Data Structure

Tree

Instructors:
Md Nazrul Islam Mondal &
Rizoan Toufiq
Department of Computer Science & Engineering
Rajshahi University of Engineering &
Technology Rajshahi-6204

Outline

- Heap; Heapsort
- Path Lengths; Huffman's Algorithm
- General Trees

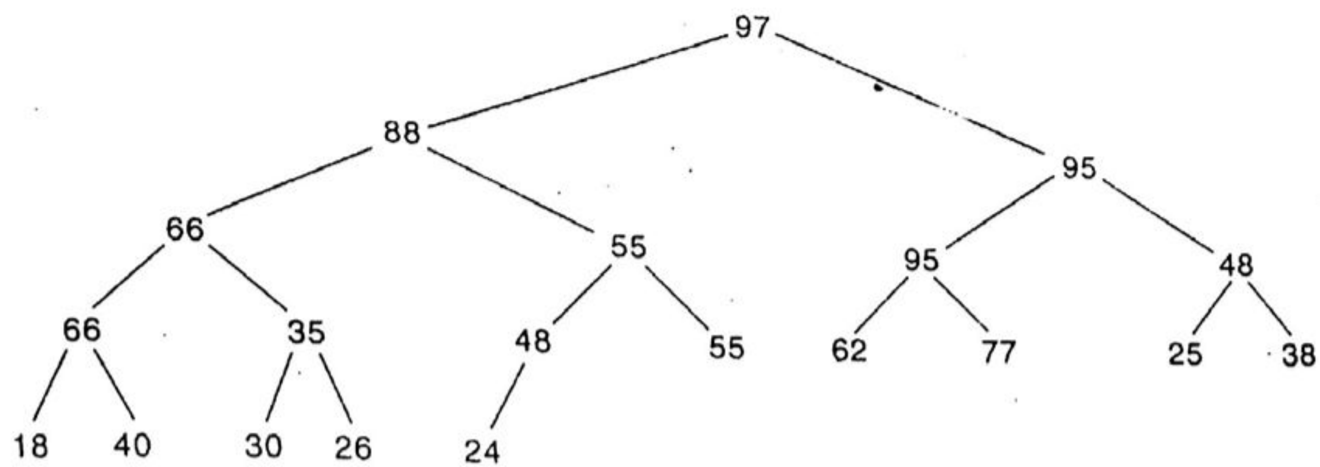
Heap; Heapsort

Heap; Heapsort

- H is a complete binary tree with n element.
- H is called heap or a maxhip, if each node N of H has the following property:
  - The value at N is *greater* than or equal to the value at each of the children of N.
  - (A minheao is defined analogously: The value at N is less than or equal to the value at each of the children of N.)

Heap; Heapsort

- Tree[1] → root,
- Tree[2k] → Left child, Tree[2k+1] → right child



(a) Heap

TREE

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
97	88	95	66	55	95	48	66	35	48	55	62	77	25	38	18	40	30	26	24

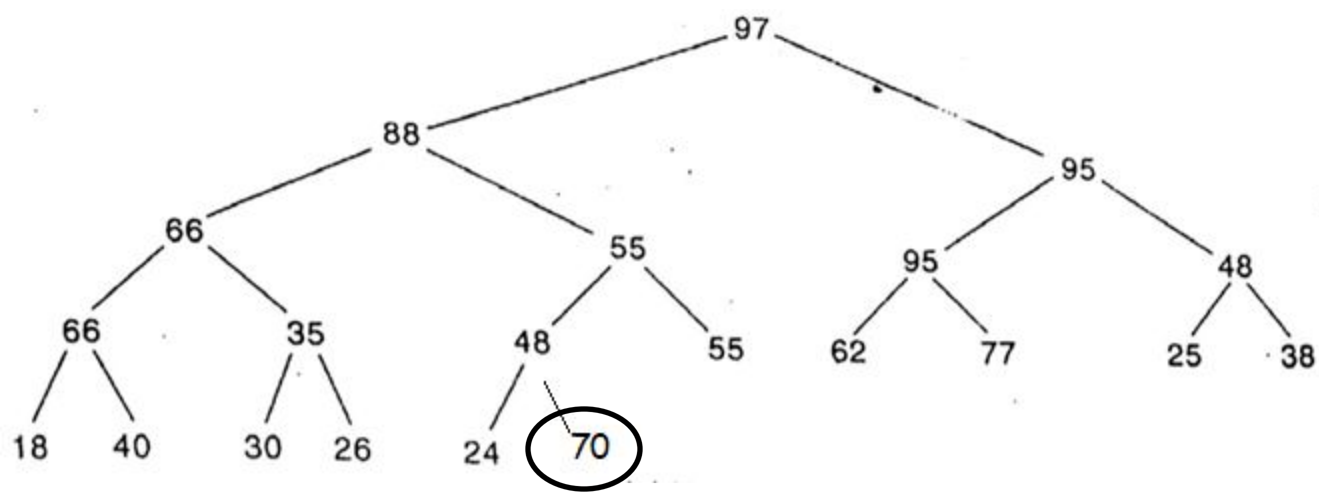
(b) Sequential representation

Heap; Heapsort (Inserting into a Heap)

- Suppose H is a heap with N elements, and suppose an ITEM of information is given. We insert ITEM into the heap H as follows:
  1) First adjoin ITEM at the end of H so that H is still a complete tree, but not necessarily a heap.
  2) Then let ITEM rise to its "appropriate place" in H so that H is finally a heap.

