Data Structure

تركيب بيانات
الفرقة الثالثة علوم إحصاء وعلوم الحاسب

By
Dr. Reda Elbarougy
د/ رضا الباروجى
Lecturer of computer sciences
In Mathematics Department
Faculty of Science
Damietta University

الأحد ٢٣-٠٤-٢٠١٧

رقم المحاضرة

م	التاريخ	الموضوع	الحالة	
١	٢٠١٧-٠٢-١٢	الفصل الأول مقدمة	تم	
٢	٢٠١٧-٠٢-١٩	الفصل الأول مقدمة	تم	
٣	٢٠١٧-٠٣-٠٥	الفصل الثانى اساسيات	تم	
٤	٢٠١٧-٠٣-١٢	الفصل الرابع المصفوفات	تم	
٥	٢٠١٧-٠٣-١٩	الفصل الرابع المصفوفات	تم	
٦	٢٠١٧-٠٣-٢٦	الفصل الخامس		
٧	٢٠١٧-٠٤-٠٢	الفصل الخامس		
٨	٢٠١٧-٠٤-٠٤	الفصل السادس	أضافى	Stacks
٩	٢٠١٧-٠٤-٠٩	الفصل السادس		
	٢٠١٧-٠٤-١٦	أجازة		
١٠	٢٠١٧-٠٤-١٩	لا		
١١	٢٠١٧-٠٤-٢٣	الفصل السابع		
١٢	٢٠١٧-٠٤-٣٠			
	٢٠١٧-٠٥-٠٧			

Chapter 7:Tree

Chapter 7

- General Tree
- Binary Trees
- Representing Binary Trees In Memory
- TRAVERSING BINARY TREES

Trees

Linear types of data structures:

- Arrays,
- Lists,
- Stacks and Queues.

Now we defines a nonlinear data structure called *Tree.*

This structure is mainly used to represent data containing a hierarchical relationship between nodes/elements e.g. family trees and tables of contents.

There are two main types of tree:

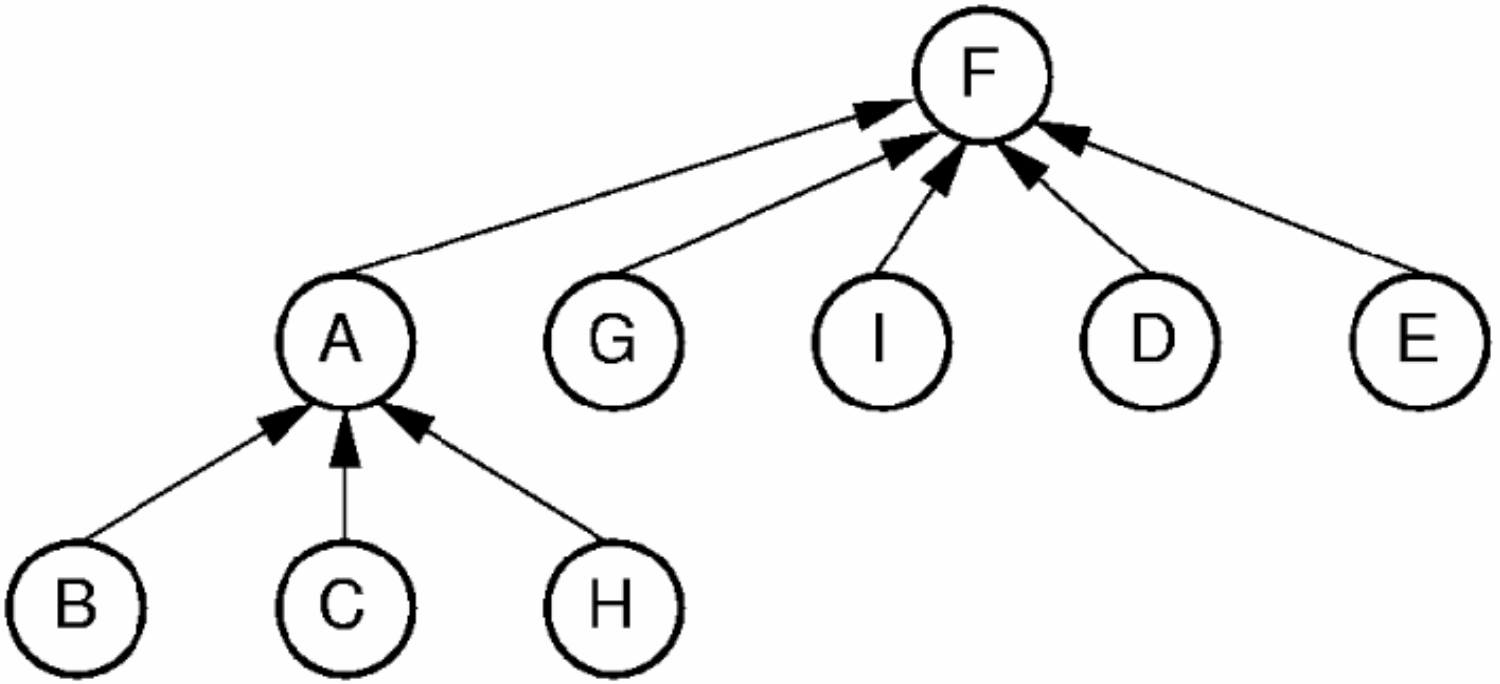
- General Tree
- Binary Tree

Trees

General Tree:

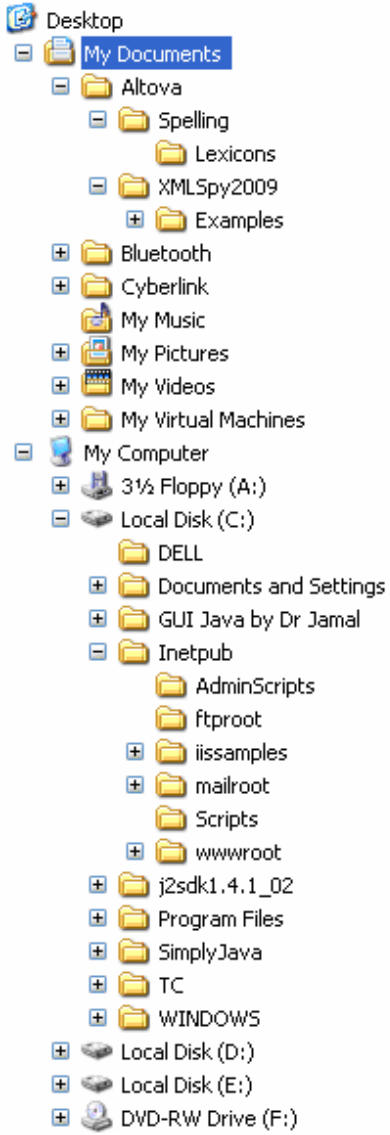
A tree where a node can has any number of children / descendants is called General Tree.

For example:



Trees

Following figure is also an example of general tree where *root is "Desktop".*

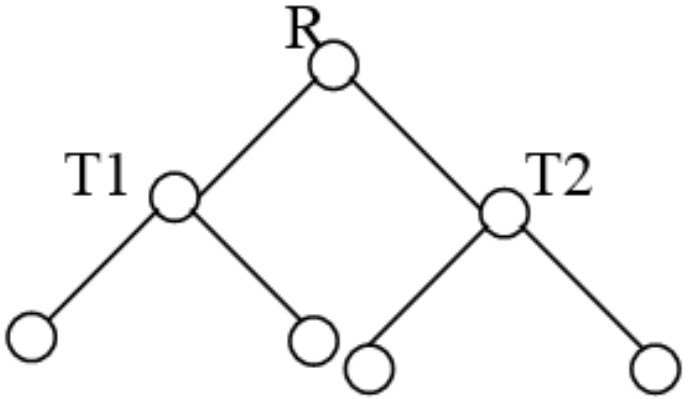


Trees

This following figure is a general tree where *root is "Visual Programming".*

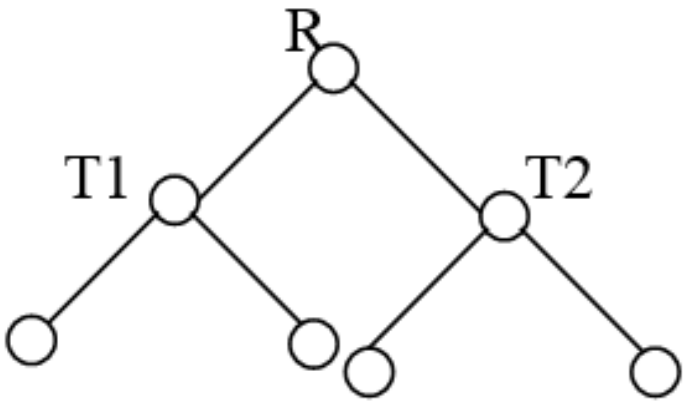
7.2 Binary Trees

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.



Binary Trees

- 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.



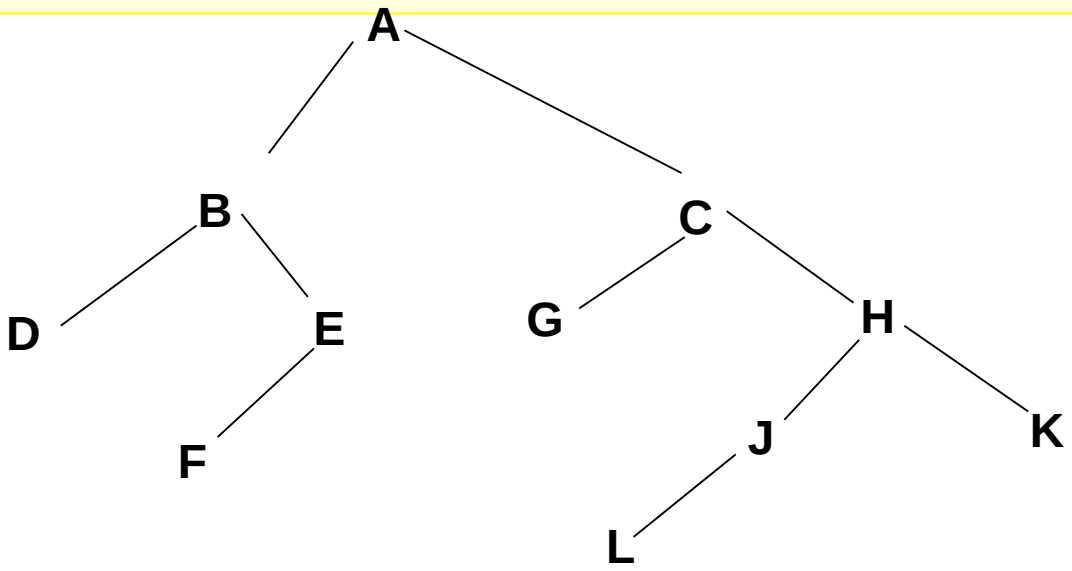
Binary Trees

A binary tree T is frequently presented by a diagram in the plane called a picture of T . Specifically, the diagram in Fig. 7-1 represents a binary tree as follows:

i.T consists of 11 nodes, represented by the letters A through L, excluding I.

ii.The root of T is the node A at the top of the diagram.

iii.A left-downward slanted line at a node N indicates a left successor of N; and a right-downward slanted line at N indicates a right successor of N.

