the binary tree T pictured in fig. 7.3.

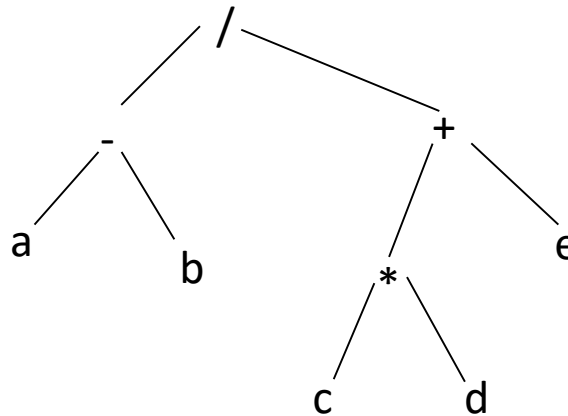


Fig. 7.3: $E = (a - b) / ((c * d) + e)$

TERMINOLOGY

Terminology describing family relationships is frequently used to describe relationships between the nodes of a tree T . Specifically, suppose N is a node in T with left successor S_1 and right successor S_2 . Then N is called the *parent* or *father* of S_1 and S_2 . Analogously, S_1 is called the *left child* or *son* of N , and S_2 is called the *right child* or *son* of N . Furthermore, S_1 and S_2 are said to be *siblings* or *brother*. Every node N in a binary tree, except the root, has a unique parent, called the *predecessor* of N .



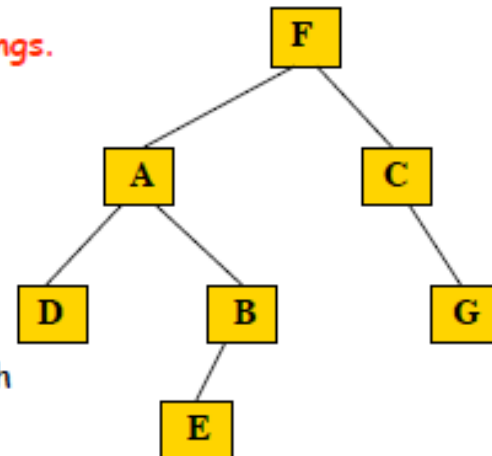
Tree Terminology

Nodes A and C are the **children** of node F ;
 F is the **parent** of A and C ; A and C are **siblings**.
Grandparents, grandchildren, etc. defined similarly.

A node without a parent is a **root**. E.g. node F is the root of the tree, and node A is the root of F 's left subtree (considered separately from the whole tree).

A **descendant** of a node n is a node on the path from n to a leaf. E.g. nodes D , B , and E are the descendants of A .

An **ancestor** of a node n is a node on the path from n to the root. E.g. Nodes B , A , and F are the ancestors of E .



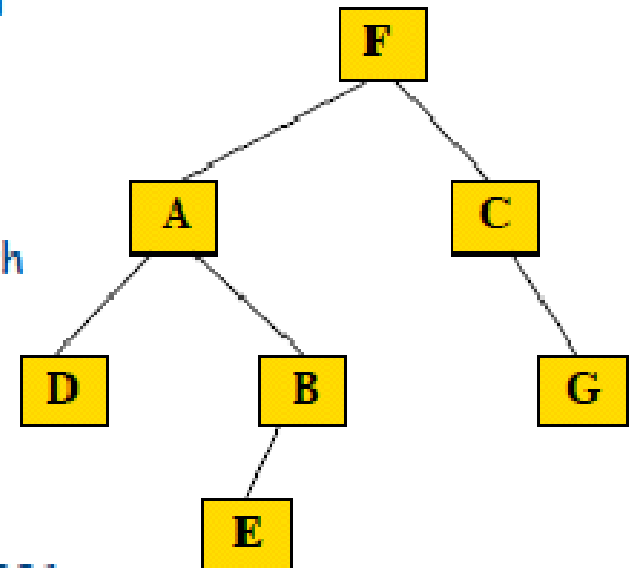


More Tree Terminology

The **height** of a node n is length of the longest path from n to a leaf below it. E.g., node G has height 1, node C has height 2, and node F has height 4.

The **depth** of a node n is the length of the path from n to the root. E.g., node F has depth 0, node C has depth 1, and node G has depth 2.

A binary tree is **height-balanced** iff at every node n , the heights of n 's left and right subtrees differ by no more than 1. The example tree is height-balanced, but would not be if G were removed.



Complete Binary Trees

The tree T is said to be complete if all its levels, except possibly the last, have the maximum number of possible nodes, and if all the nodes at the last level appear as far left as possible.

Complete Binary Trees

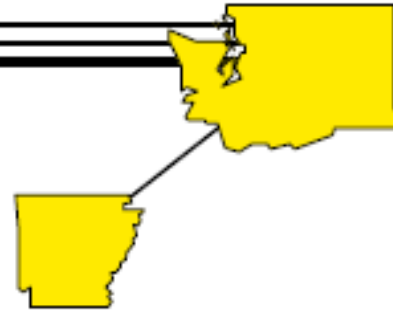
A complete binary tree is a special kind of binary tree which will be useful to us.

When a complete binary tree is built, its first node must be the root.

When a complete binary tree is built, its nodes are generally added one at a time. As with any tree, the first node must be the root.

Complete Binary Trees

The second node of a complete binary tree is always the left child of the root...

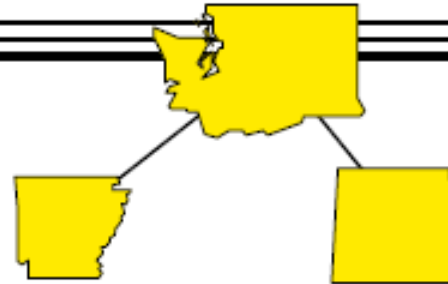


With a complete binary tree, the second node must be the left child of the root.

Complete Binary Trees

The second node of a complete binary tree is always the left child of the root...

... and the third node is always the right child of the root.

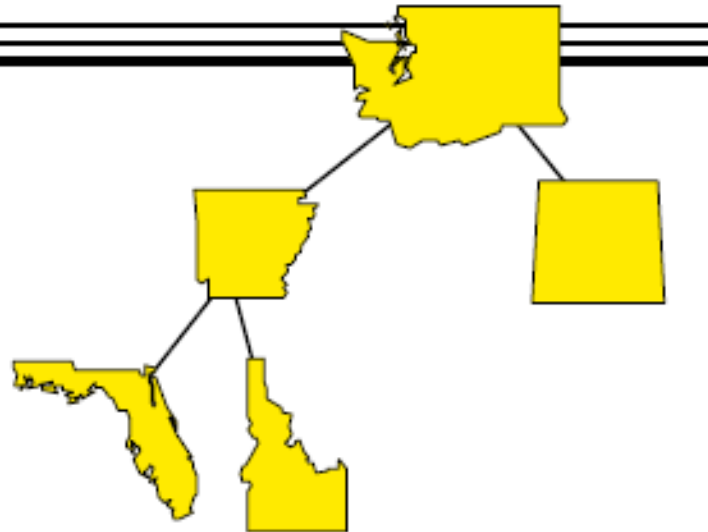


The next node must be the right child of the root.

Following this rule, where would the fifth node have to be placed?

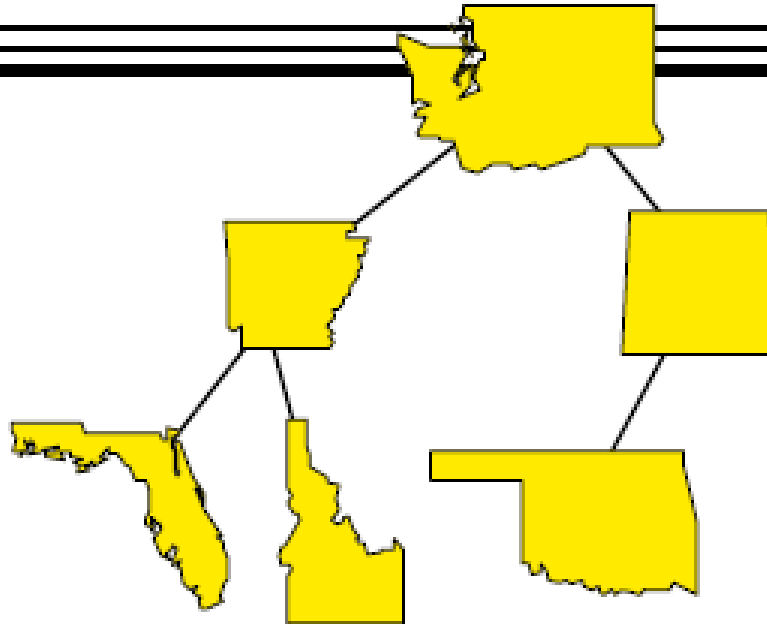
Complete Binary Trees

The next nodes must always fill the next level from **left to right**.