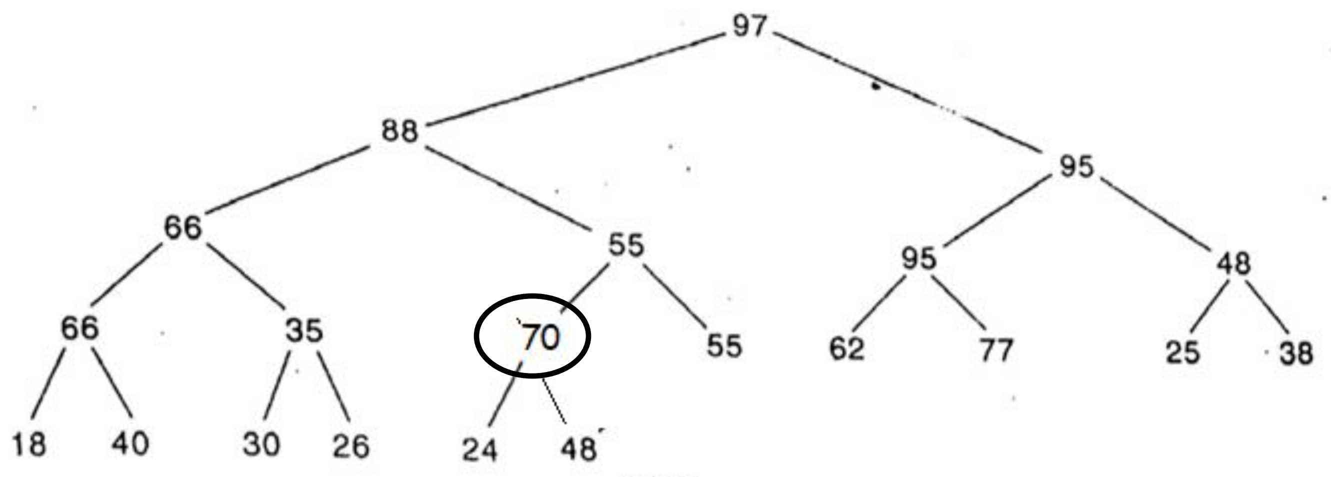
Heap; Heapsort (Inserting into a Heap)

- Insert 70



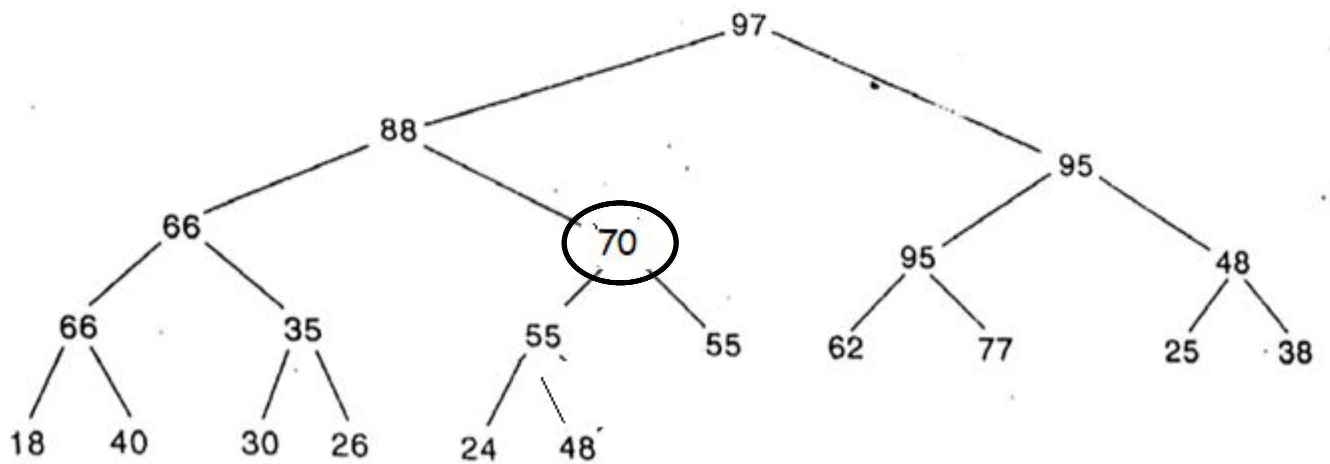
Heap; Heapsort (Inserting into a Heap)

- Insert 70



Heap; Heapsort (Inserting into a Heap)

- Insert 70



Heap; Heapsort (Inserting into a Heap)