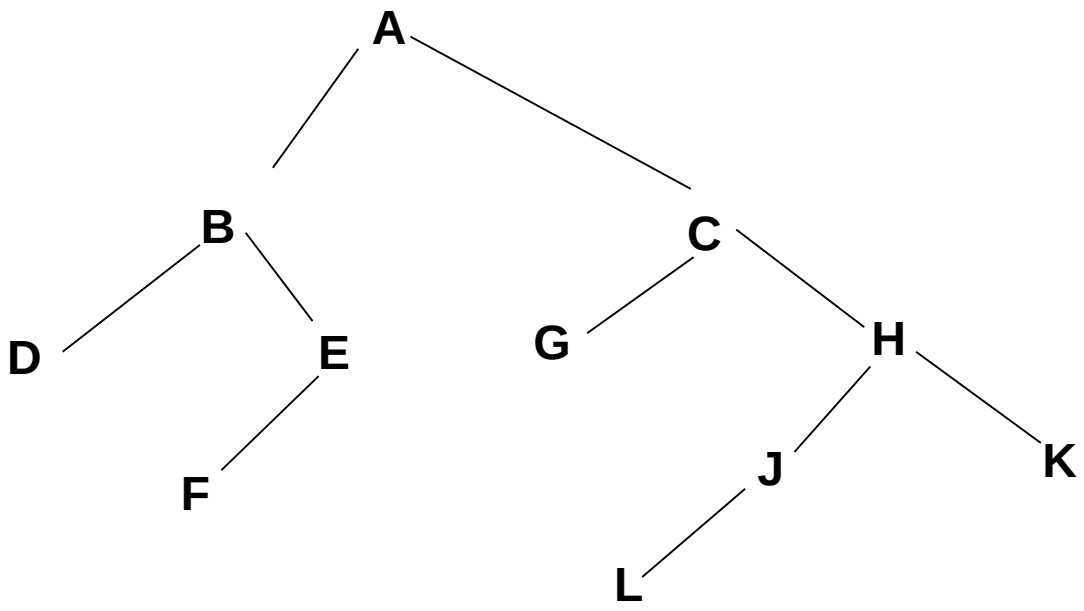


Binary Trees

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.



Binary Trees

- Any node N in a binary tree T has either 0, 1 or 2 successor.
- 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 trees T1 and T2.
- 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.

Similar Binary Trees

- 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.

Example 7.1

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.

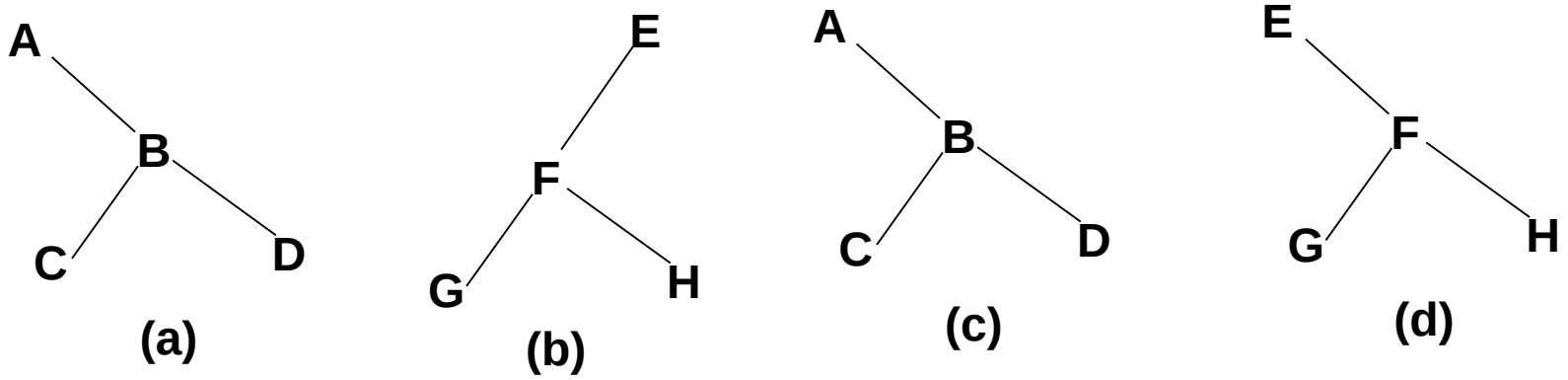
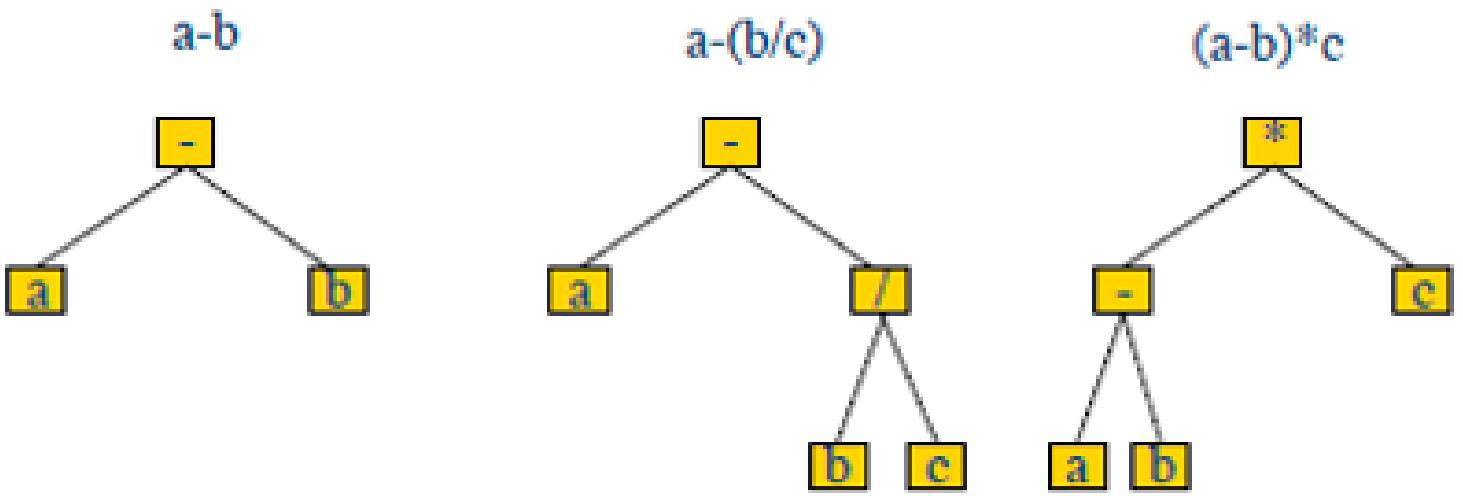


Fig. 7.2

Arithmetic Expressions as trees

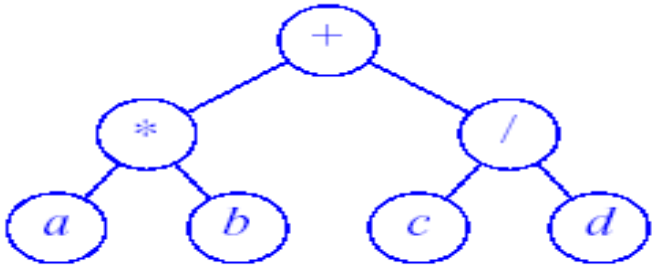


Arithmetic expressions are often represented as binary trees.

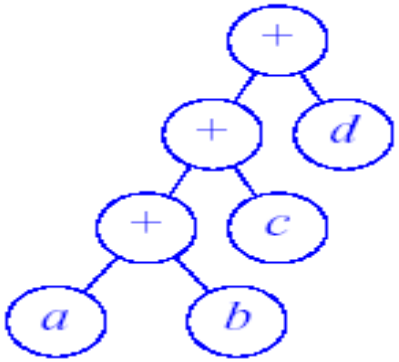
Internal nodes are **operations** - Leaves are **numbers/variables**.

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

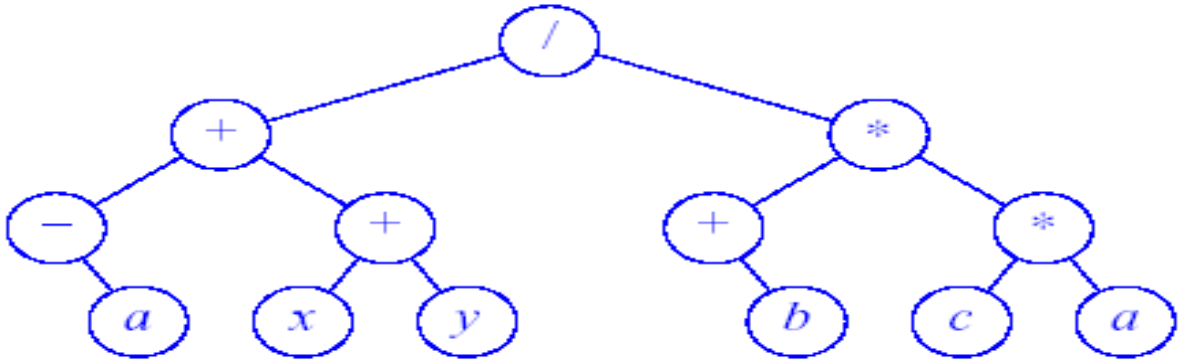
Arithmetic Expressions as trees(cont.)