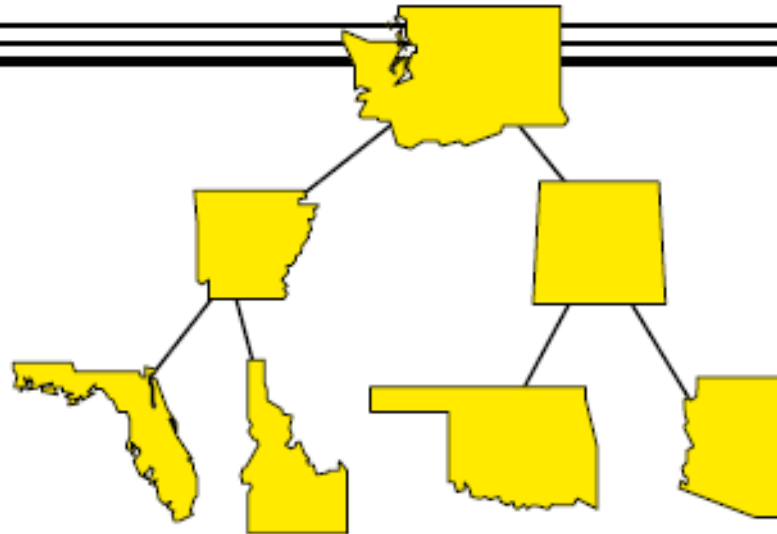
Complete Binary Trees

The next nodes must always fill the next level from left to right.



Complete Binary Trees

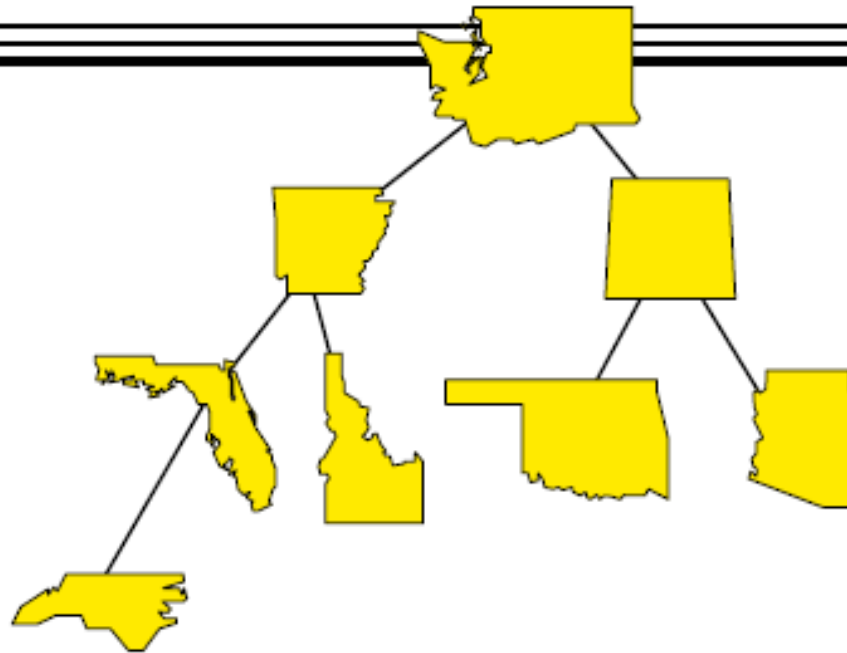
The next nodes must always fill the next level from left to right.



Where would the next node after this go?...

Complete Binary Trees

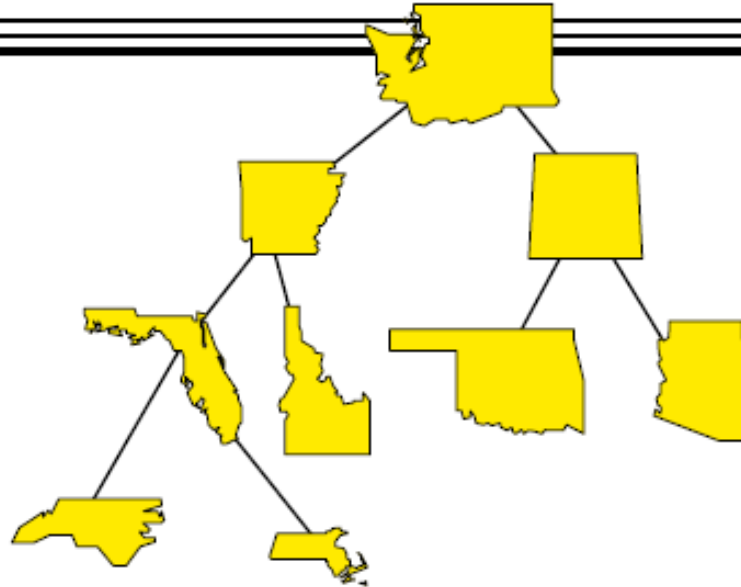
The next nodes must always fill the next level from **left to right**.



...now we can move to the next level, and add the first node

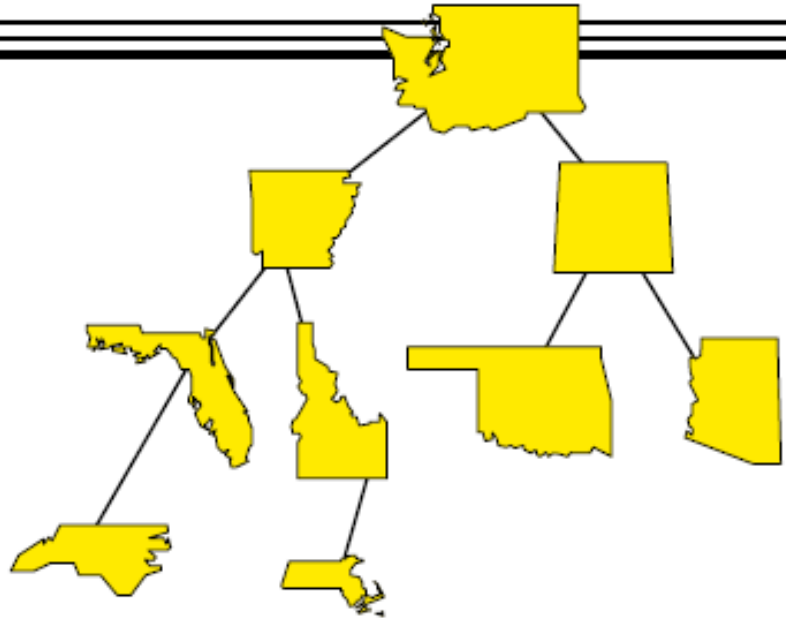
Complete Binary Trees

The next nodes must always fill the next level from **left to right**.



Then the right child

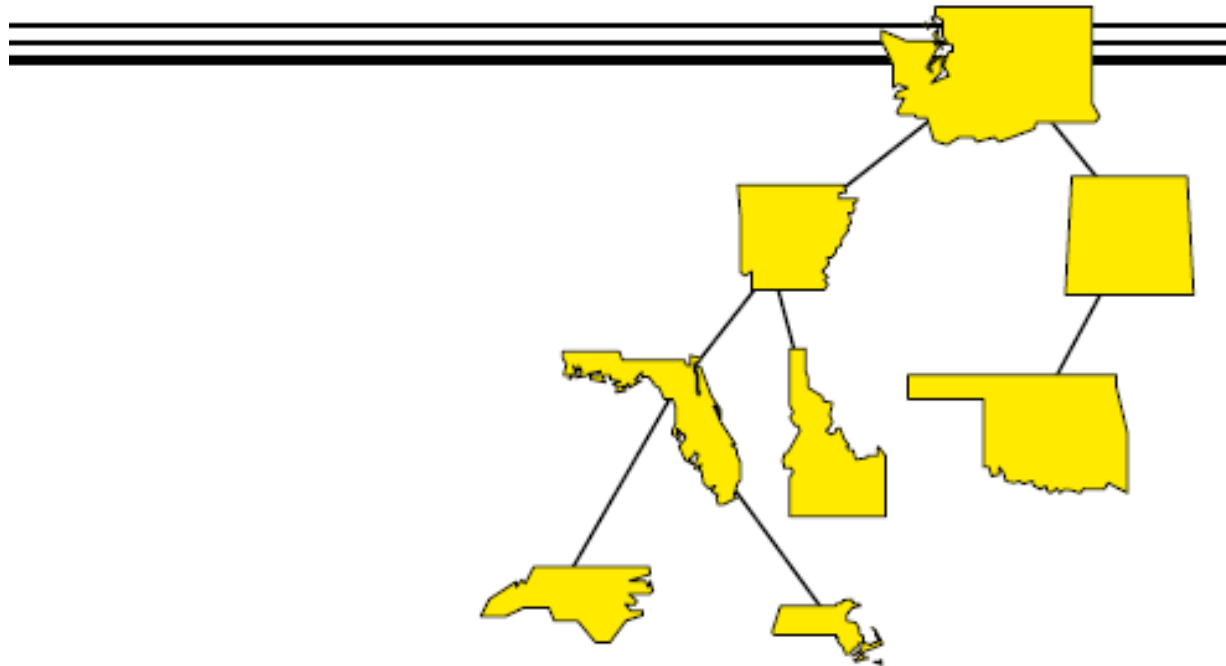
Is This Complete?



Just to check your understanding, is this binary tree a complete binary tree?

No, it isn't since the nodes of the bottom level have not been added from left to right.