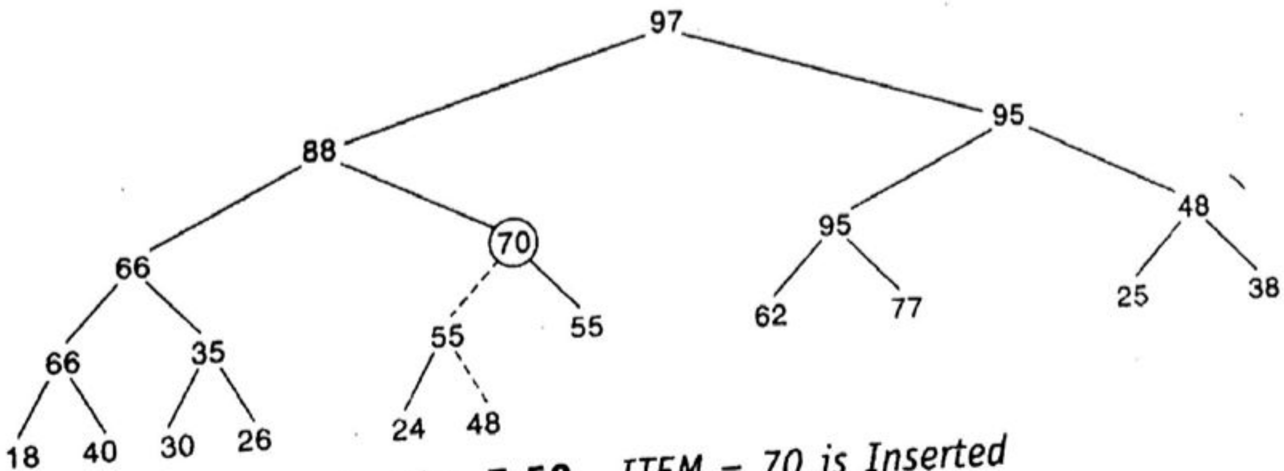
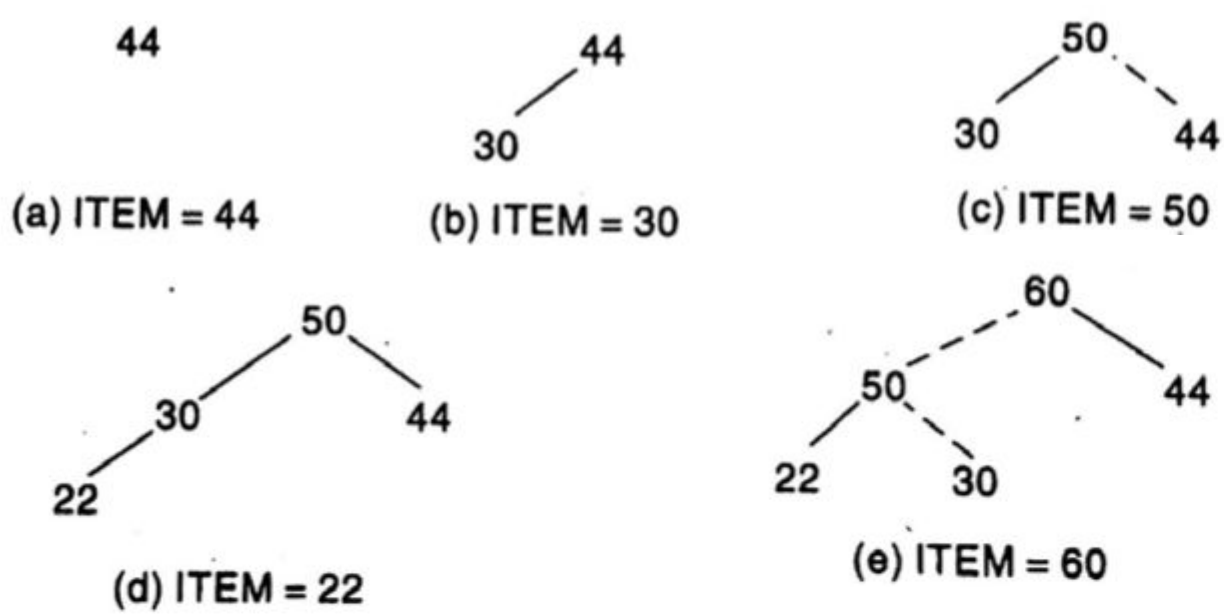


Fig. 7.59 ITEM = 70 is Inserted

Heap; Heapsort (Inserting into a Heap)

Suppose we want to build a heap H from the following list of numbers:

44, 30, 50, 22, 60, 55, 77, 55

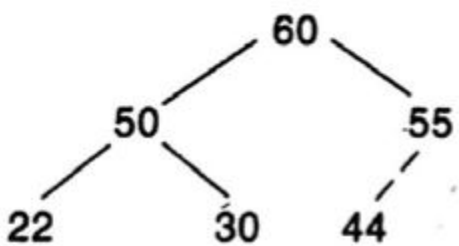


(a) ITEM = 44
(b) ITEM = 30
(c) ITEM = 50
(d) ITEM = 22
(e) ITEM = 60

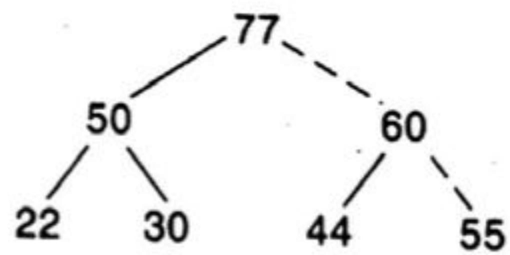
Heap; Heapsort (Inserting into a Heap)

Suppose we want to build a heap H from the following list of numbers:

44, 30, 50, 22, 60, 55, 77, 55



(f) ITEM = 55

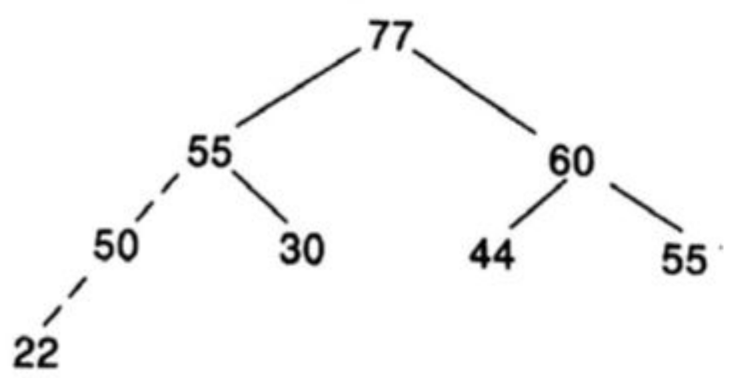


(g) ITEM = 77

Heap; Heapsort (Inserting into a Heap)

Suppose we want to build a heap H from the following list of numbers:

44, 30, 50, 22, 60, 55, 77, 55



(h) ITEM = 55

Heap; Heapsort (Inserting into a Heap)

Procedure 7.9: INSHEAP(TREE, N, ITEM)

A heap H with N elements is stored in the array TREE, and an ITEM of information is given. This procedure inserts ITEM as a new element of H. PTR gives the location of ITEM as it rises in the tree, and PAR denotes the location of the parent of ITEM.