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



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



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

Figure 11.5 Expression Trees

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.
- That is, each variable or constant in E appears as an "internal" node in T whose left and right subtrees correspond to the operands of the operation.

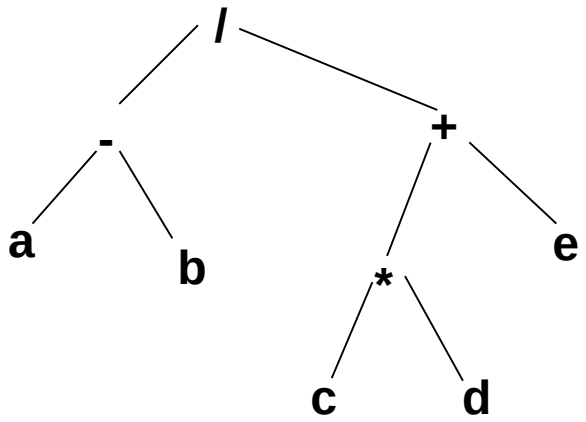


Fig. 7.3: E = (a – b) / ((c * d) + e)

Example 7.2: Algebraic Expressions

For example:

a) In the expression E, the operands of + are c * d and e.

b) In the tree T, the subtrees of the node + correspond to the subexpression c * d and e.

Every algebraic expression will correspond to a unique tree, and vice versa.

TERMINOLOGY

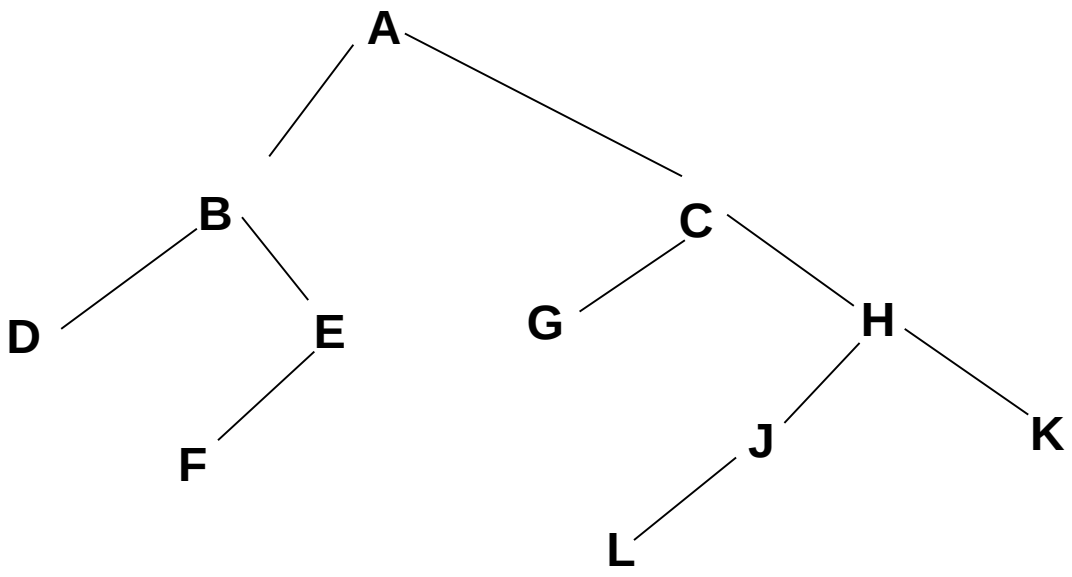
- Terminology describing family relationships is frequently used to describe relationships between the nodes of a tree T.
- Specifically, suppose N is a node in T with left successor S_1 and right successor S_2. Then N is called the *parent* or *father* of S_1 and S_2.
- Analogously, S_1 is called the *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.
- The terms descendant and ancestor have their usual meaning. That is, a node L is called a *descendant* of a node N (and N is called an *ancestor* of L) if there is a succession of children from N to L. In particular, L is called a left or *right descendant* of N according to whether L belongs to the left or right subtree of N.

TERMINOLOGY

- Terminology from graph theory and horticulture are also used with a binary tree T .
- Specifically, the line drawn from a node N of T to a successor is called an *edge*, and a sequence of consecutive edges is called a *path*.
- A terminal node is called a *leaf*, and a path ending in a leaf is called a *branch*.
- Each node in a binary tree T is assigned a *level number*, as follows. The root R of the tree T is assigned the level number 0, and every other node is assigned a level number which is 1 more than the level number of its parent.
- Furthermore, those nodes with the same level number are said to belong to the same generation.

TERMINOLOGY

- The *depth* (or *height*) of a tree T is the maximum number of nodes in a branch of T . This turns out to be 1 more than the largest level number of T . The tree T in Fig. 7-1 has depth 5.



Representing Binary Trees In Memory

- Let T be a binary tree. This section discusses the two ways of representing T in memory.

i. Linked representation of T.

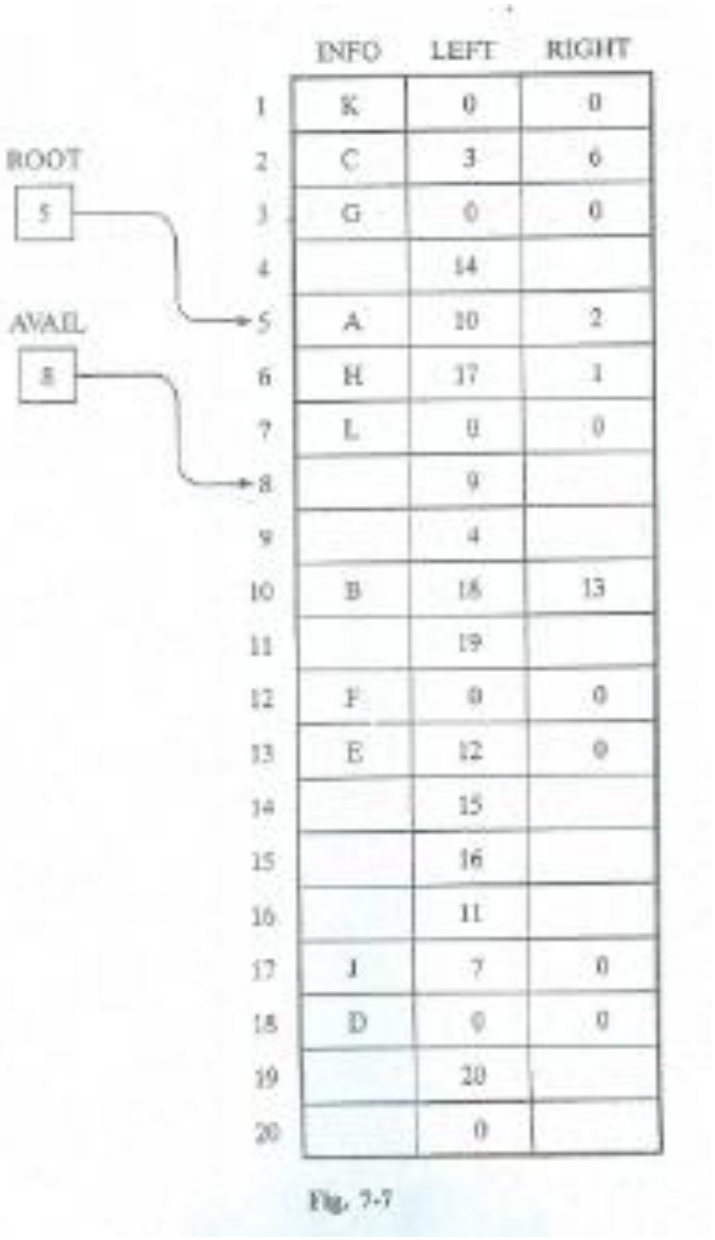
ii. Sequential representation of T.

- The main requirement of any representation of T is that
  a) one should have direct access to the root R of T and,
  b) given any node N of T, one should have direct access to the children of N.

Linked Representation of Binary Tree

- The most popular way to present a binary tree is linked representation.
- The Linked representations of tree, maintains three parallel arrays.
- An INFO array contains the data of each node, LEFT array contains the location of left child and RIGHT array contains location of right child.
- A ROOT pointer points to the root node of the tree.
- Each element is represented by a node that has two link fields *LEFT* (leftChild) and RIGHT (rightChild) plus an Info field
- The space required by an *n* node binary tree is *n* * sizeof(binaryTreeNode)

Linked Representation of Binary Tree



	INFO	LEFT	RIGHT
1	K	0	0
2	C	3	6
3	G	0	0
4		14	
5	A	10	2
6	H	17	1
7	L	0	0
8		9	
9		4	
10	B	18	13
11		19	
12	F	0	0
13	E	12	0
14		15	
15		16	
16		11	
17	J	7	0
18	D	0	0
19		20	
20		0	

Fig. 7-7

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.**

Linked Representation of Binary Tree

1. Furthermore, ROOT will contain the location of the root R of T.
2. If any subtree is empty, then the corresponding pointer will contain the null value; if the tree T itself is empty, then ROOT will contain the null value.

Example 7.3

Consider the binary tree T in Fig. 7-1. A schematic diagram of the linked representation of T appears in Fig. 7-6. Observe that each node is pictured with its three fields, and that the empty subtrees are pictured by using × for the null entries. Figure 7-7 shows how this linked representation may appear in memory. The choice of 20 elements for the arrays is arbitrary. Observe that the AVAIL list is maintained as a one-way list using the array LEFT.

