

Tree

Prepared by Maksuda Rabeya

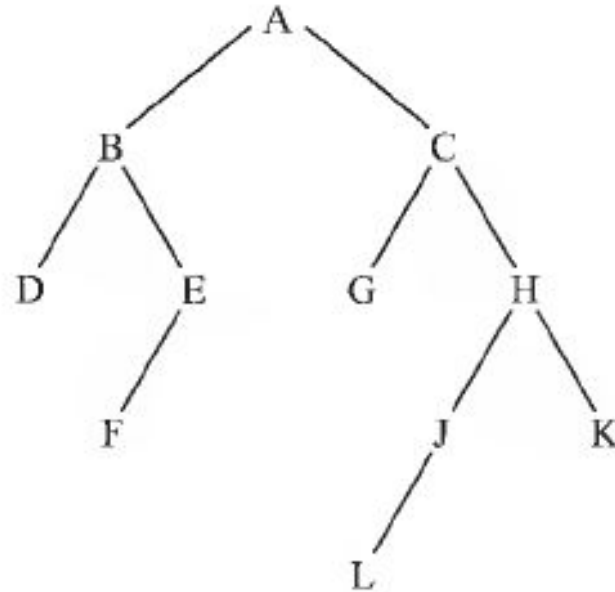
Binary Tree

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

- (1) T is empty (called the null tree or empty tree), or
- (2) 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 T_1 and T_2 .

If T does contain a root R , then the two trees T_1 and T_2 are called, respectively, the left and right subtrees of R . If T_1 is nonempty, then its root is called the left successor of R ; similarly, if T_2 is nonempty, then its root is called the right successor of R . The above definition of a 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, and either subtree or both subtrees may be empty. Thus every node N in T has 0, 1, or 2 successors. A node with no successors is called a terminal node. Thus both subtrees of a terminal node are empty.

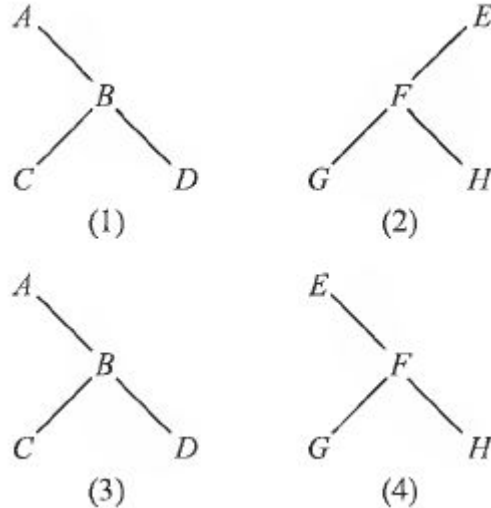
B is a left successor and C is a right successor of the root A.



Similar Binary Tree

Binary trees 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 10.1 Consider the four binary trees in Fig. 10-1(£>). The three trees (1), (3), and (4) are similar. In particular the trees (1) and (3) are copies since they also have the same data at corresponding nodes. The tree (2) is neither similar nor a copy of the tree (4) because, in a binary tree, we distinguish between a left successor and a right successor even when there is only one successor.



Terminology

- Successor
- Predecessor
- Parent
- Child
- Leaf
- Branch
- Path
- Generation
- Siblings
- Depth

Complete Binary Tree

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.