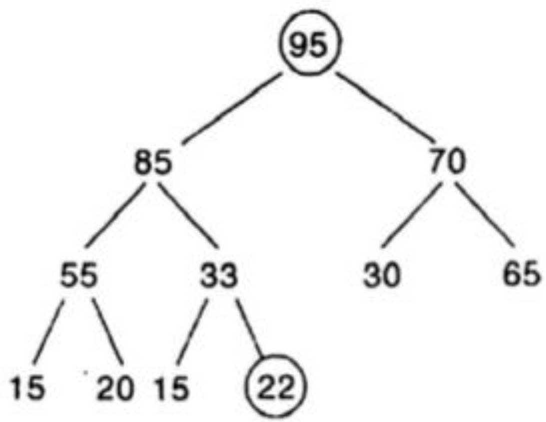
1. [Add new node to H and initialize PTR.]
   Set N := N + 1 and PTR := N.
2. [Find location to insert ITEM.]
   Repeat Steps 3 to 6 while PTR < 1.
3. Set PAR := $\lfloor PTR/2 \rfloor$. [Location of parent node.]
4. If ITEM ≤ TREE[PAR], then:
   Set TREE[PTR] := ITEM, and Return.
   [End of If structure.]
5. Set TREE[PTR] := TREE[PAR]. [Moves node down.]
6. Set PTR := PAR. [Updates PTR.]
   [End of Step 2 loop.]
7. [Assign ITEM as the root of H.]
   Set TREE[I] := ITEM.
8. Return.

Heap; Heapsort
(Deleting the Root of a Heap)

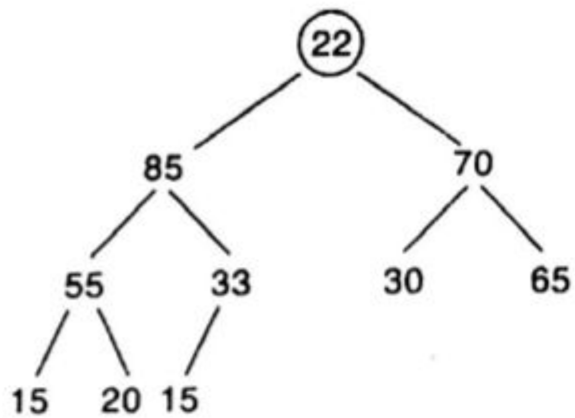
- Suppose H is a heap with N elements, and suppose we want to delete the root R of H. This is accomplished as follows
  1) Assign the R to some variable ITEM i.e. ITEM←R
  2) Replace the deleted node R by the last node L of H so that H is still a complete tree, but not necessarily a heap.
  3) (Reheap) Let L sink to its appropriate place in H so that H is a finally a heap

Heap; Heapsort (Deleting the Root of a Heap)

- Delete Root



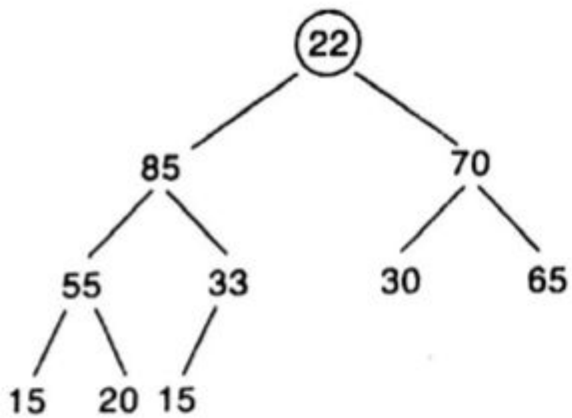
(a)



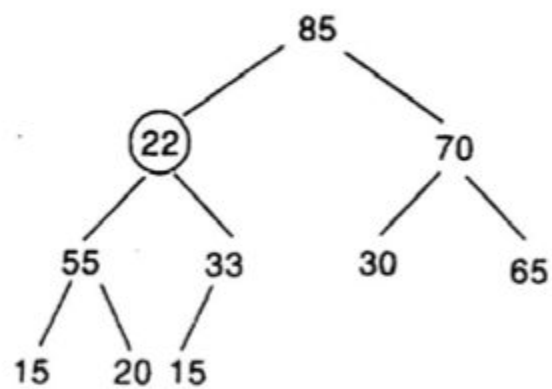
(b)

Heap; Heapsort (Deleting the Root of a Heap)

- Delete Root



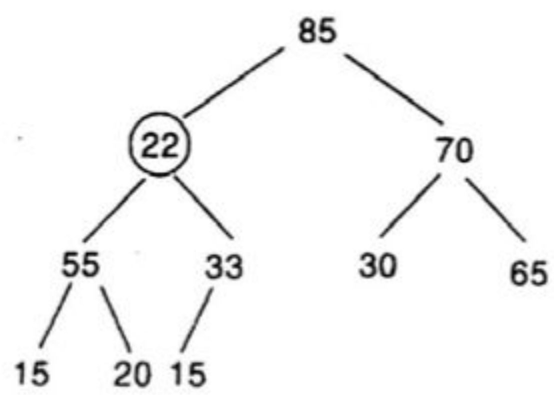
(b)