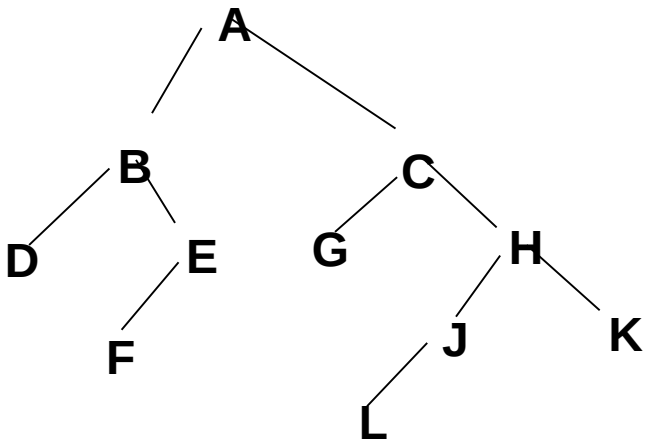


Fig : 7-1

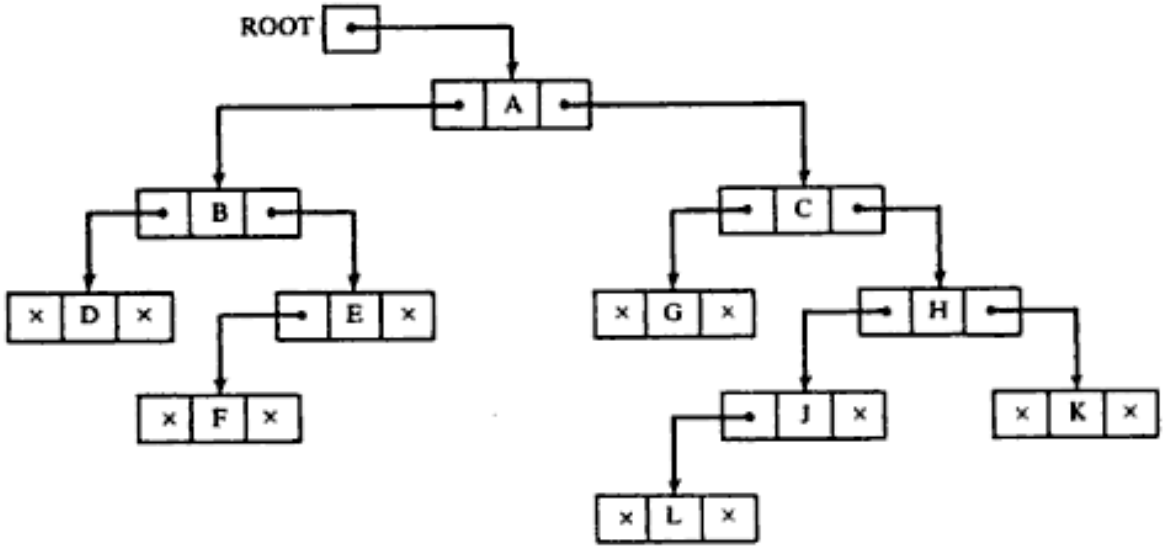


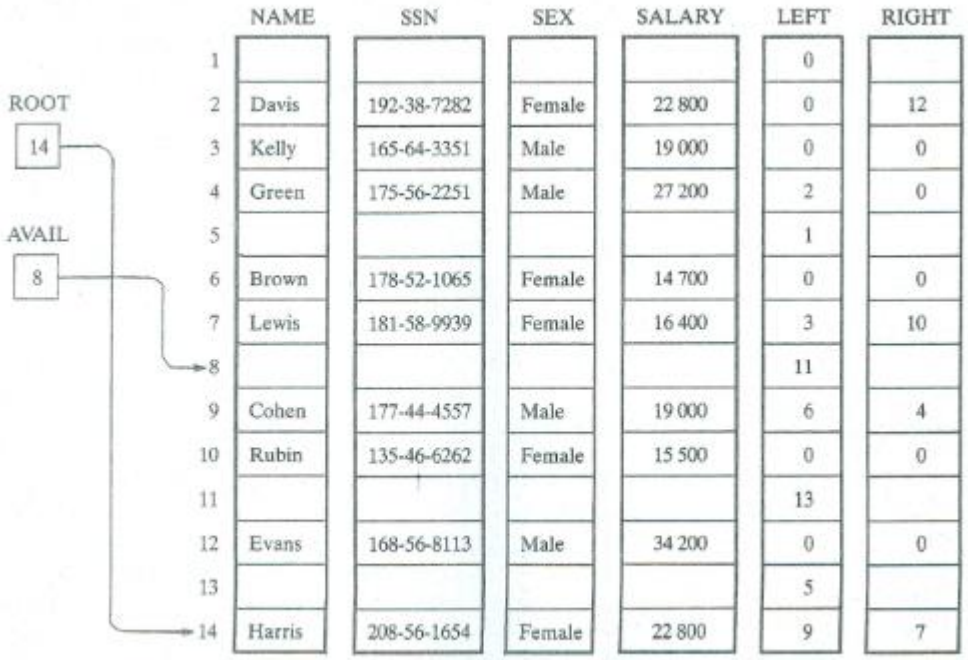
Fig : 7-6

Example 7.4

Suppose the personnel file of a small company contains the following data on its nine employees:

Name, Social Security Number, Sex, Monthly Salary

Figure 7-8 shows how the file may be maintained in memory as a binary tree. Compare this data structure with Fig. 5-12, where the exact same data are organized as a one-way list.



	NAME	SSN	SEX	SALARY	LEFT	RIGHT
1					0	
2	Davis	192-38-7282	Female	22 800	0	12
3	Kelly	165-64-3351	Male	19 000	0	0
4	Green	175-56-2251	Male	27 200	2	0
5					1	
6	Brown	178-52-1065	Female	14 700	0	0
7	Lewis	181-58-9939	Female	16 400	3	10
8					11	
9	Cohen	177-44-4557	Male	19 000	6	4
10	Rubin	135-46-6262	Female	15 500	0	0
11					13	
12	Evans	168-56-8113	Male	34 200	0	0
13					5	
14	Harris	208-56-1654	Female	22 800	9	7

Fig. 7-8

Example 7.4

Suppose we want to draw the tree diagram which corresponds to the binary tree in Fig. 7-8. For notational convenience, we label the nodes in the tree diagram only by the key values NAME. We construct the tree as follows:

(*a*) The value ROOT = 14 indicates that Harris is the root of the tree.

(*b*) LEFT[14] = 9 indicates that Cohen is the left child of Harris, and RIGHT[14] = 7 indicates that Lewis is the right child of Harris.

Repeating Step (*b*) for each new node in the diagram, we obtain Fig. 7-9.

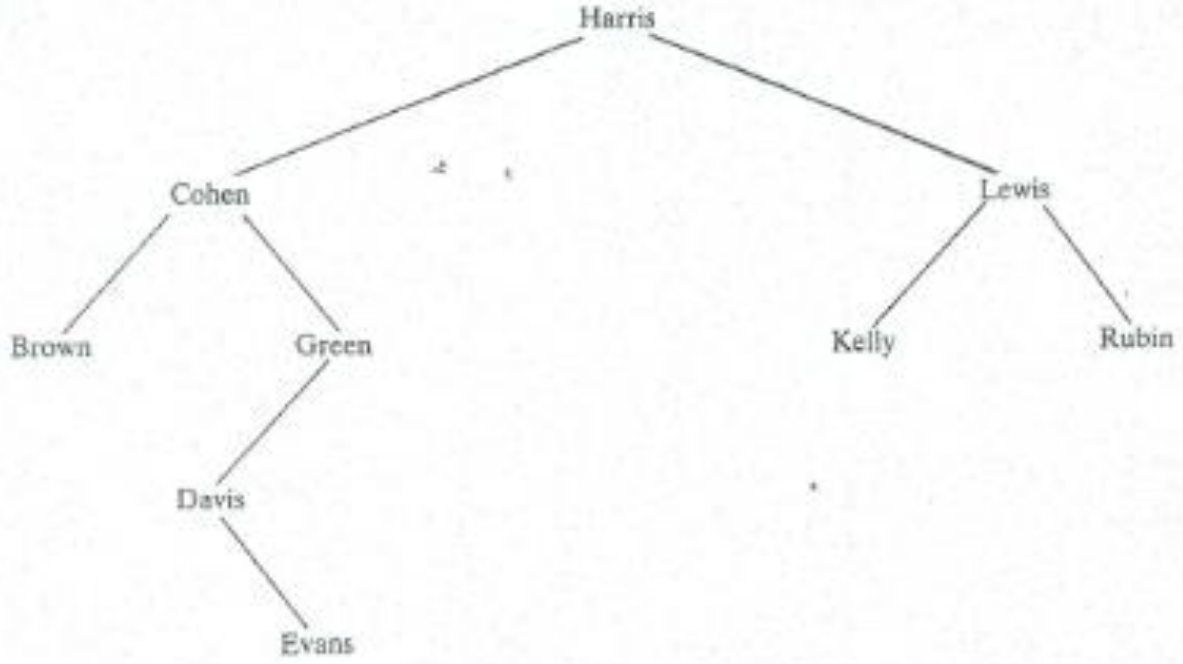


Fig. 7-9

Binary Trees: Array representation

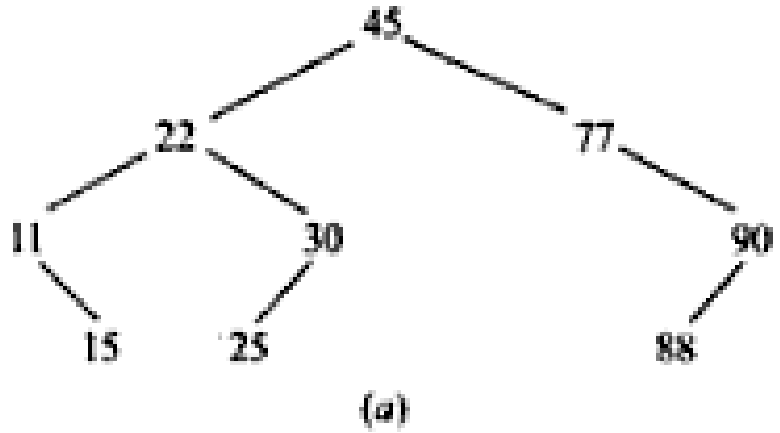
- Suppose T is a binary tree that is complete or nearly complete. Then there is an efficient way of maintaining T in memory called the *sequential representation* of T.
- This representation uses only a single linear array TREE together with a pointer variable END as follows:

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.