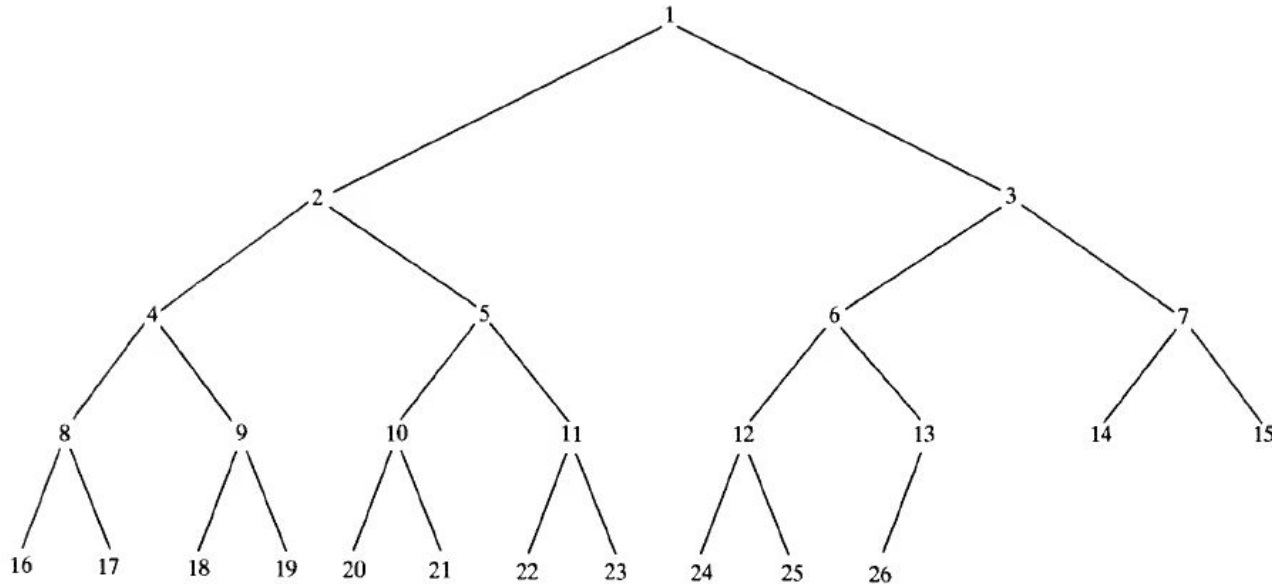
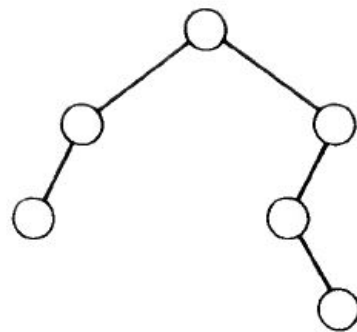


Fig. 10-2 Complete tree T_{26}

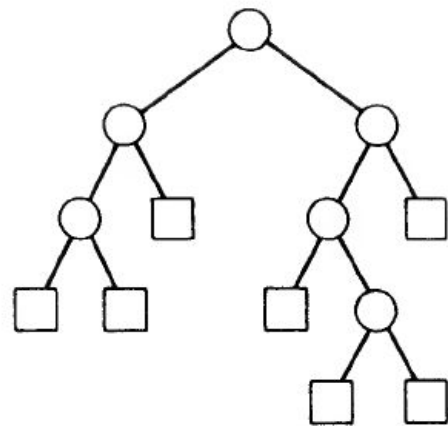
Left child = $2k$
Right child = $2k+1$
 $d_n = \log_{\text{base}2}(n+1)$
This is a relatively small number. For example, if the complete tree T_n has $n = 1\,000\,000$ nodes, then its depth $d_n = 21$.

Extended Binary Trees: 2-Trees

A binary tree T is said to be a 2-tree or an extended binary tree if each node N has either 0 or 2 children. In such a case, the nodes with two children are called internal nodes, and the nodes with 0 children are called external nodes. Sometimes the nodes are distinguished in diagrams by using circles for internal nodes and squares for external nodes.



(a) Binary tree T .