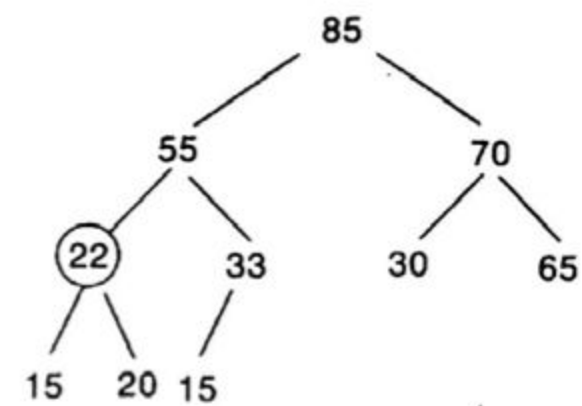
(c)

Heap; Heapsort (Deleting the Root of a Heap)

- Delete Root



(c)



(d)

Heap; Heapsort (Deleting the Root of a Heap)

Procedure 7.10: DELHEAP(TREE, N, ITEM)

A heap H with N elements is stored in the array TREE. This procedure assigns the root TREE[1] of H to the variable ITEM and then reheaps the remaining elements. The variable LAST saves the value of the original last node of H. The pointers PTR, LEFT and RIGHT give the locations of LAST and its left and right children as LAST sinks in the tree.

1. Set ITEM := TREE[1]. [Removes root of H.]
2. Set LAST := TREE[N] and N := N − 1. [Removes last node of H.]
3. Set PTR := 1, LEFT := 2 and RIGHT := 3. [Initializes pointers.]
4. Repeat Steps 5 to 7 while RIGHT ≤ N:
5. If LAST ≥ TREE[LEFT] and LAST ≥ TREE[RIGHT], then:
   Set TREE[PTR] := LAST and Return.
   [End of If structure.]
6. IF TREE[RIGHT] ≤ TREE[LEFT], then:
   Set TREE[PTR] := TREE[LEFT] and PTR := LEFT.
   Else:
   Set TREE[PTR] := TREE[RIGHT] and PTR := RIGHT.
   [End of If structure.]
7. Set LEFT := 2*PTR and RIGHT := LEFT + 1.
   [End of Step 4 loop.]
8. If LEFT = N and if LAST < TREE[LEFT], then: Set PTR := LEFT.
9. Set TREE[PTR] := LAST.
10. Return.

Heap: Heapsort (Heapsort)

- Two Phase
  A. Build a heap H out of the elements of A
  B. Repeatedly delete the root element of H

Algorithm 7.11: HEAPSORT(A, N)
An array A with N elements is given. This algorithm sorts the elements of A.

1. [Build a heap H, using Procedure 7.9.]
   Repeat for J = 1 to N – 1:
   Call INSHEAP(A, J, A[J + 1]).
   [End of loop.]
2. [Sort A by repeatedly deleting the root of H, using Procedure 7.10.]
   Repeat while N > 1:
   (a) Call DELHEAP(A, N, ITEM).
   (b) Set A[N + 1] := ITEM.
   [End of Loop.]
3. Exit.

- **Remember: A[N+1] does not belong to the heap H, it is safe to save the root**

Heap; Heapsort (Heapsort)

- Phase A: $f_1(n) = O(n\log n)$
- Phase B: $f_2(n) = O(n\log n)$
- Heapsort Algorithm: $O(n\log n)$ [Best, Average and Worst]

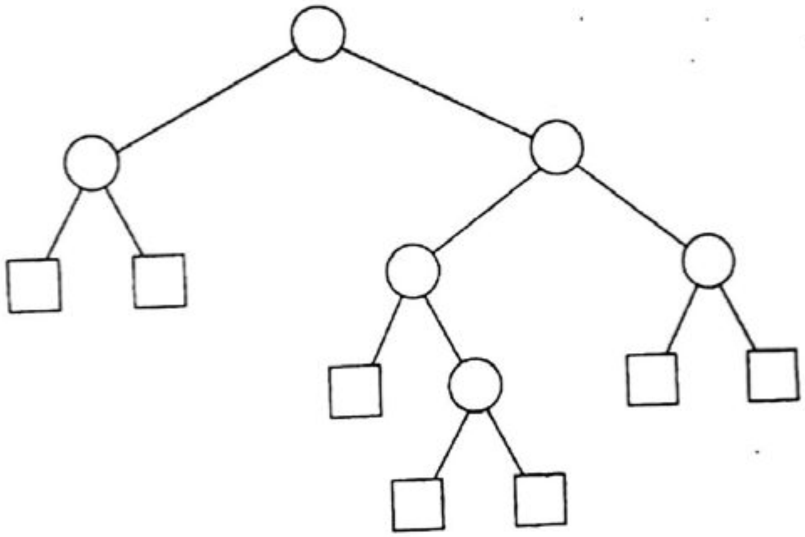
- But best and Average case complexity of QuickSort: $O(n\log n)$
- The worst Case Complexity of QuickSort: $O(n^2)$

- The best, average and worst Case Complexity of bubble sort: $O(n^2)$

Path Lengths; Huffman's Algorithm

Path Lengths; Huffman's Algorithm

- Extended Binary Tree or 2-tree
  - External Node (N_E) – the node with 0 children, denoted by square
  - Internal Node(N_I) – the node with 2 children, denoted by circle
  - **Equation**: $N_E = N_I + 1$