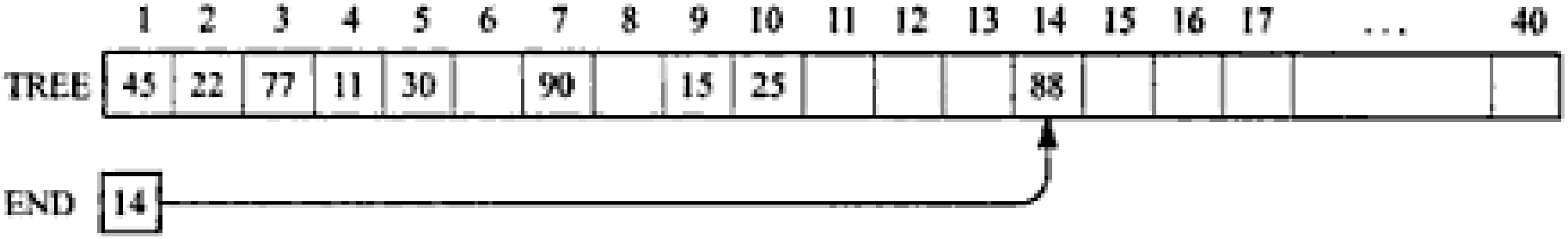
NULL is used to indicate an empty subtree. TREE[1] = NULL indicates that the tree is empty.

The sequential representation of a tree with depth d will require an array with approximately 2^{d+1} elements. The sequential representation is usually inefficient unless, the binary tree T is complete or nearly complete.

Binary Trees: Array representation



(a)



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:

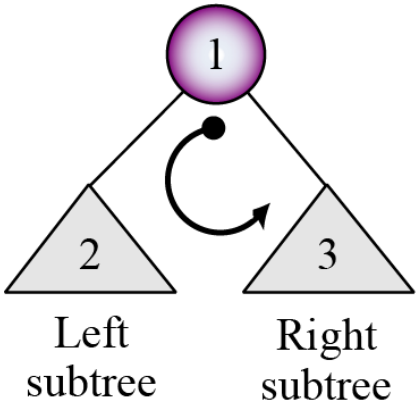
1. Additions and deletions of nodes are inefficient, because of the data movements in the array.
2. Space is wasted if the binary tree is not complete. That is, the linear representation is useful if the number of missing nodes is small.

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.

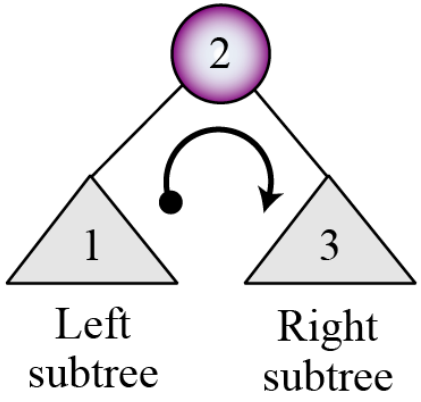
TRAVERSING BINARY TREES

- there are three standard ways of traversing a binary tree T with root R. They are

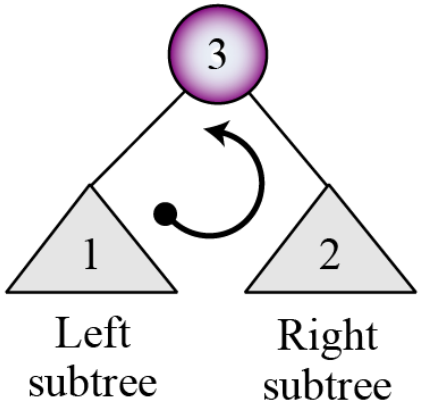
1) Preorder.
2) Inorder and
3) Postorder.



a. Preorder traversal



b. Inorder traversal

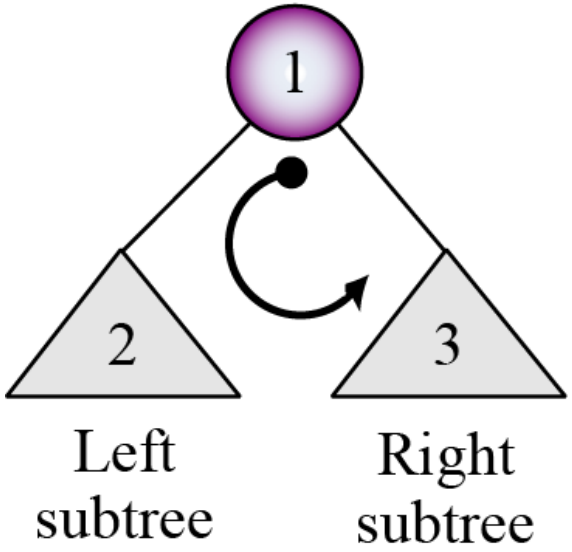


c. Postorder Ttraversal

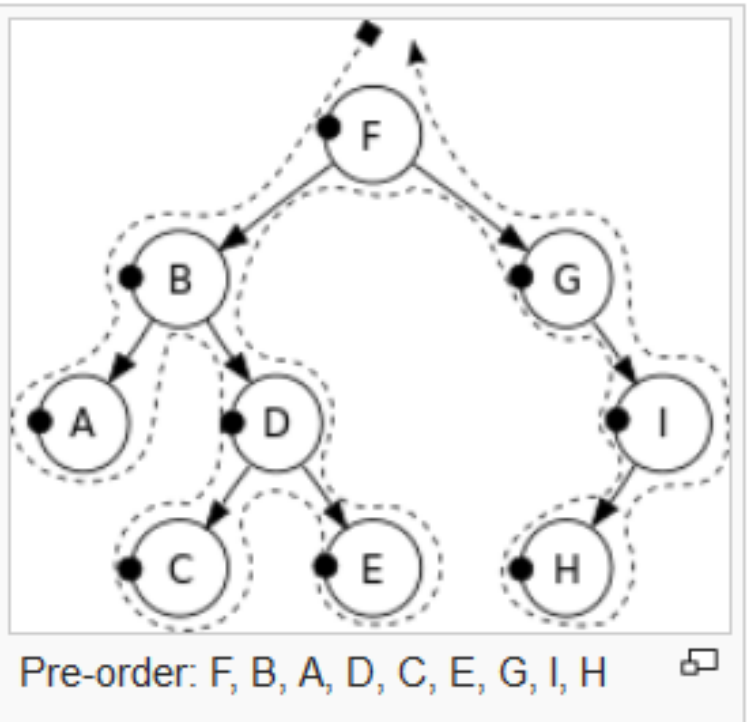
TRAVERSING BINARY TREES

Preorder: is also called Node-Left-Right (NLR)

1. Process the root R
2. Traverse the left subtree of R in preorder.
3. Traverse the right subtree of R in preorder.



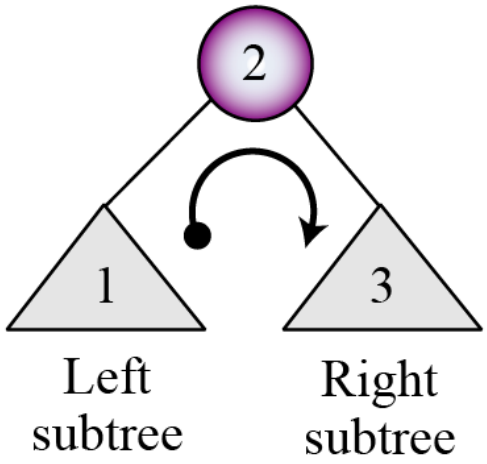
a. Preorder traversal



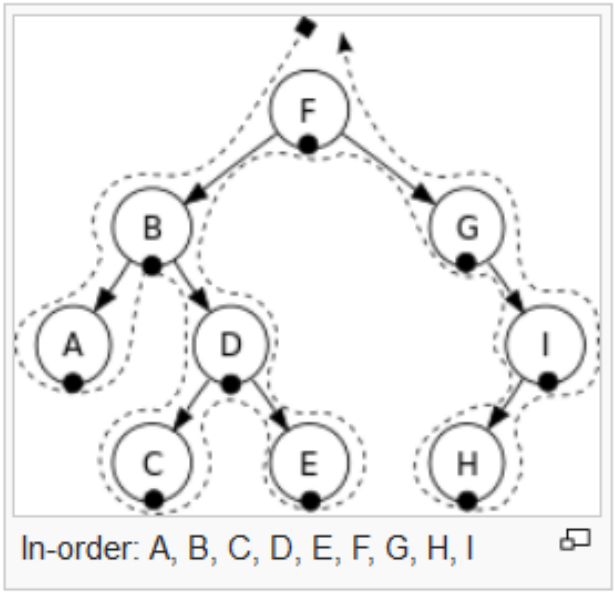
TRAVERSING BINARY TREES

Inorder: is also called Left-Node-Right (LNR)