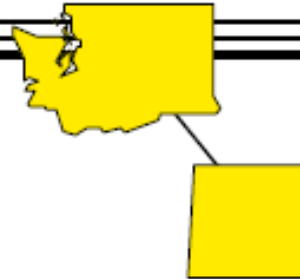
Is This Complete?



Is this complete?

No, since the third level was not completed before starting the fourth level.

Is This Complete?



This is also not complete since
has a right child but no left child.

Is This Complete?



But this binary tree is complete. It has only one node, the root

Is This Complete?

Yes!

- ✓ It is called the empty tree, and it has no nodes, not even a root.

This binary tree is also complete,
You see, this binary tree has no nodes at all. It is called the empty tree, and it is considered to be a complete binary tree.

Complete Binary tree

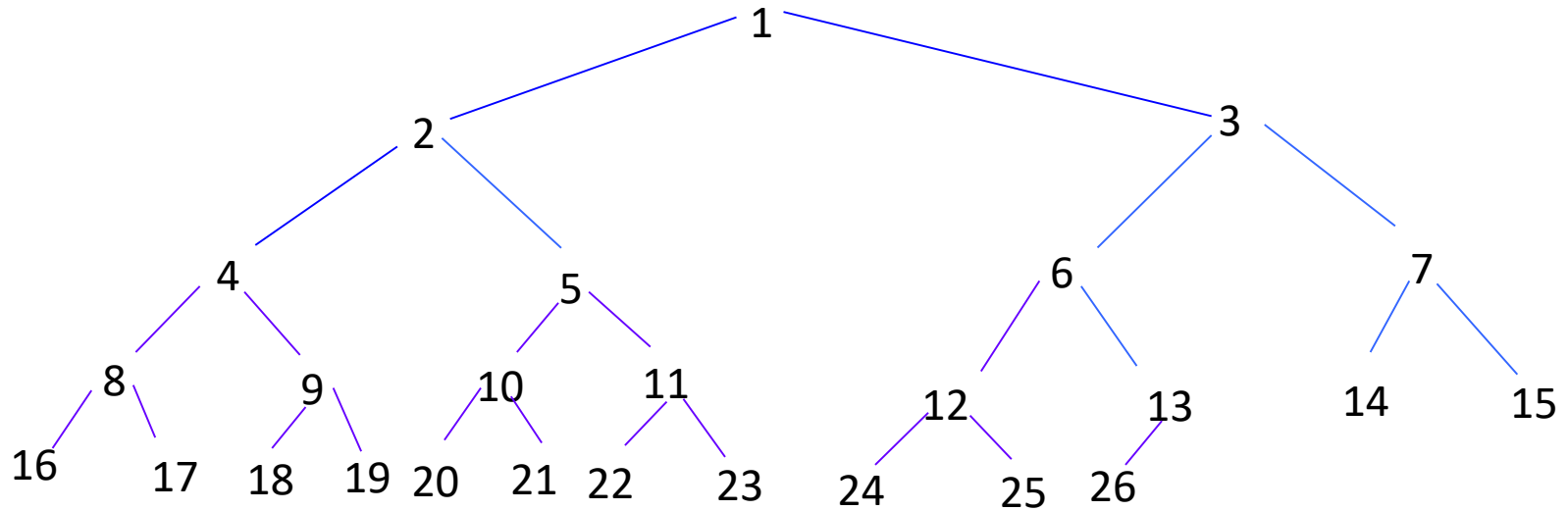


Fig. Complete tree T_{26}

The left and right children of the node K are , respectively, $2 * K$ and $2 * K + 1$, and the parent of K is the node $[K/2]$.

The children of node 9 are the nodes 18, 19, and its parent is the node $[9/2] = 4$.

The depth d_n of the complete tree T_n with n nodes is given by

$$D_n = [\log_2 n + 1]$$

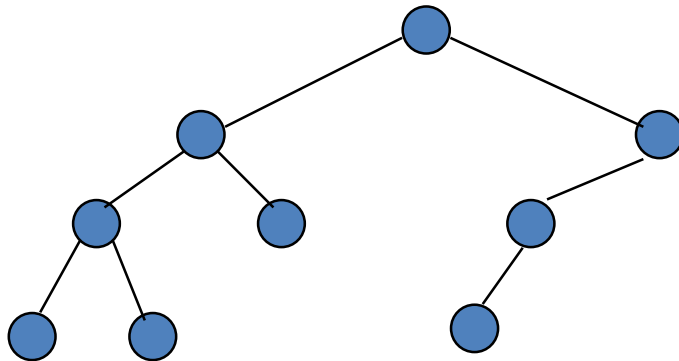
This is relatively small number. For example if the complete tree T_n has $n = 1000000$ nodes, then its depth $D_n = 21$.

Extended Binary trees: 2 trees

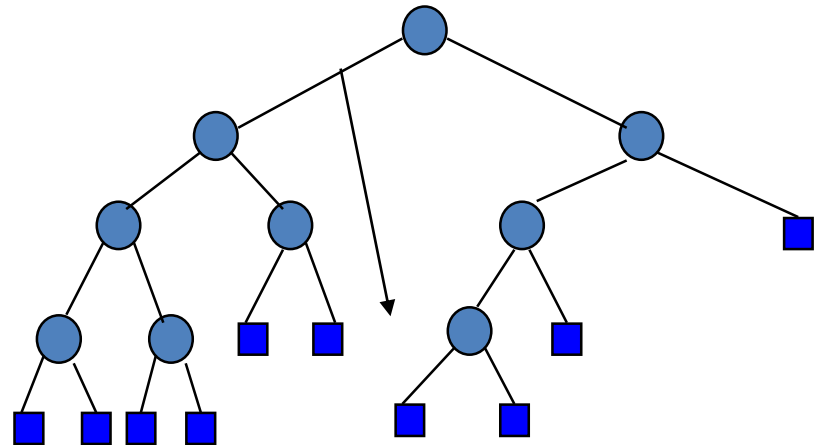
A binary tree T is said to be a 2 – tree or extended binary tree if each node N has either 0 or 2 children. The nodes with two children are called internal nodes, and the nodes with 0 children are called external nodes.

- Circles for internal nodes.
- Squares for external nodes.

Start with any binary tree and add an external node wherever there is an empty subtree.
Result is an extended binary tree.



A Binary Tree



An Extended Binary Tree

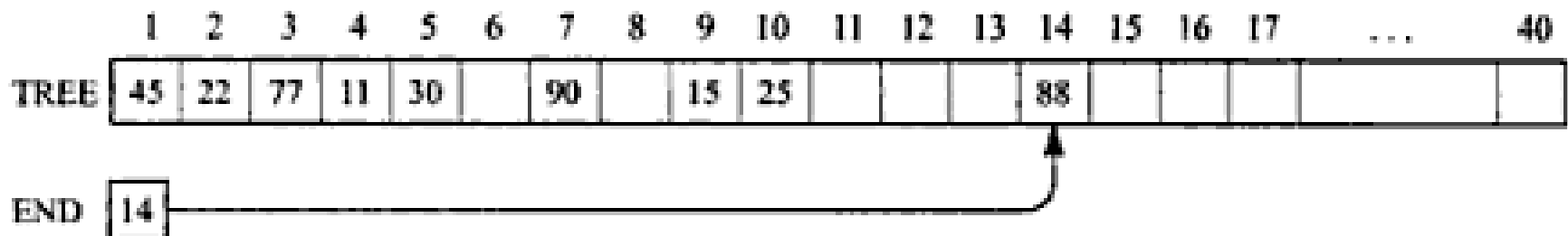
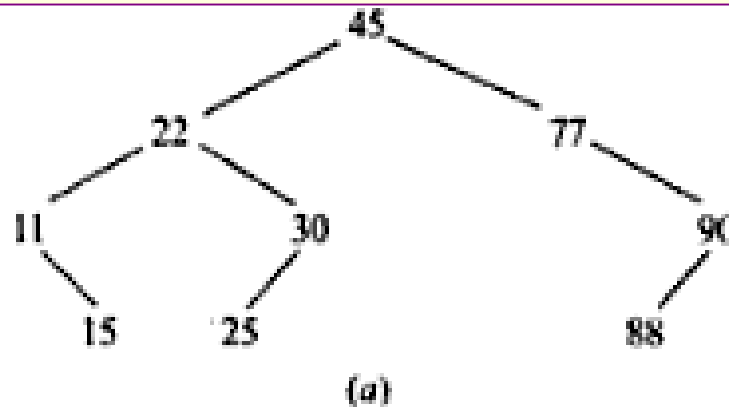
Binary Trees: Array representation

Sequential Representation of binary Trees uses only a single linear array TREE together with a pointer variable END as follows:

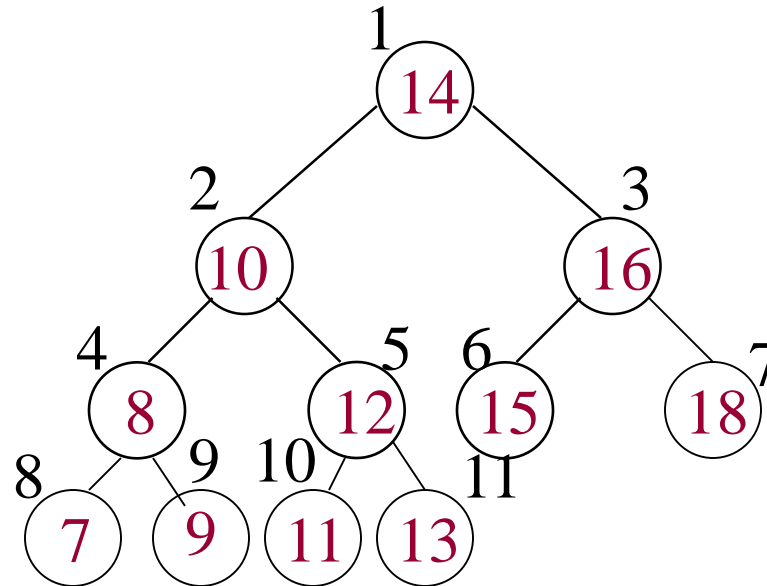
(a) The root R of T is stored in TREE[1].