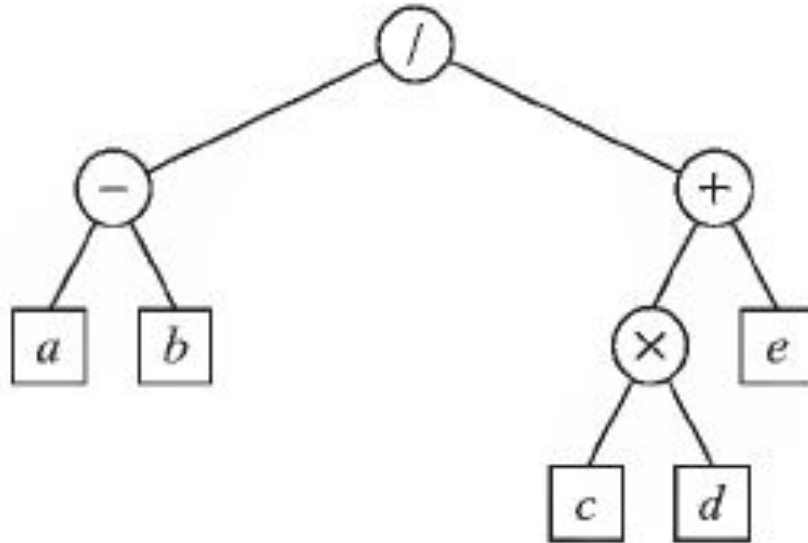


(b) **Extended 2-tree.**

Fig. 10-3 Converting a binary tree T into a 2-tree

Algebraic expression

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

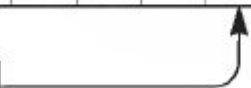


Representation Binary Tree in Memory

Linked Representation of Binary Trees:

- (1) INFO[T] contains the data at the node N.
- (2) LEFT[A"] contains the location of the left child of node N.
- (3) RIGHT[T] contains the location of the right child of node N.

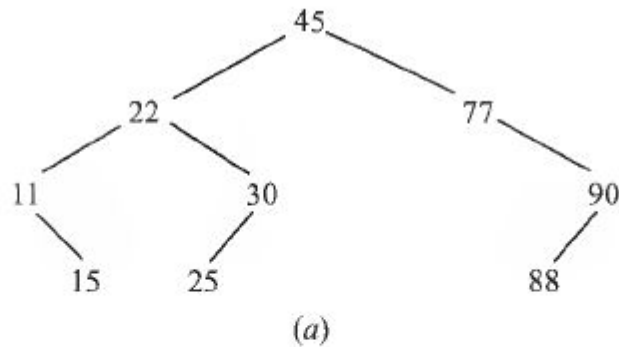
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
INFO	K	C	G		A	H	L			B		F	E			J	D	
LEFT	0	3	0		10	16	0			17		0	12			7	0	
RIGHT	0	6	0		2	1	0			13		0	0			0	0	

ROOT 5 

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

(2) If a node N occupies TREE[.ST], then its left child is stored in TREE[2*.7(f)] and its right child is stored in TREE[2*A" + 1].

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



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40
TREE	45	22	77	11	30		90		15	25				88					

TRAVERSING BINARY TREES

Preorder: (1) Process the root R.

(2) Traverse the left subtree of R in preorder.

(3) Traverse the right subtree of R in preorder.

Inorder: (1) Traverse the left subtree of R in inorder.

(2) Process the root R.

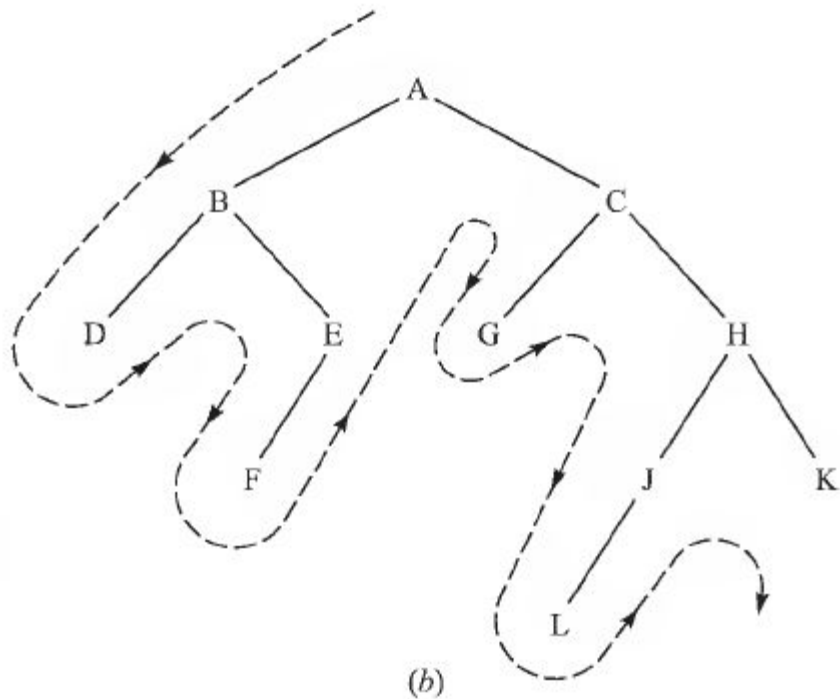
(3) Traverse the right subtree of R in inorder.