1. Traverse the left subtree of R in inorder
2. Process the root R.
3. Traverse the right subtree of R in inorder.



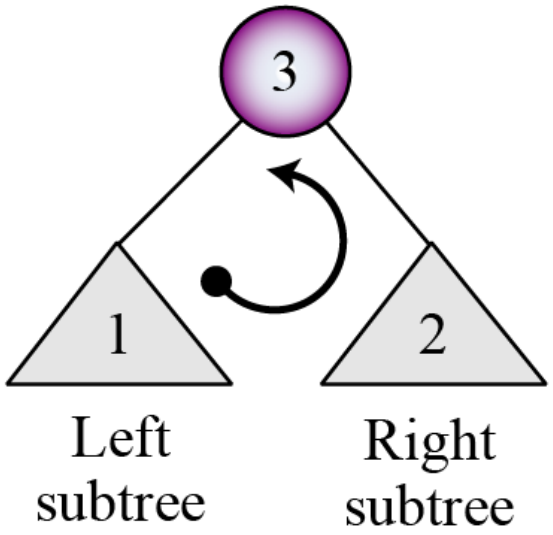
b. Inorder traversal



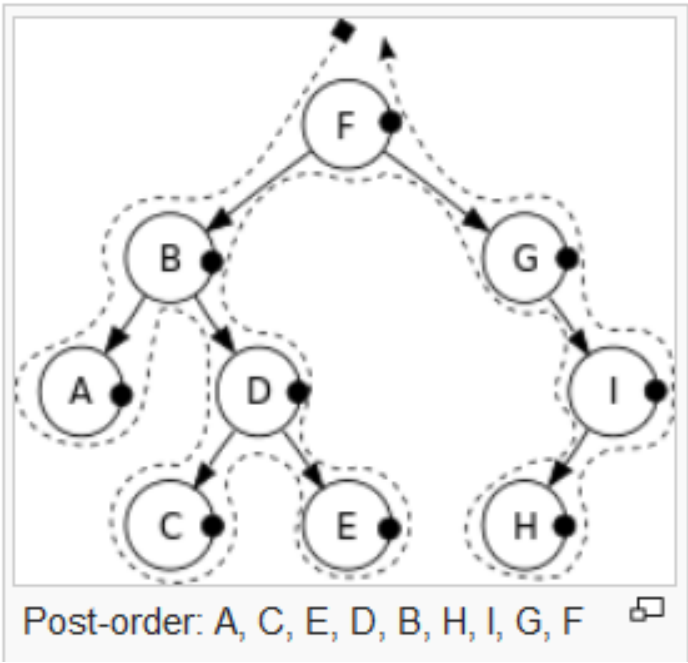
TRAVERSING BINARY TREES

PostOrder: is also called Left-Right-Node (LRN)

1. Traverse the left subtree of R in postorder
2. Traverse the right subtree of R in postorder
3. Process the root R.

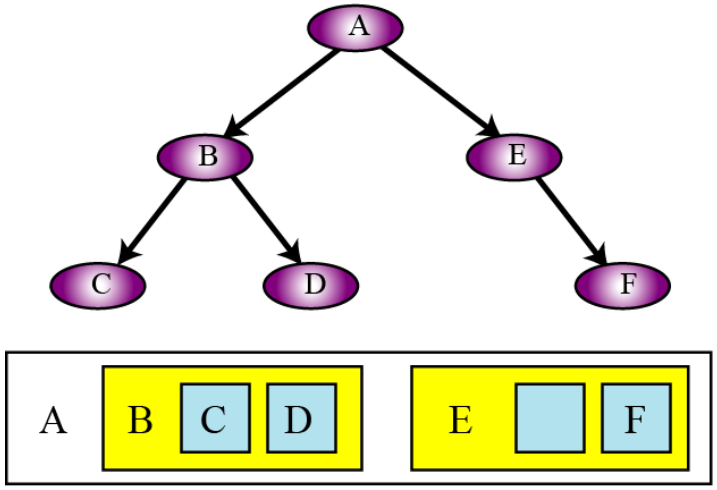


c. Postorder Ttraversal

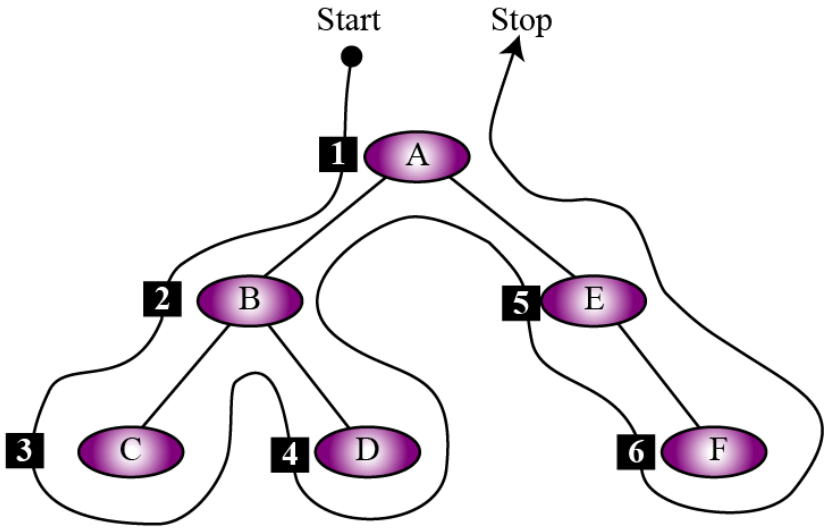


Example

This figure shows how we visit each node in a tree using preorder traversal. The figure also shows the walking order. In preorder traversal we visit a node when we pass from its left side. The nodes are visited in this order: A, B, C, D, E, F.



a. Processing order



b. "Walking" order

Example

Example 7.5

Consider the binary tree T in Fig. 7-11. Observe that A is the root, that its left subtree L_T consists of nodes B, D and E and that its right subtree R_T consists of nodes C and F.

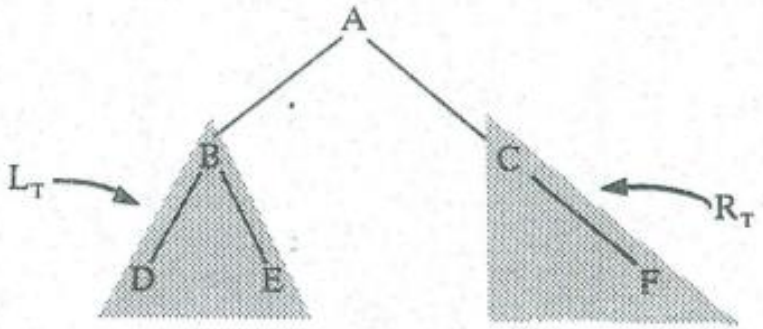


Fig. 7-11

(*a*) The preorder traversal of T processes A, traverses L_T and traverses R_T. However, the preorder traversal of L_T processes the root B and then D and E, and the preorder traversal of R_T processes the root C and then F. Hence ABDECF is the preorder traversal of T.

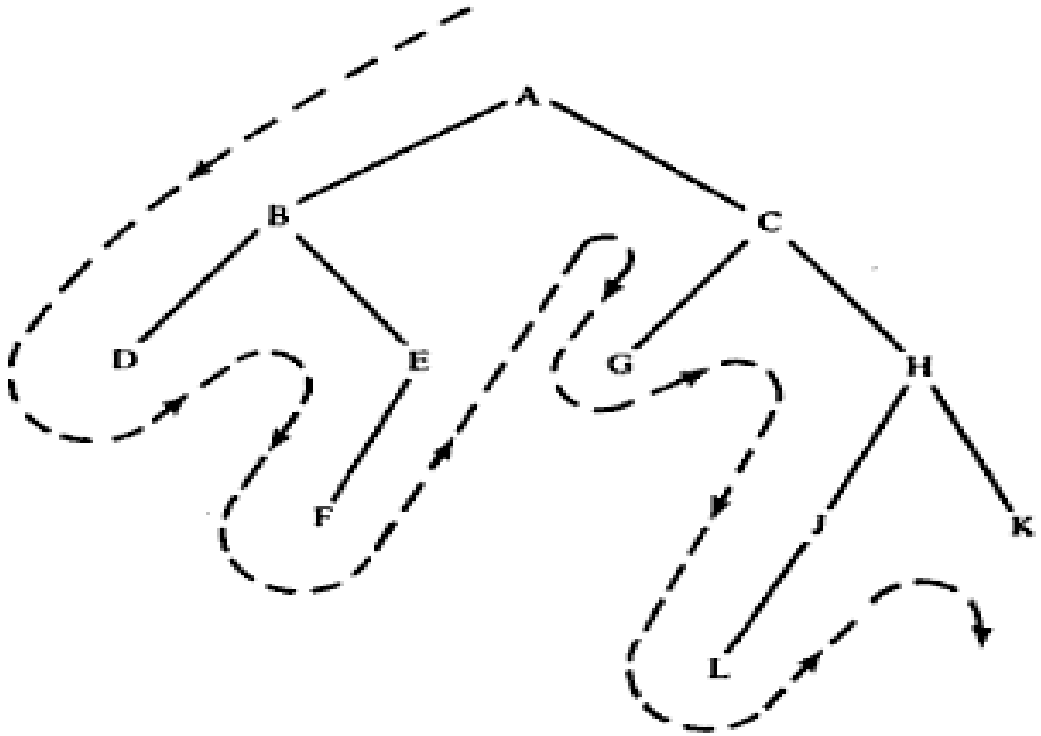
(*b*) The inorder traversal of T traverses L_T, processes A and traverses R_T. However, the inorder traversal of L_T processes D, B and then E, and the inorder traversal of R_T processes C and then F. Hence DBEACF is the inorder traversal of T.

(*c*) The postorder traversal of T traverses L_T, traverses R_T, and processes A. However, the postorder traversal of L_T processes D, E and then B, and the postorder traversal of R_T processes F and then C. Accordingly, DEBFCA is the postorder traversal of T.