Path Lengths; Huffman's Algorithm

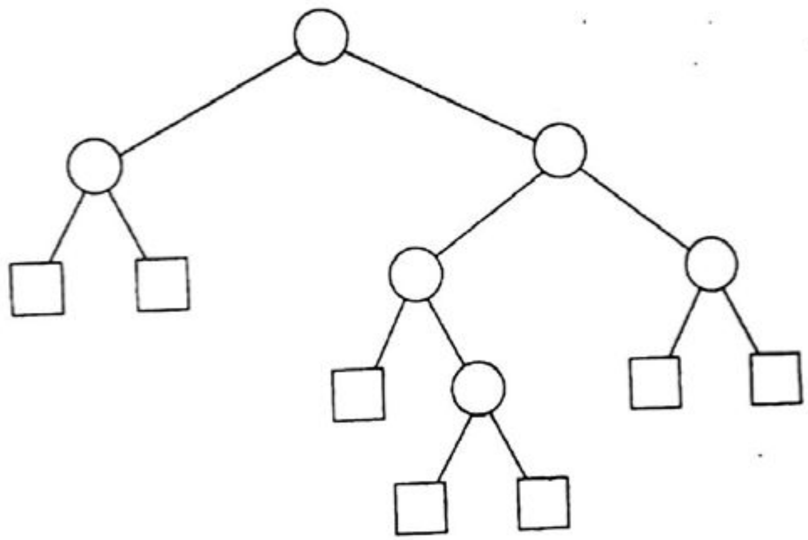
- External Path Length (L_E):
  - The sum of all path lengths summed over each path from the root R of T to external node.

- Internal Path Length (L_I):
  - The sum of all path lengths summed over each path from the root R of T to internal node.

Path Lengths; Huffman's Algorithm

- $L_E = 2+2+3+4+4+3+3 = 21$
- $L_I = 0+1+1+2+3+2 = 9$

- $L_E = L_I + 2n$, n number of internal node



Path Lengths; Huffman's Algorithm

- Suppose T is a 3-tree with n **external** nodes,
- Suppose each of the external nodes is assigned a **nonnegative weight**.
- The **external weighted path length P** of the tree is defined to be the sum of the weighted path length i.e.

$$P = W_1L_1 + W_2L_2 + \dots + W_nL_n$$

W_i the weight of an external Node i

L_i the path length of an external Node i

Path Lengths; Huffman's Algorithm

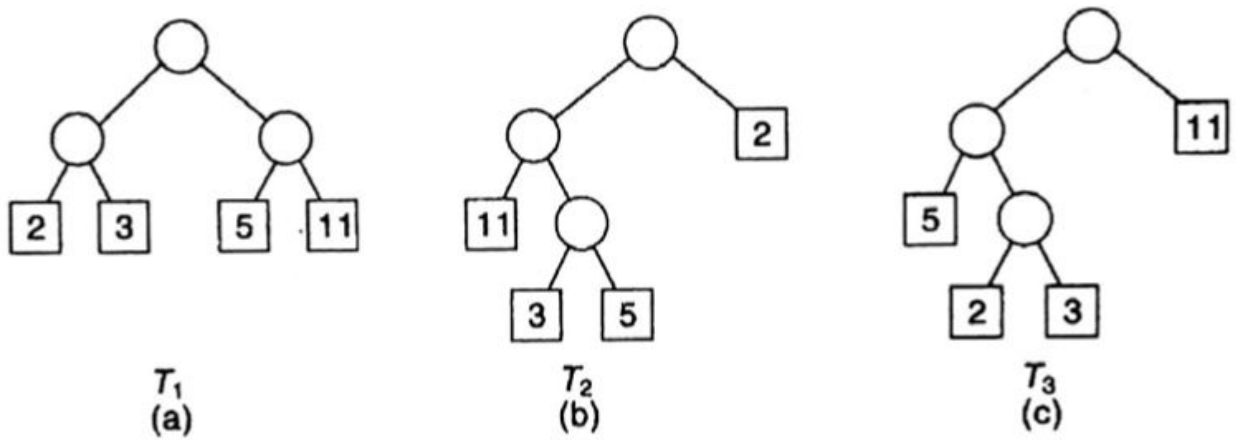


Fig. 7.63

$$P_1 = 2 \cdot 2 + 3 \cdot 2 + 5 \cdot 2 + 11 \cdot 2 = 42$$
$$P_2 = 2 \cdot 1 + 3 \cdot 3 + 5 \cdot 3 + 11 \cdot 2 = 48$$
$$P_3 = 2 \cdot 3 + 3 \cdot 3 + 5 \cdot 2 + 11 \cdot 1 = 36$$

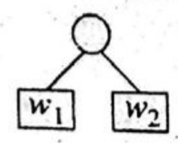
Path Lengths; Huffman's Algorithm

- General Problem:
  - Suppose a list **of n weights** is given: w_1 , w_2 , … , w_n . Among all the 2 trees with **n external nodes** with the given **n weight**, find a tree T with **a minimum-weighted path length**.

- Solution: Huffman's Algorithm

Path Lengths; Huffman's Algorithm

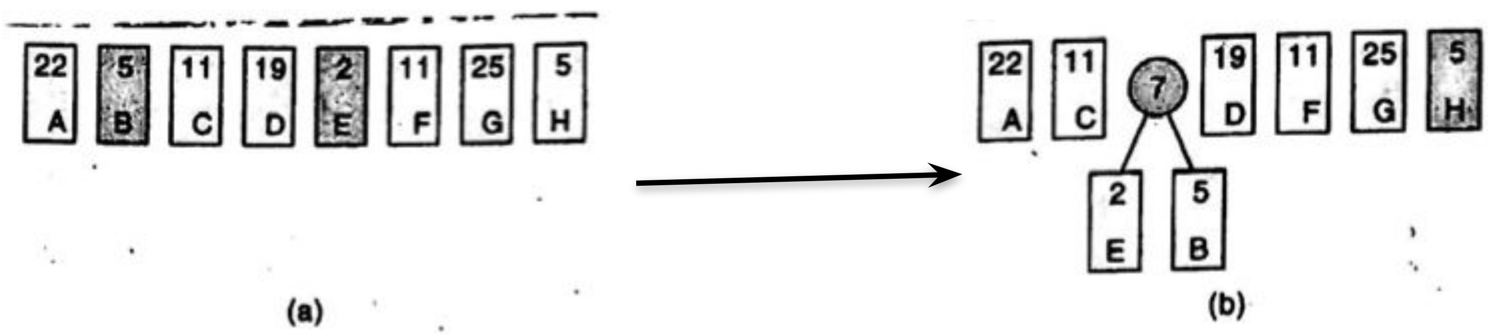
- Huffman's Algorithm
  - Suppose w_1 and w_2 are two minimum weights among the n given weights w_1 , w_2 , … , w_n.
  - Find a tree T' which gives a solution for the n-1 weights: $w_1 + w_2$, w_3 … w_n
  - Then in the tree T', replace the external node $w_1 + w_2$ by the subtree