(b) If a node occupies TREE[k], then its left child is stored in TREE[2 * K] and its right child is stored in TREE[2*k+1]

(c) END contains the location of the last node of T.



Binary Trees: Array Representation

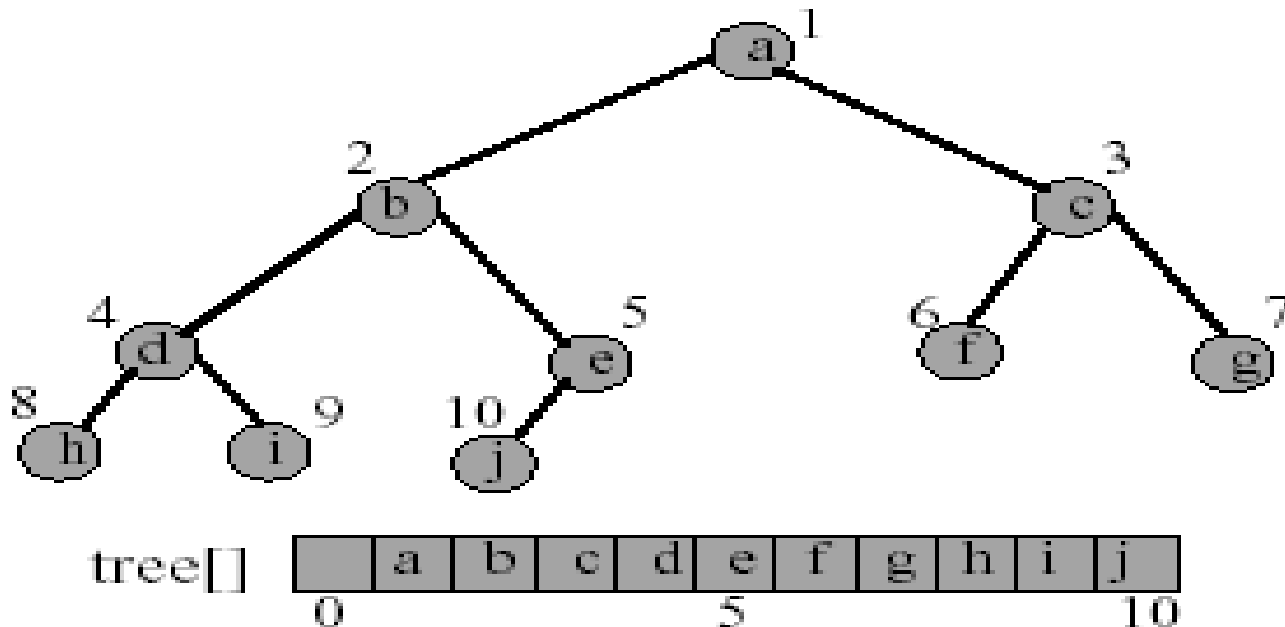


Array A:

1	2	3	4	5	6	7	8	9	10	11
14	10	16	8	12	15	18	7	9	11	13

Binary Trees: Array representation

- The binary tree is represented in an array by storing each element at the array position corresponding to the number assigned to it.



Linked Representation of Binary Tree

- The most popular way to present a binary tree
- Each element is represented by a node that has two link fields (`leftChild` and `rightChild`) plus an `Info` field
- The space required by an n node binary tree is $n * \text{sizeof}(\text{binaryTreeNode})$

Linked Representation of Binary trees

1. INFO[K] contains the data at the node N
2. LEFT[K] contains the location of the left child of node N
3. RIGHT[K] contains the location of the right child of node N.