Example 7.6

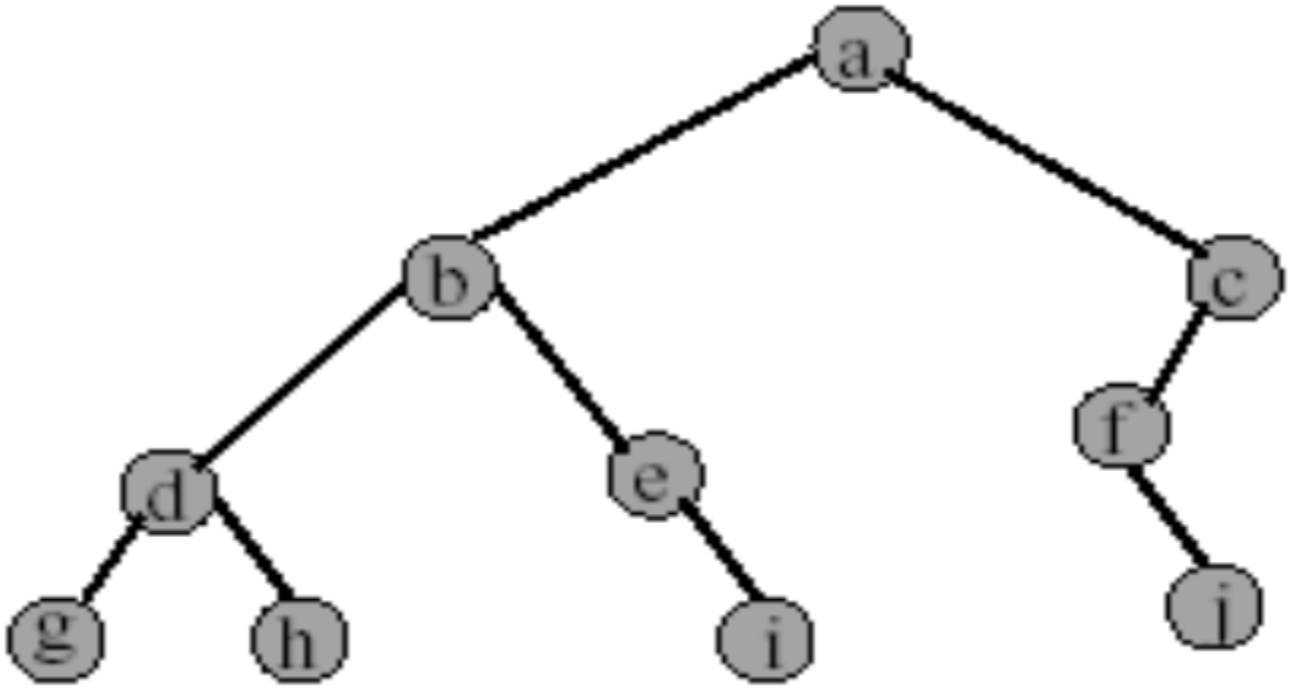
Consider the tree T in Fig. 7-12. The preorder traversal of T is ABDEFCGHJLK. This order is the same as the one obtained by scanning the tree from the left as indicated by the path in Fig. 7-12. That is, one "travels" down the left-most branch until meeting a terminal node, then one backtracks to the next branch, and so on. In the preorder traversal, the right-most terminal node, node K, is the last node scanned. Observe that the left subtree of the root A is traversed before the right subtree, and both are traversed after A. The same is true for any other node having subtrees, which is the underlying property of a preorder traversal.

(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



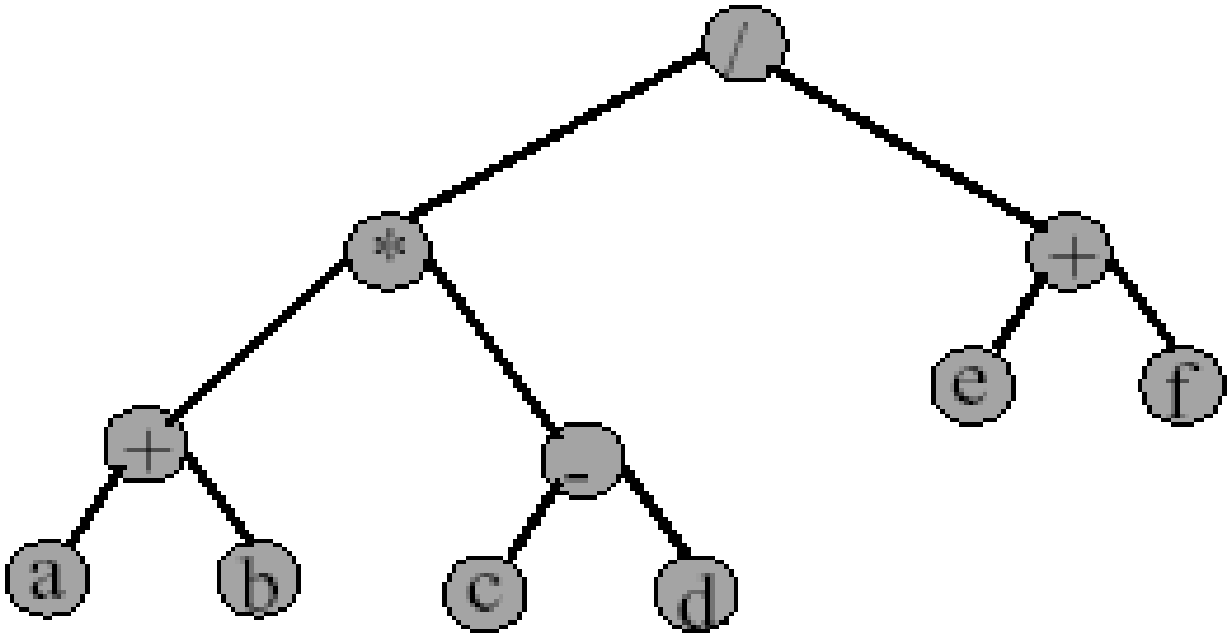
Example

Preorder Example (visit = print)



a b d g h e i c f j

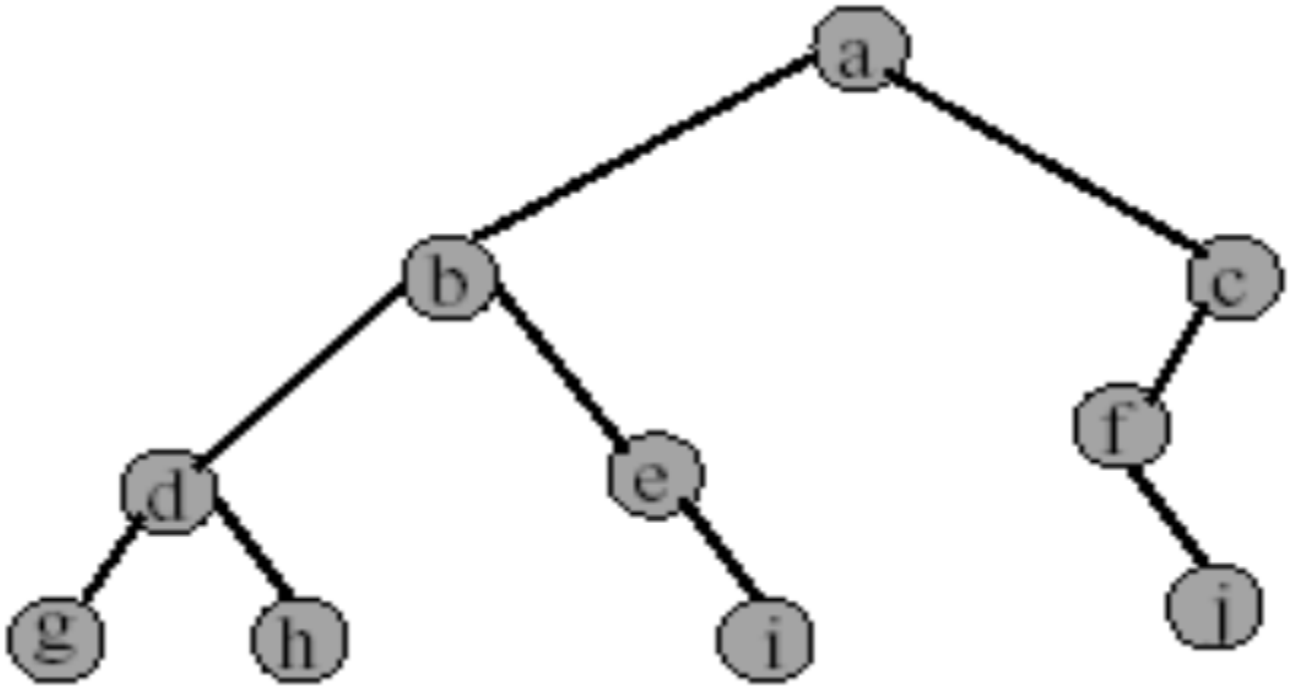
Preorder of Expression Tree



/ * + a b - c d + e f

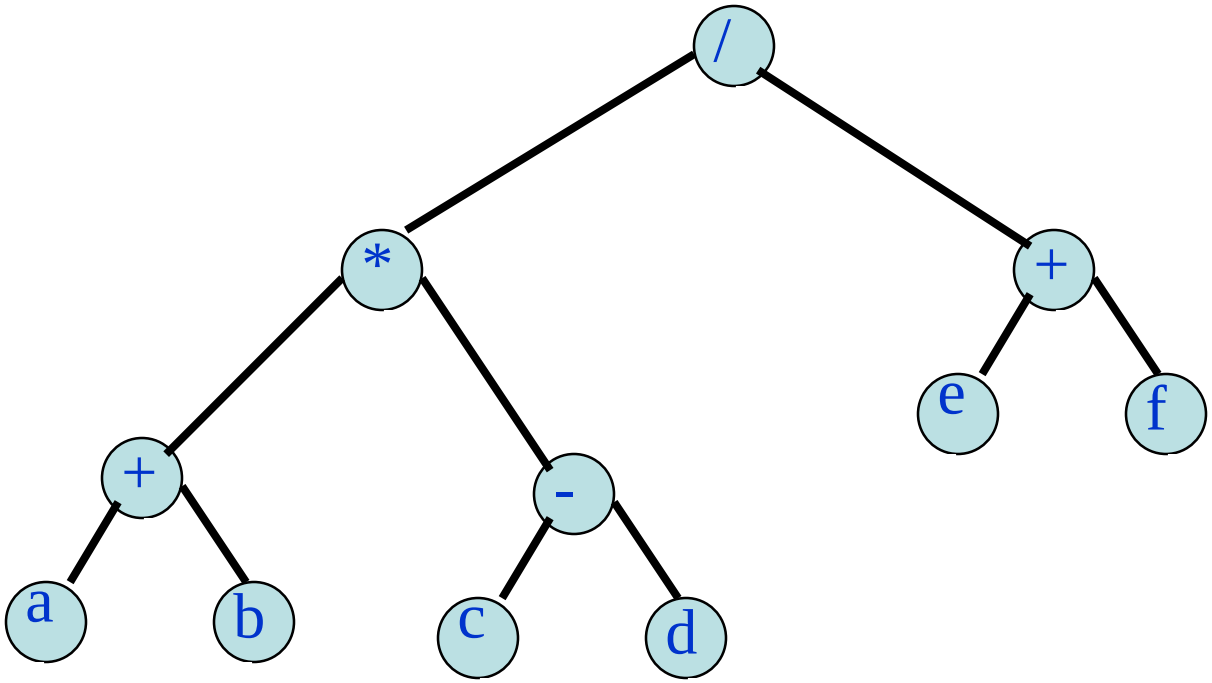
Gives prefix form of expression.