  - The new 2-tree T is the desired solution

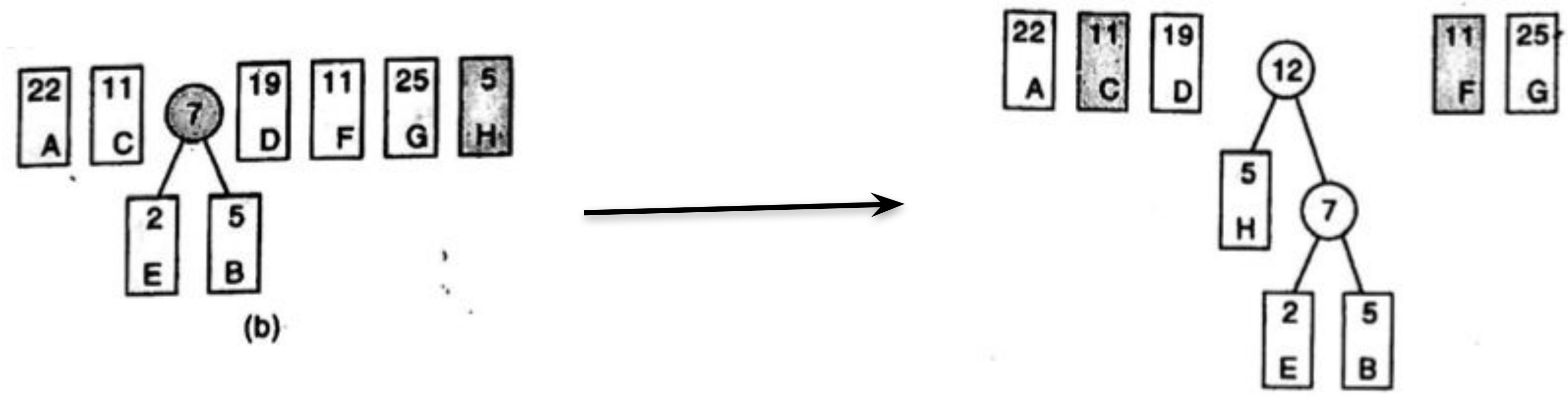
Path Lengths; Huffman's Algorithm

- Huffman's Algorithm



Path Lengths; Huffman's Algorithm

- Huffman's Algorithm

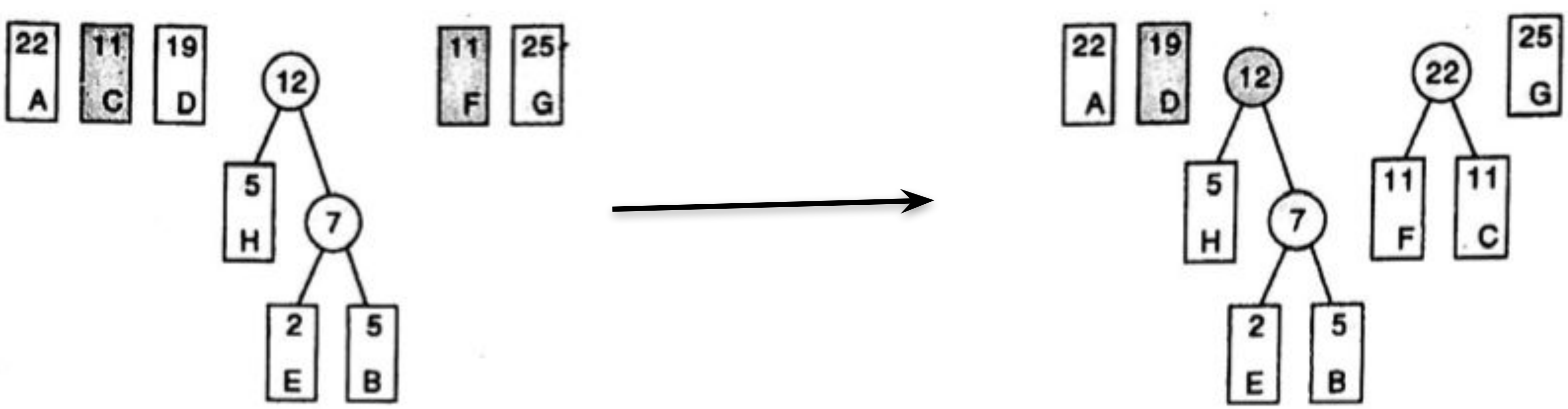


(b)

(c)