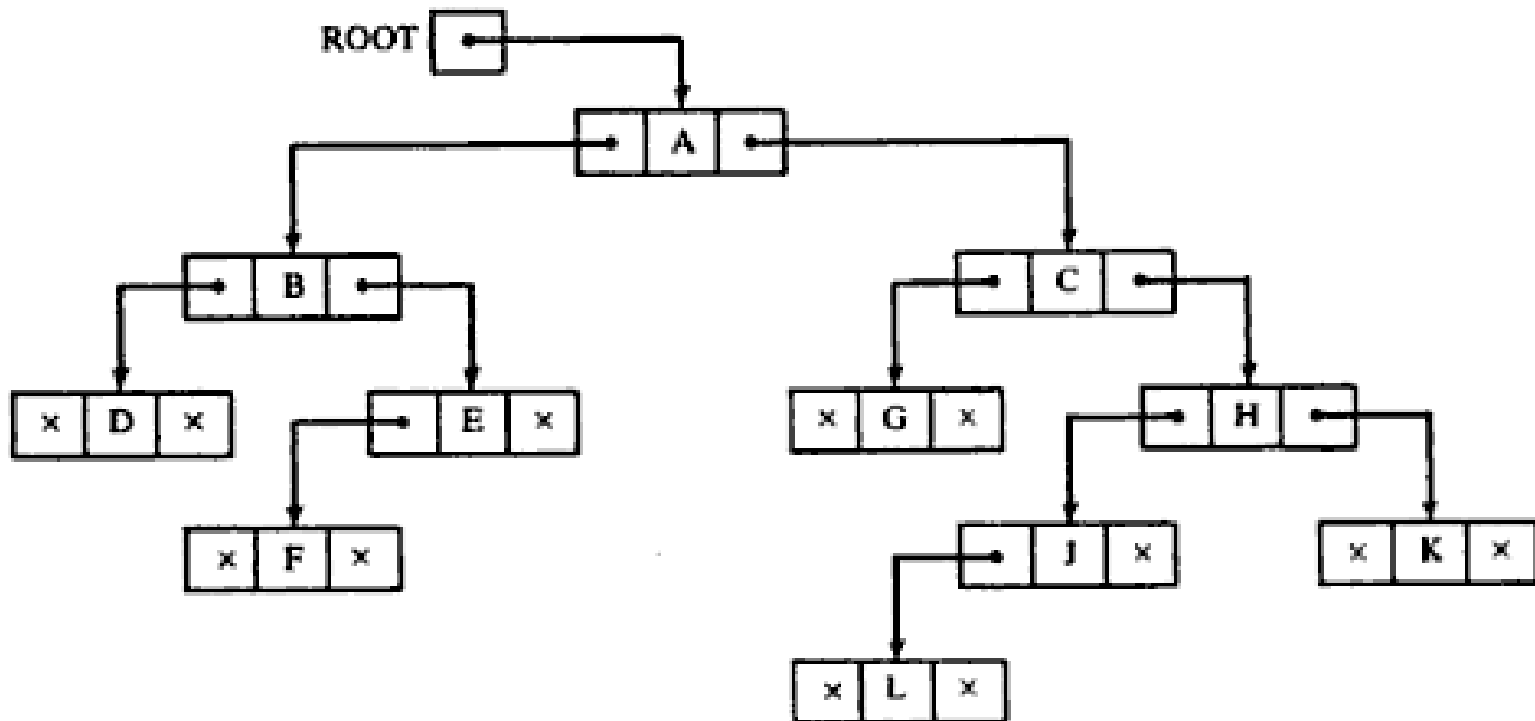


Fig : 7-6

Linked Representation of Binary trees

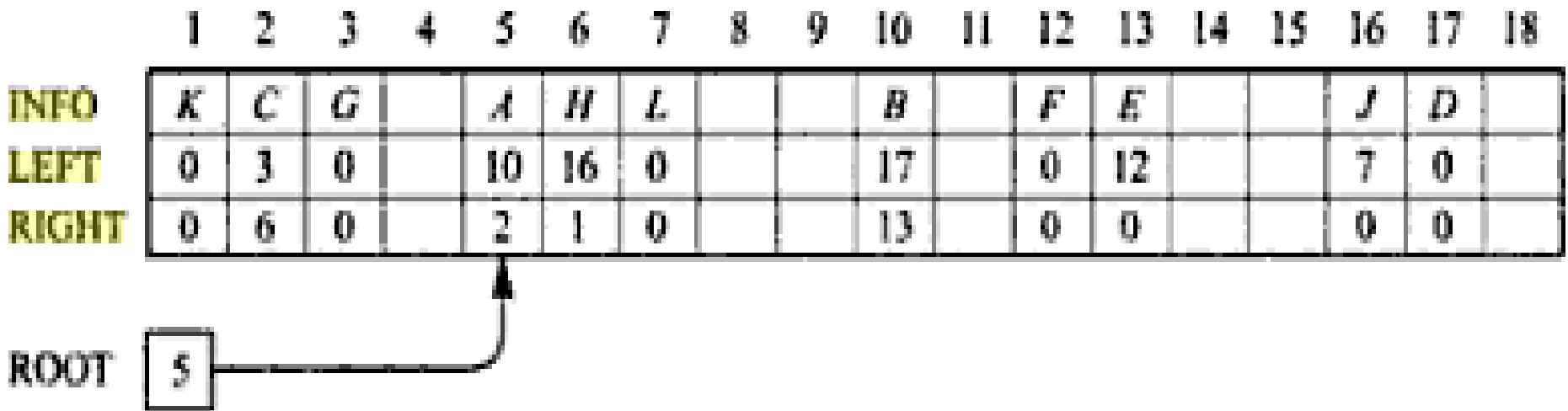


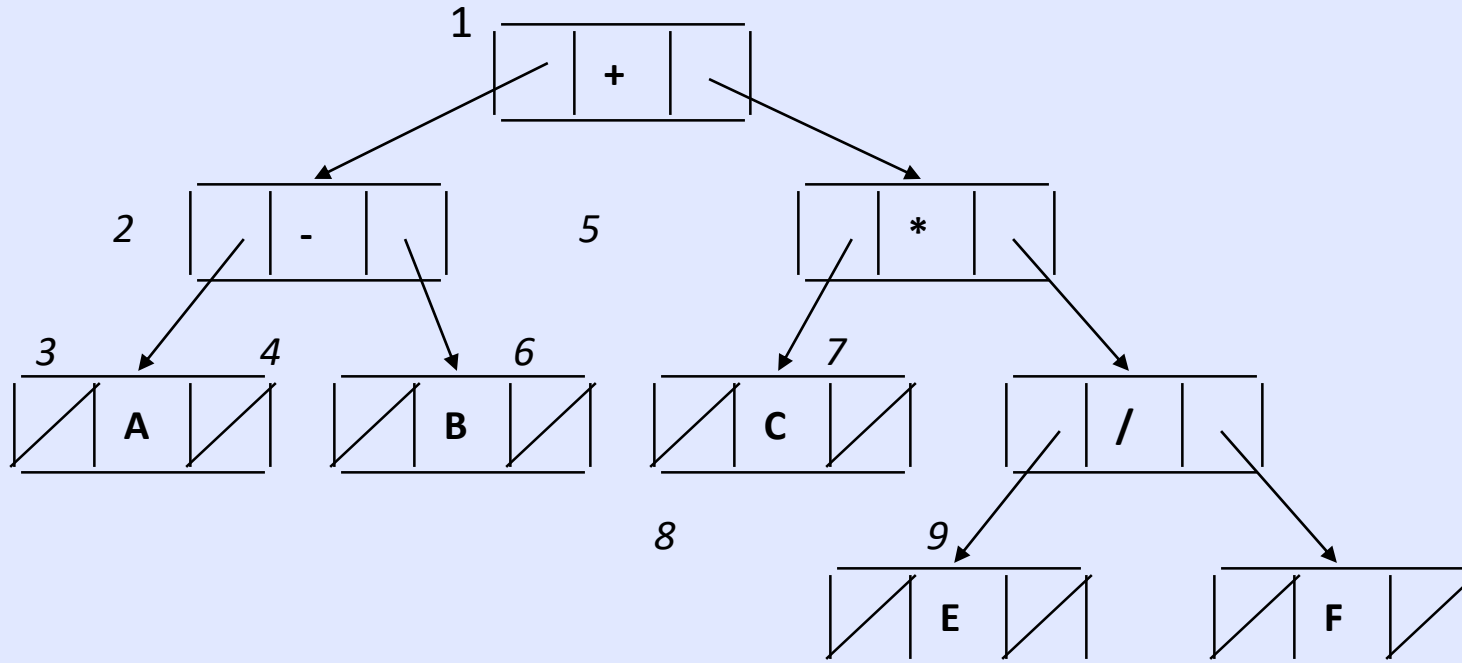
Fig:7-7

ROOT = 5 Points to **INFO**[5] :

INFO[5] = A since A is the root of T. Also, note that **LEFT**[5] = 10 points to **INFO**[10] = B since B is the **left** child of A, and **RIGHT**[5] = 2 points to **INFO**[2] = C since C is the **right** child of A. The choice of 18 elements for the arrays is arbitrary.

Linked representation of a binary tree

Linked representation uses explicit links to connect the nodes. Example:



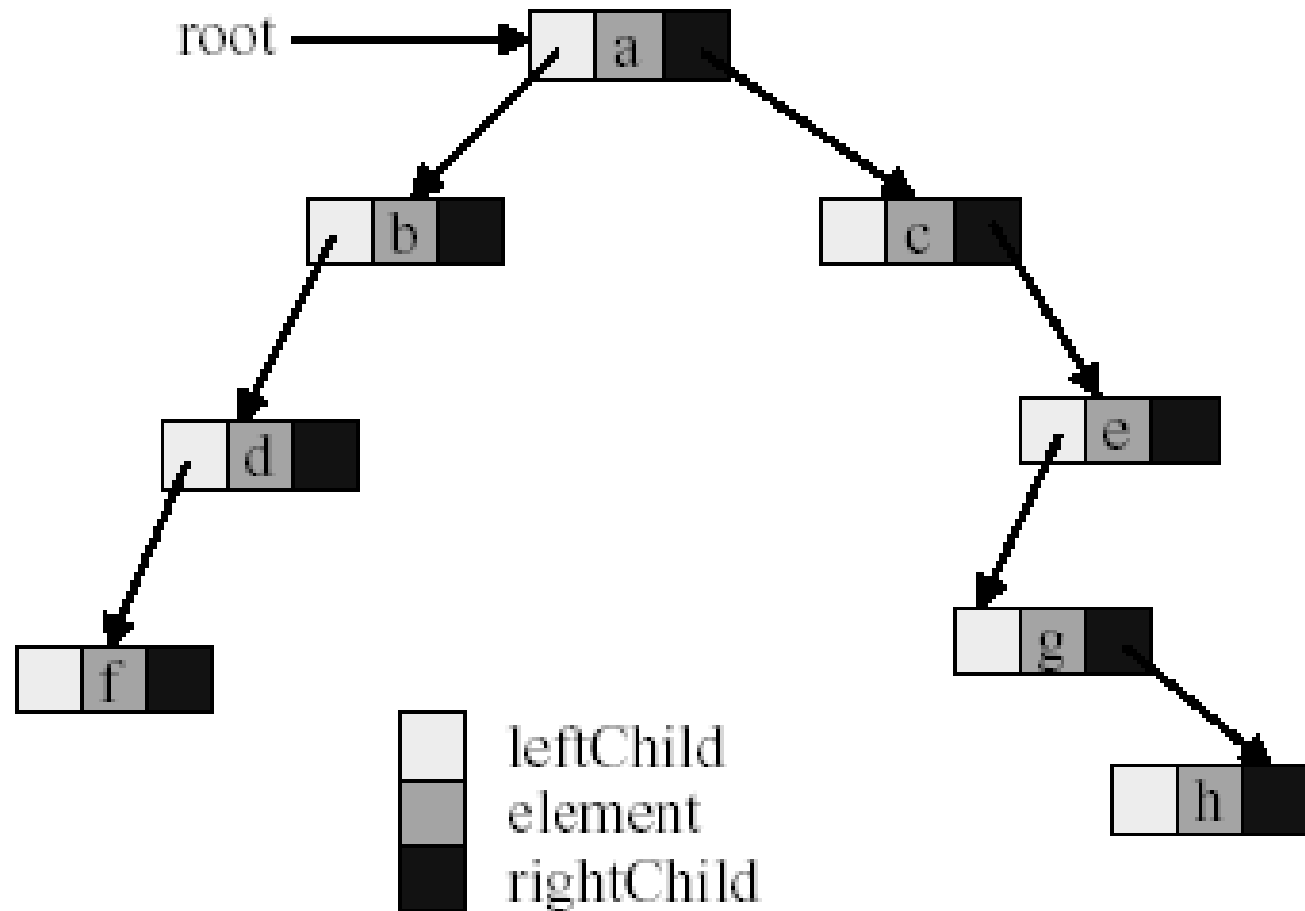
Nodes in this tree can be viewed as positions in a sequence (numbered 1 through 9).