Postorder: (1) Traverse the left subtree of R in postorder.

(2) Traverse the right subtree of R in postorder.

(3) Process the root R.

Example



(Preorder) ABDEFCGHJLK

(Inorder) DBFEAGCLJHK

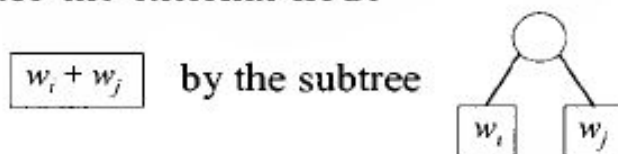
(Postorder) DFEBGLJHKHCA

Algorithm 10.5 (Huffman): The algorithm recursively finds a weighted 2-tree T with n given weights w_1, w_2, \dots, w_n which has a minimum weighted path length.

Step 1. Suppose $n = 1$. Let T be the tree with one node N with weight w_1 , and then Exit.

Step 2. Suppose $n > 1$.

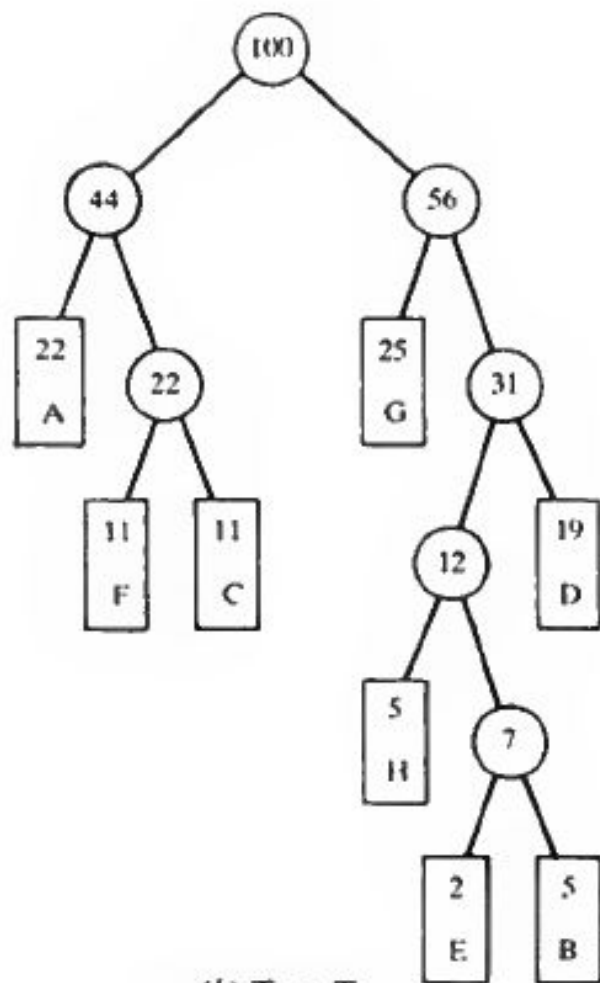
- (a) Find two minimum weights, say w_i and w_j , among the n given weights.
- (b) Replace w_i and w_j in the list by $w_i + w_j$, so the list has $n - 1$ weights.
- (c) Find a tree T' which gives a minimum weighted path length for the $n - 1$ weights
- (d) In the tree T' , replace the external node



- (e) Exit.