Inorder Example (visit = print)



g d h b e i a f j c

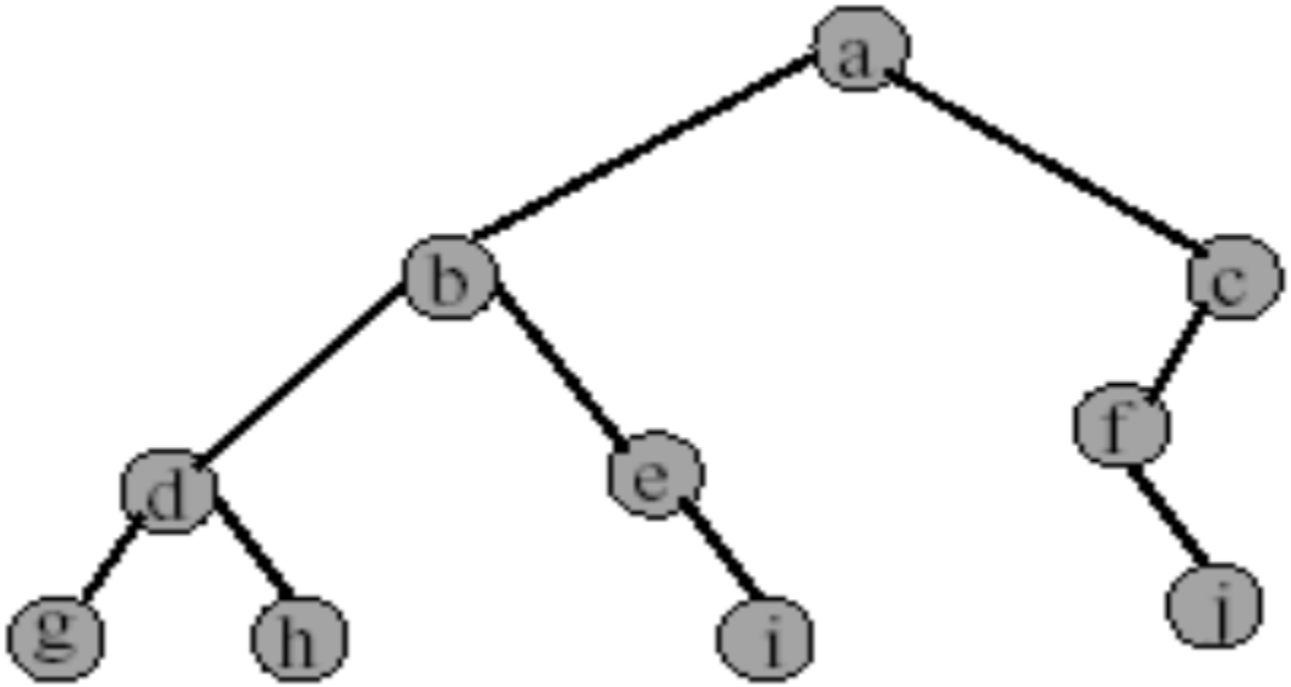
Inorder Of Expression Tree



a + b * c - d / e + f

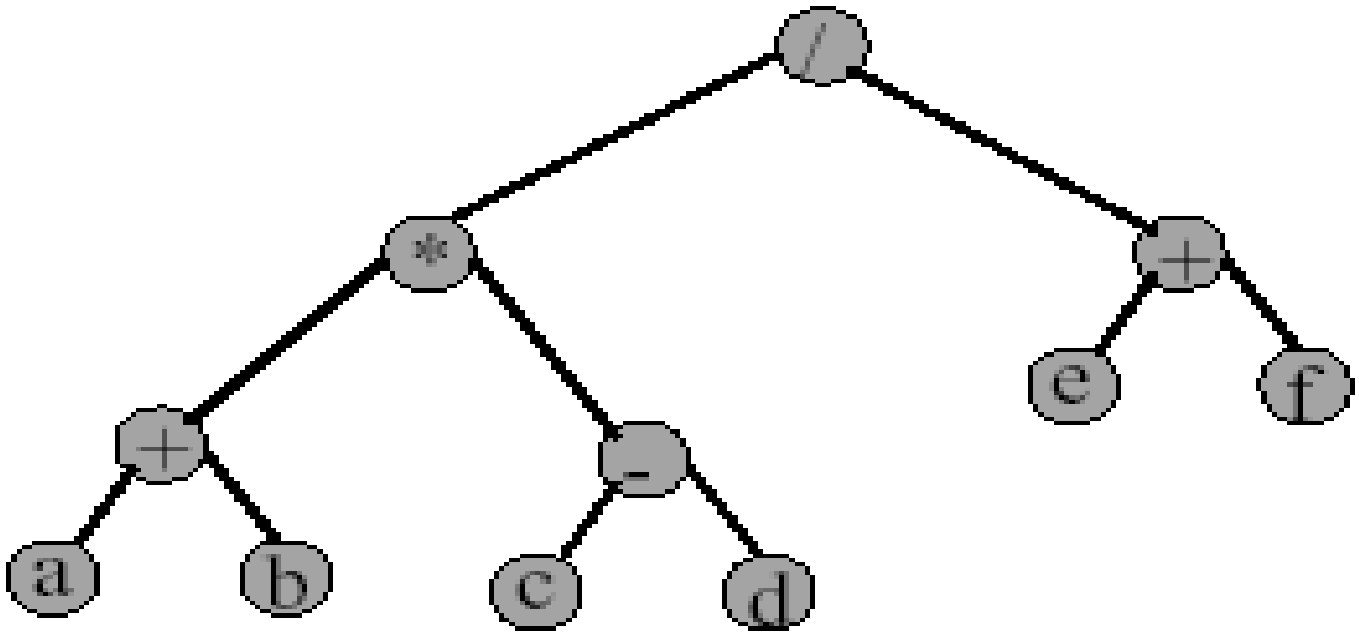
Gives infix form of expression

Postorder Example (visit = print)



g h d i e b j f c a

Postorder of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression.

Example 7.7

EXAMPLE 7.7

Let E denote the following algebraic expression:

$$[a + (b - c)] * [(d - e)/(f + g - h)]$$

The corresponding binary tree T appears in Fig. 7-13. The reader can verify by inspection that the preorder and postorder traversals of T are as follows:

(Preorder)	*	+	a	−	b	c	/	−	d	e	−	+	f	g	h
(Postorder)	a	b	c	−	+	d	e	−	f	g	+	h	−	/	*

The reader can also verify that these orders correspond precisely to the prefix and postfix Polish notation of E as discussed in Sec. 6.4. We emphasize that this is true for any algebraic expression E.

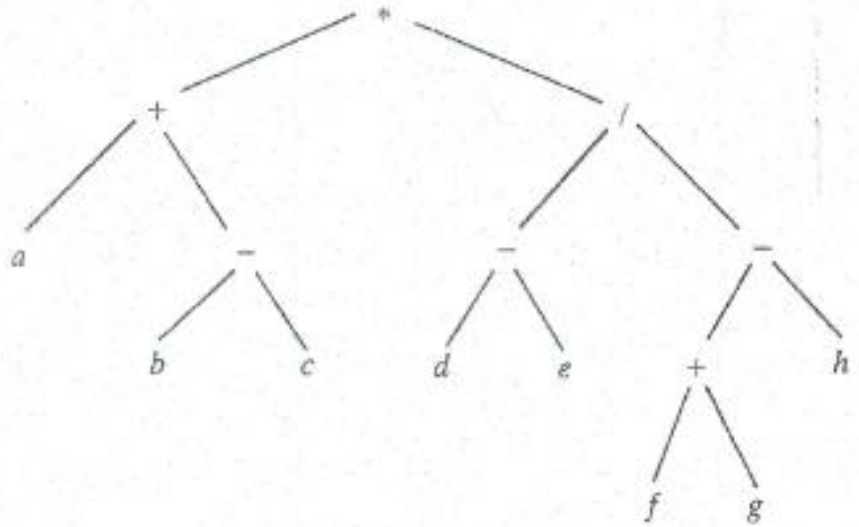


Fig. 7-13

Example 7.8

Consider the binary tree T in Fig. 7-14. The reader can verify that the postorder traversal of T is as follows:

$$S_3,\ S_6,\ S_4,\ S_1,\ S_7,\ S_8,\ S_5,\ S_2,\ M$$

One main property of this traversal algorithm is that every descendant of any node N is processed before the node N. For example, S_6 comes before S_4, S_6 and S_4 come before S_1. Similarly, S_7 and S_8 come before S_5, and S_7, S_8 and S_5 come before S_2. Moreover, all the nodes $S_1, S_2, \ldots, S_8$ come before the root M.

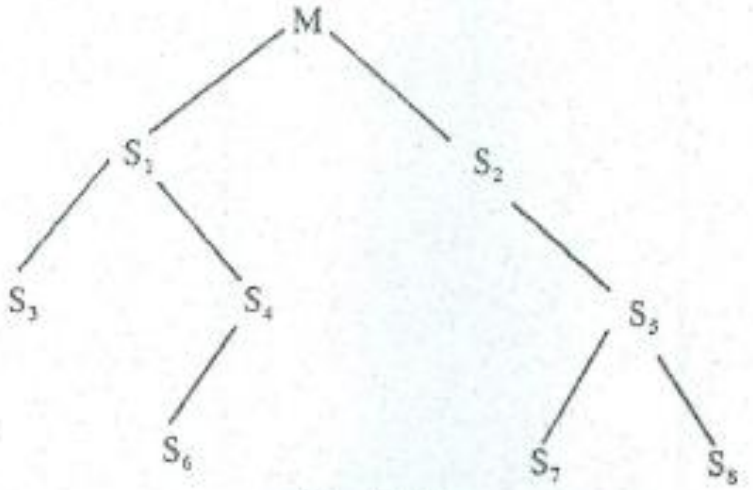


Fig. 7-14

تم الإنتهاء من المحاضرة