Binary tree (linked representation)

We can use a positional sequence ADT to implement a binary tree. Our example tree, in this case, we be represented as follows:

position	1	2	3	4	5	6	7	8	9
data	+	-	A	B	*	C	/	E	F
leftChild	2	3	null	null	6	null	8	null	null
rightChild	5	4	null	null	7	null	9	null	null
parent	null	1	2	2	1	5	5	7	7

Linked Representation of Binary Tree



Linear representation of a binary tree

Advantages of linear representation:

1. *Simplicity.*
2. *Given the location of the child (say, k), the location of the parent is easy to determine ($k / 2$).*

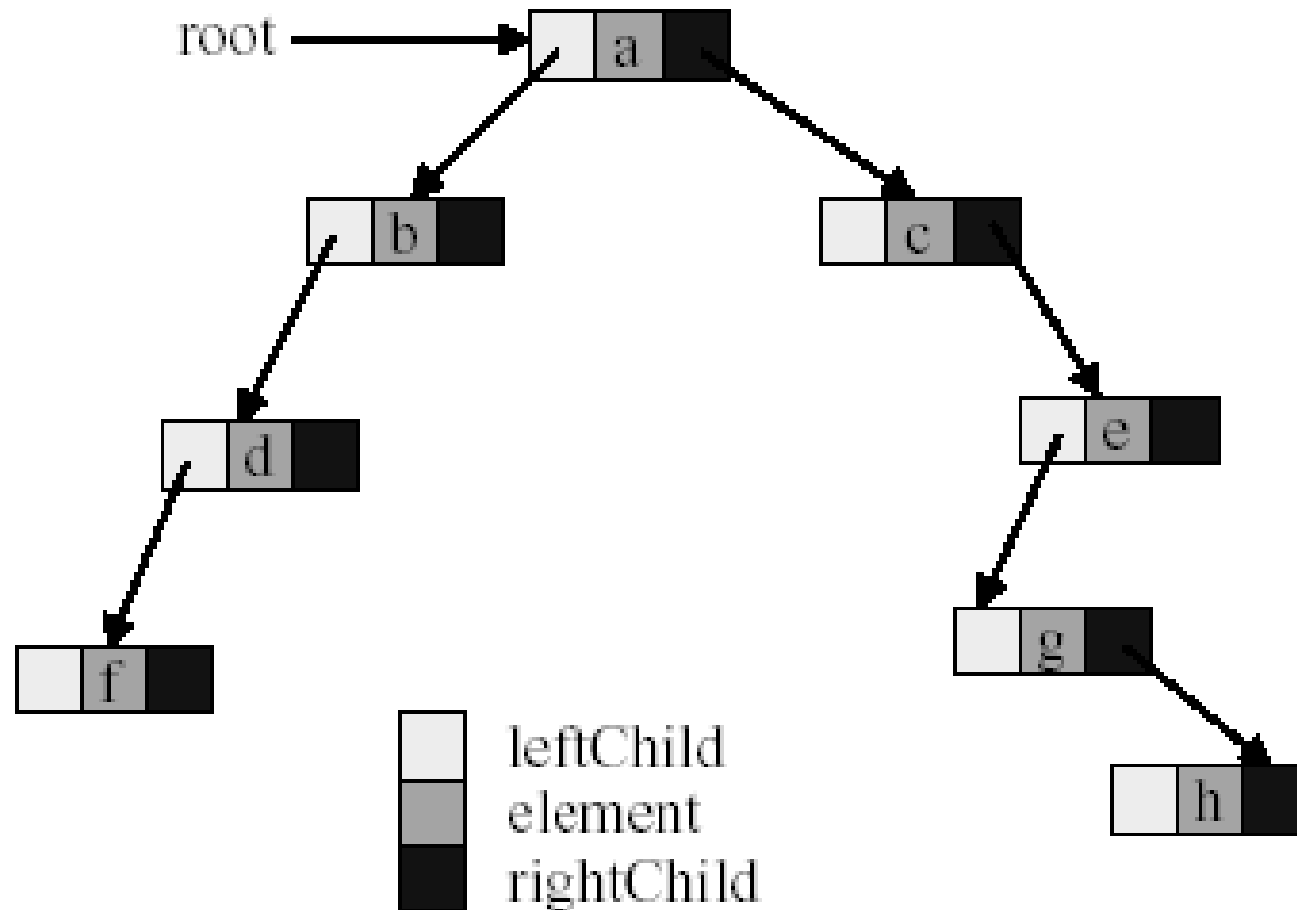
Disadvantages of linear representation:

1. *Additions and deletions of nodes are inefficient, because of the data movements in the array.*
2. *Space is wasted if the binary tree is not complete. That is, the linear representation is useful if the number of missing nodes is small.*

Linear representation of a binary tree can be implemented by means of a linked list instead of an array

This way the above mentioned disadvantages of the linear representation is resolved.

Linked Representation of Binary Tree



Binary Tree Traversal

- Many binary tree operations are done by performing a **traversal** of the binary tree
- In a traversal, each element of the binary tree is **visited** exactly once

Traversal of Binary Trees

There are three ways to traverse a tree, and these are:

- In-Order (Left-Root-Right)
- Pre-Order (Root-Left-Right)
- Post-Order (Left-Right-Root)

These traversal methods can be implemented in a recursive manner as you will notice:

Traversals

To Traverse a non-empty binary tree in In-Order, we need

to perform the following operations:

1. Visit the Left Subtree in In-Order
2. Visit the Root
3. Visit the Right Subtree in In-Order

To Traverse a non-empty binary tree in Pre-Order, we need to perform the following operations:

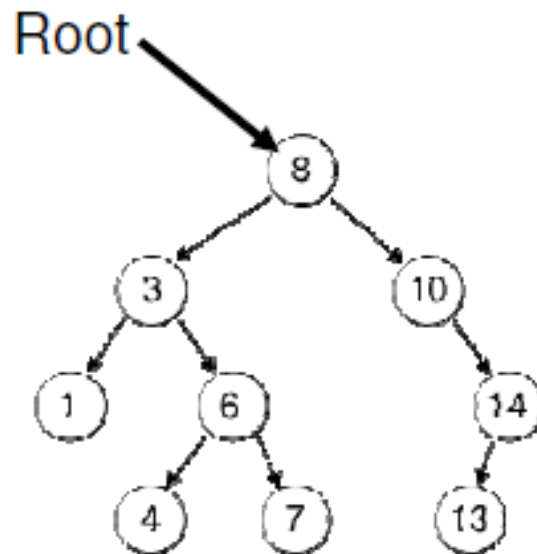
1. Visit the Root
2. Visit the Left Subtree in Pre-Order
3. Visit the Right Subtree in Pre-Order

Traversals

To Traverse a non-empty binary tree in Post-Order, we need to perform the following operations:

1. Visit the Left Subtree in Post-Order
2. Visit the Right Subtree in Post-Order
3. Visit the Root

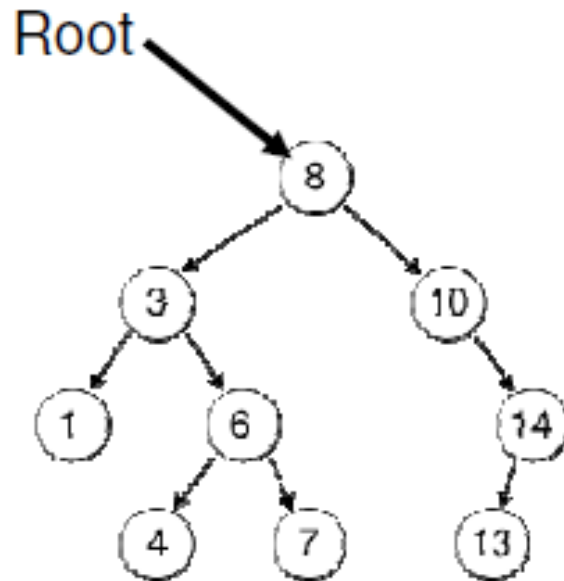
Question



Print the value in each node after an:

- In-Order Traversal
- Pre-Order Traversal
- Post-Order Traversal

Answer



Print the value in each node after an:

- In-Order Traversal

{1,3,4,6,7,8,10,13,14}

- Pre-Order Traversal

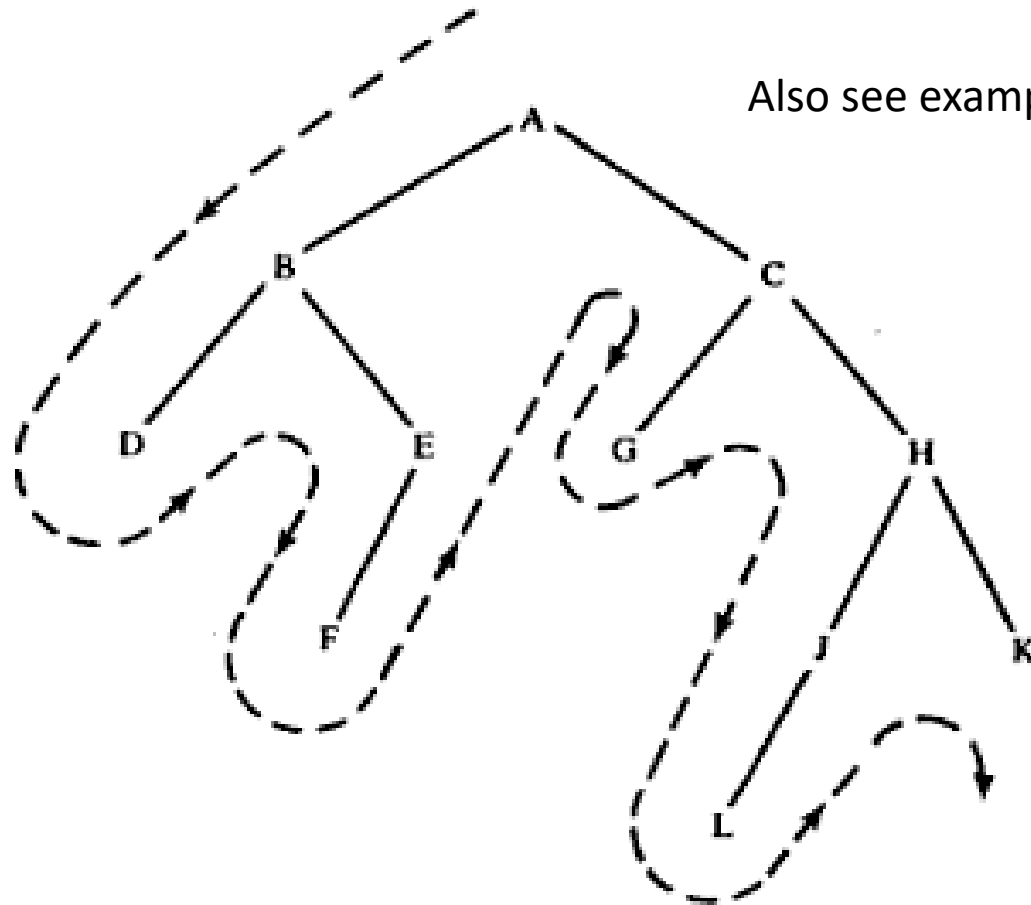
{8,3,1,6,4,7,10,14,13}

- Post-Order Traversal

{1,4,7,6,3,13,14,10,8}

19

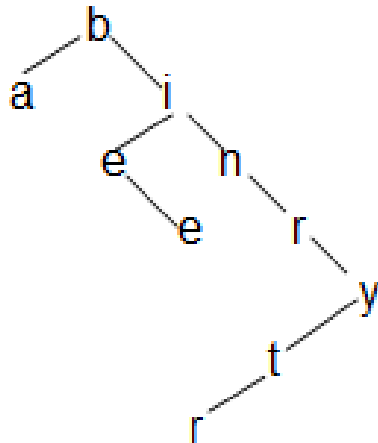
(Inorder)	D	B	F	E	A	G	C	L	J	H	K
(Postorder)	D	F	E	B	G	L	J	K	H	C	A
(Preorder)	A	B	D	E	F	C	G	H	J	L	K



Also see example 7.7, 7.9, 7.10

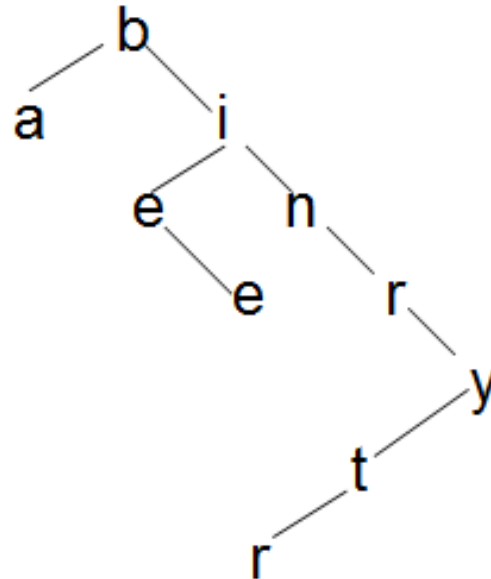
Binary Tree Traversal Methods

Example Consider a tree with an ordering property, where nodes are inserted in the following order b i n a r y t r e e, i.e.



The preorder traversal is: b a i e e n r y t r

Binary Tree Traversal Methods



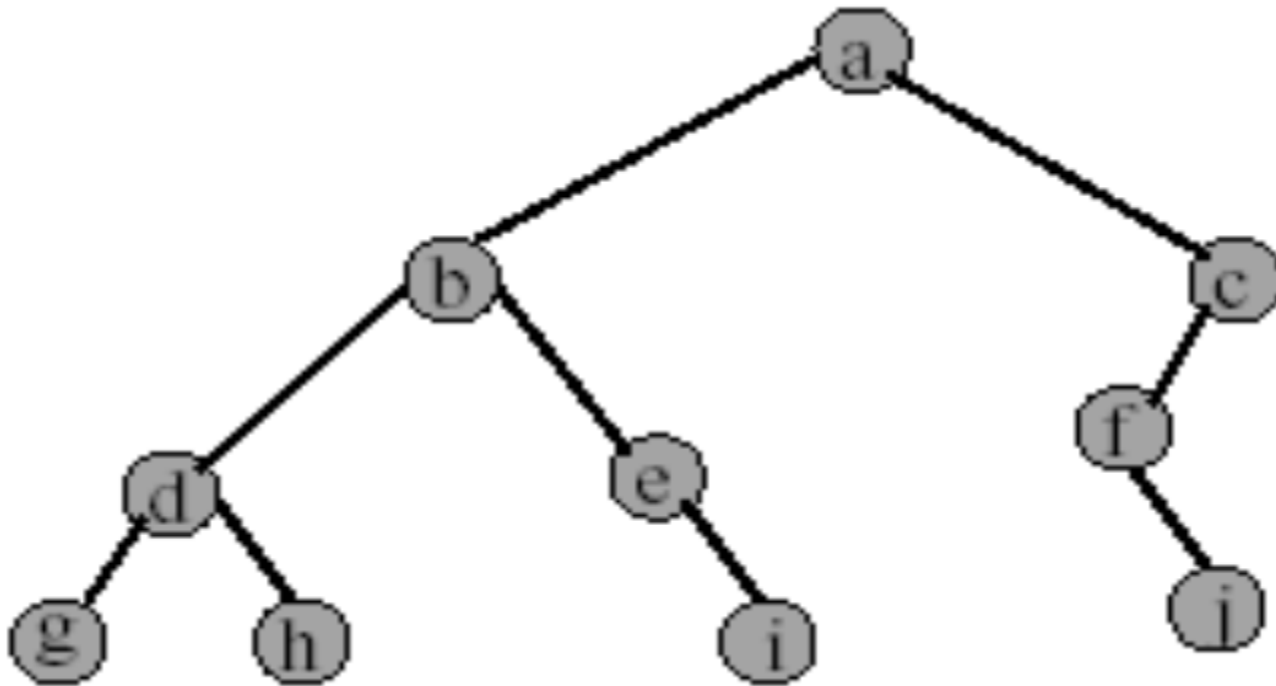
The nodes in the example tree are traversed in post-order as follows:

a e e r t y r n i b

The nodes in the example tree are traversed in in-order as follows:

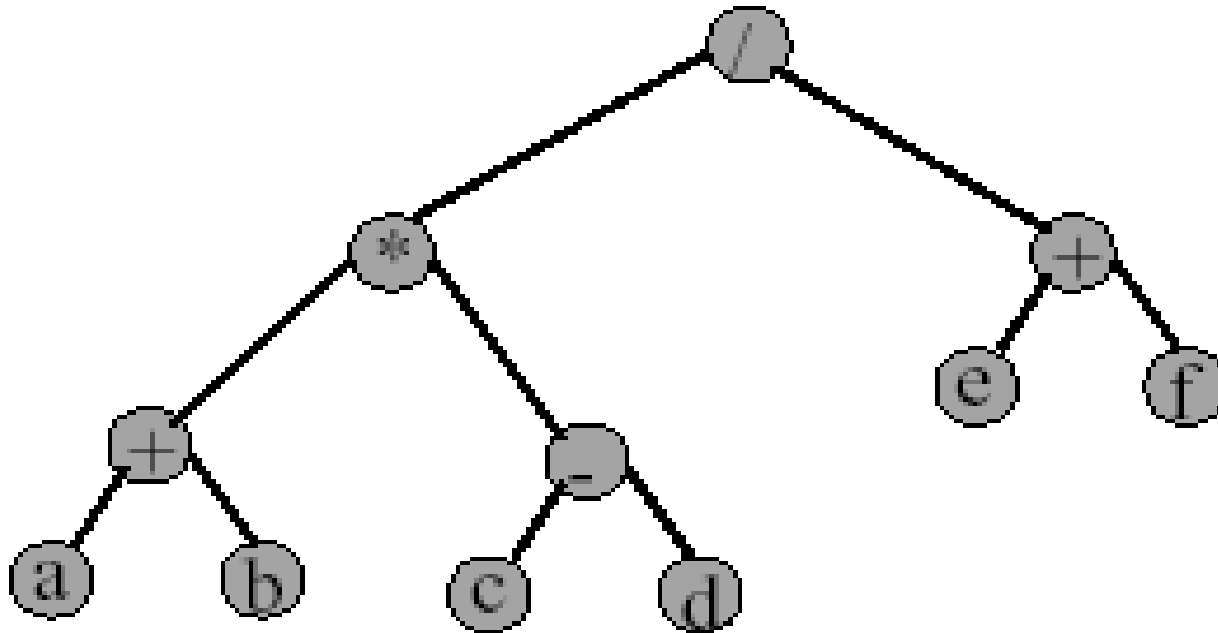
a b e e i n r r t y

Preorder Example (visit = print)