- (1) 22, 5, 11, 19, 2, 11, 25, 5
 (2) 22, 11, 19, (7), 11, 25, 5
 (3) 22, 11, 19, 11, 25, (12)
 (4) 22, 19, (22), 25, 12
 (5) 22, (31), 22, 25
 (6) 31, (44), 25
 (7) 44, (56)
 (8) (100)

(a) Huffman algorithm



(b) Tree T

EXAMPLE: Consider again the eight data items A, B, C, D, E, F, G, H the weights represent the percentage probabilities that the items will occur. Assigning, as above, bit labels to the edges in the Huffman tree in Fig. 10- 18(b), that is, assigned 0 or 1 according as the edge points to a left child or to a right child, we obtain the following code for the data:

A: 00, B: 11011, C : 011, D : 111,
E : 11010, F:010, G : 10, H : 1100.

For example, to get to E from the root, the path consists of a right edge, right edge, left edge, right edge, and left edge, yielding the code 11010 for E.

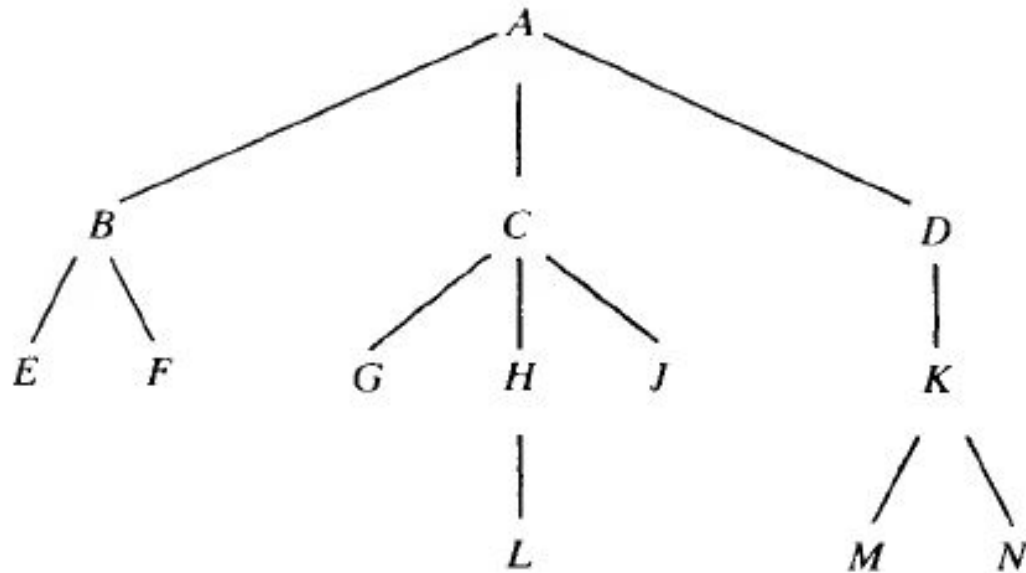
General Tree

T may be formally defined as a nonempty set of elements, called nodes, such that:

- (1) T contains a distinguished element R , called the root of T .
- (2) The remaining elements of T form an ordered collection of zero or more disjoint trees, T_1, T_2, \dots, T_n . The trees T_1, T_2, \dots, T_n are called subtrees of the root R , and the roots of T_1, T_2, \dots, T_n are called successors of R .

Terminology from family relationships, graph theory, and horticulture is used for general trees in the same way as for binary trees. In particular, if N is a node with successors S_1, S_2, \dots, S_n then N is called the parent of the S_i , the S_i are called children of N , and the S_i are called siblings of each other.

General Tree



Practice exercises of Schaums series book.