a b d g h e i c f j

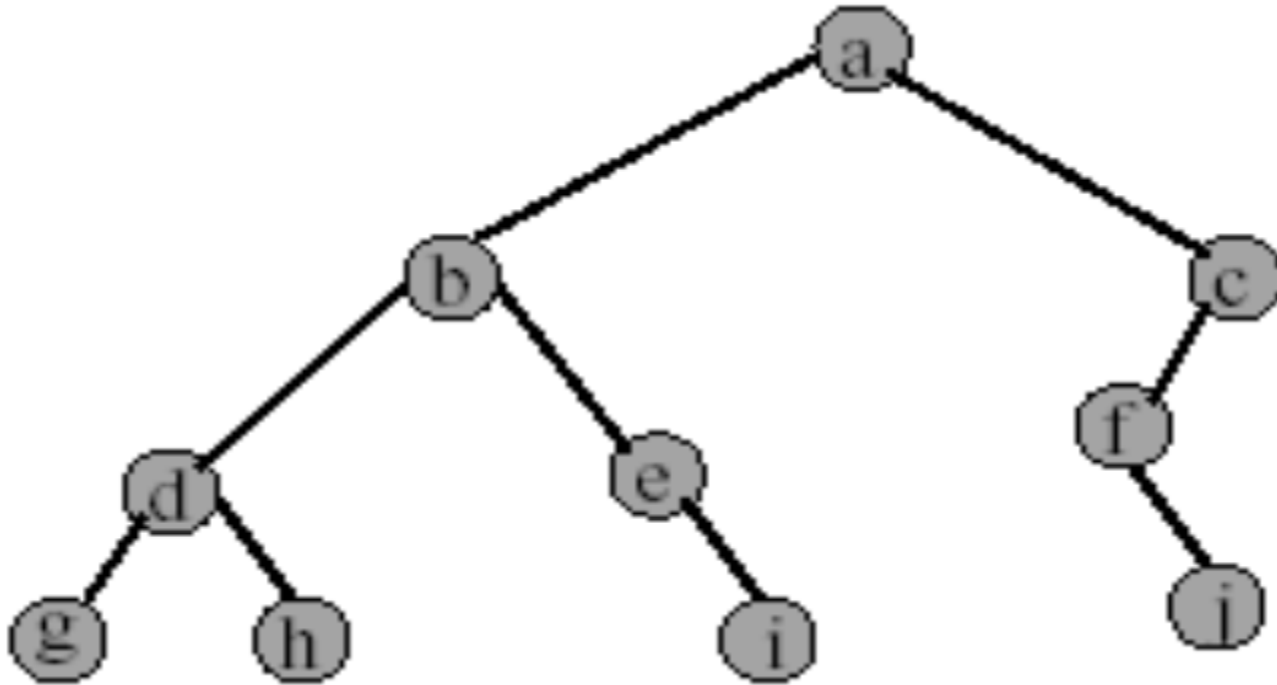
Preorder of Expression Tree



/ * + a b - c d + e f

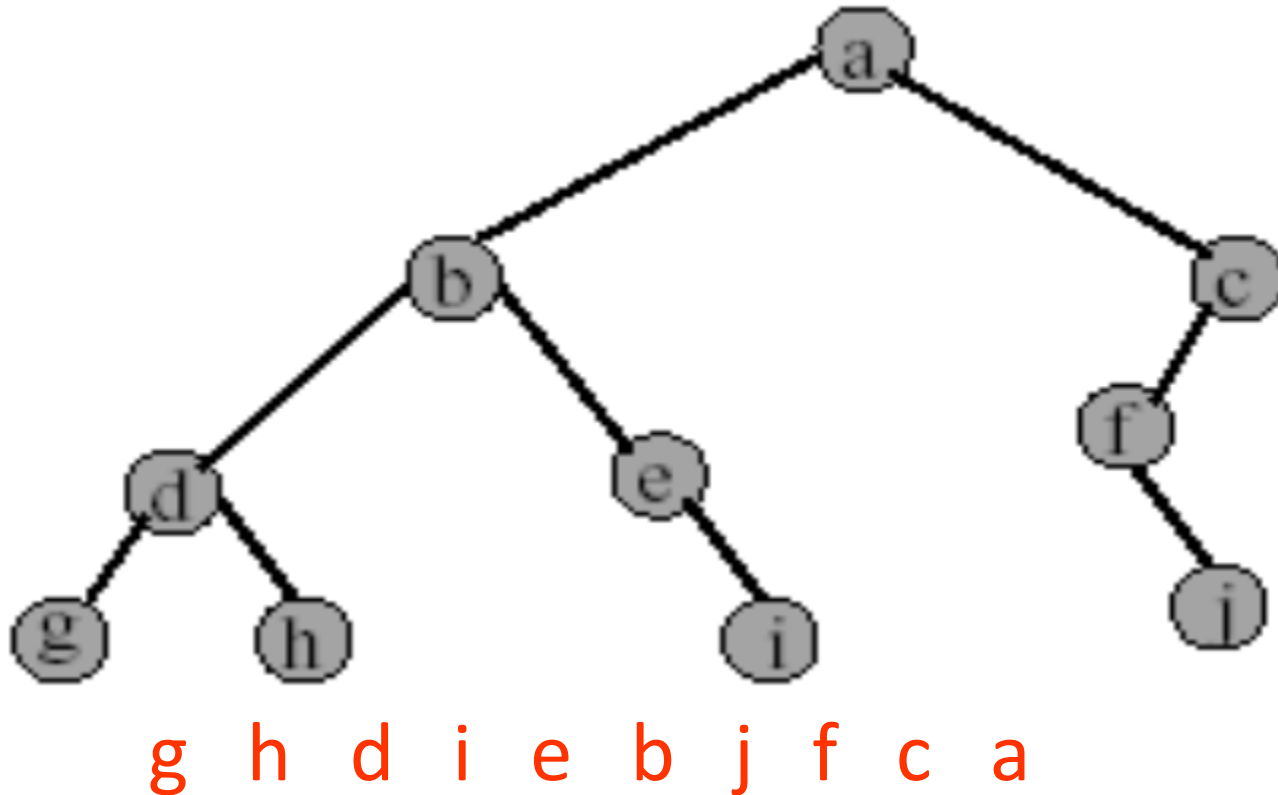
Gives **prefix** form of expression.

Inorder Example (visit = print)

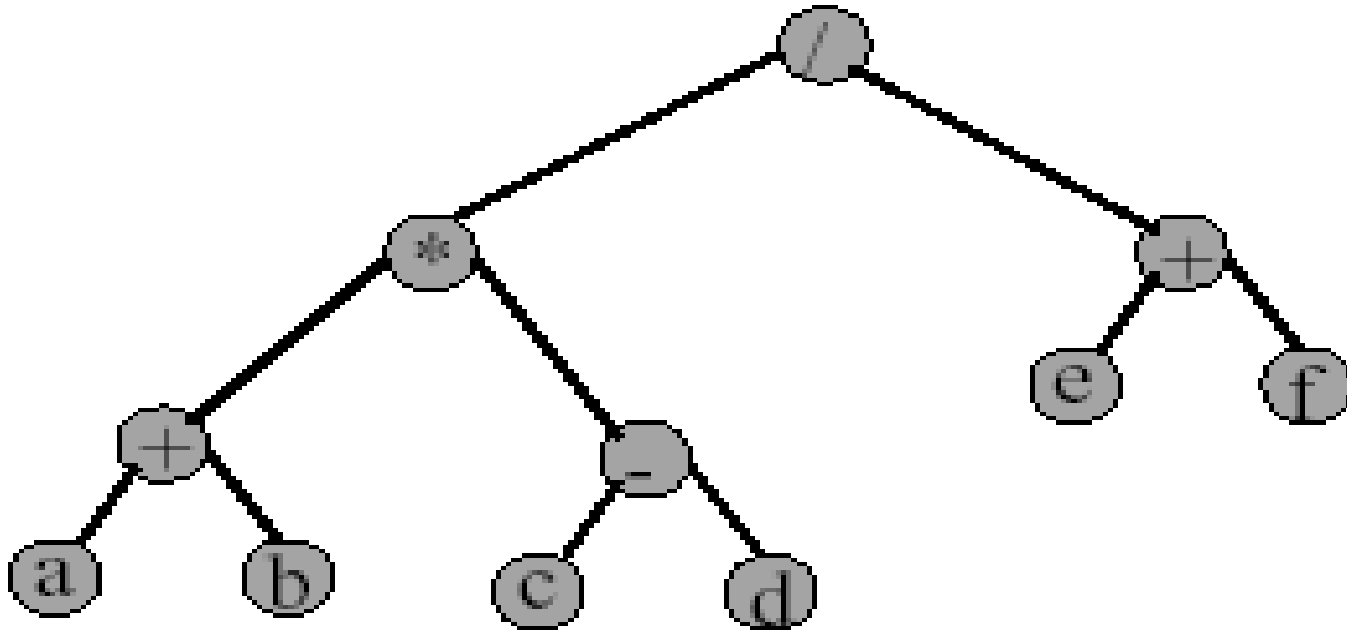


g d h b e i a f j c

Postorder Example (visit = print)



Postorder of Expression Tree



a b + c d - * e f + /

Gives **postfix** form of expression.