Path Lengths; Huffman's Algorithm

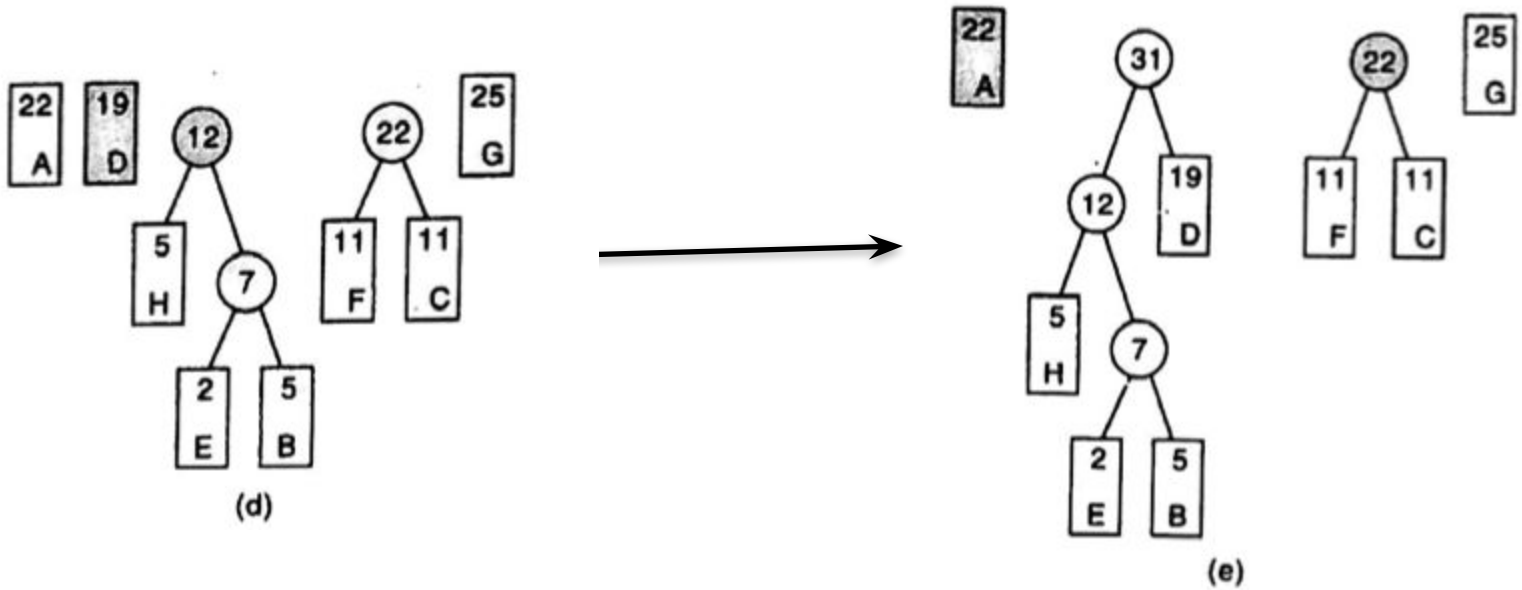
- Huffman's Algorithm



(c) (d)

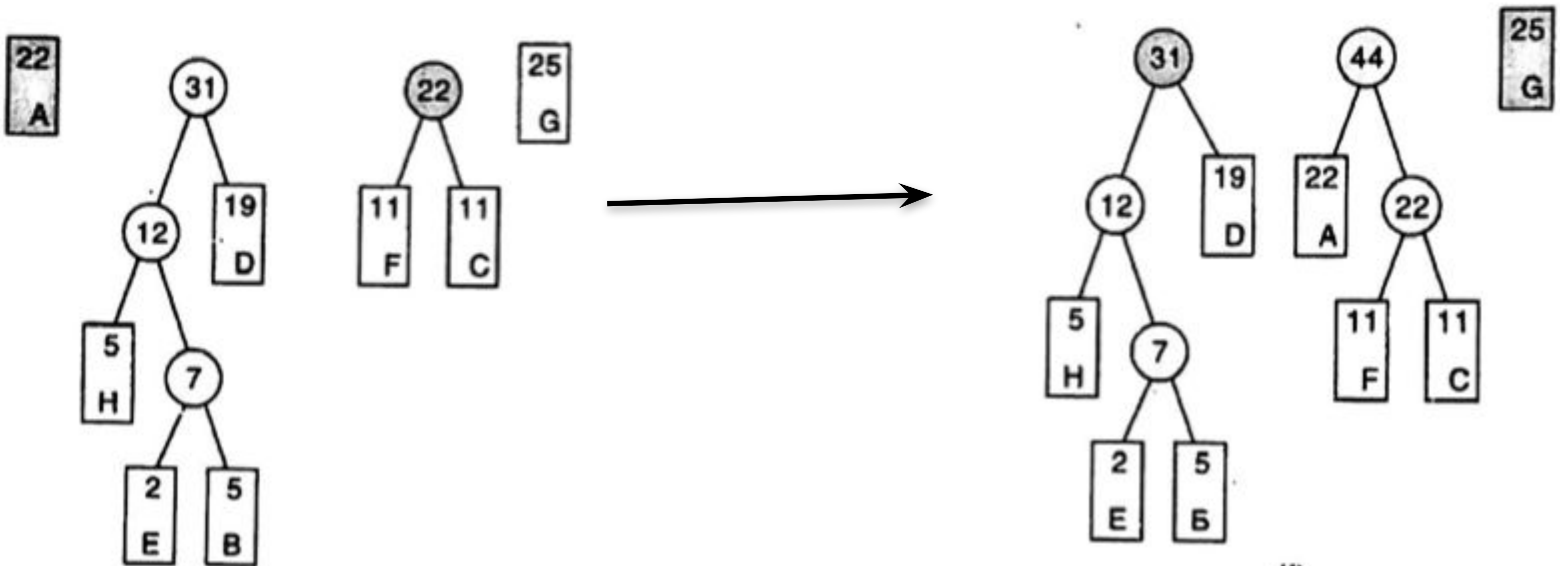
Path Lengths; Huffman's Algorithm

- Huffman's Algorithm



Path Lengths; Huffman's Algorithm

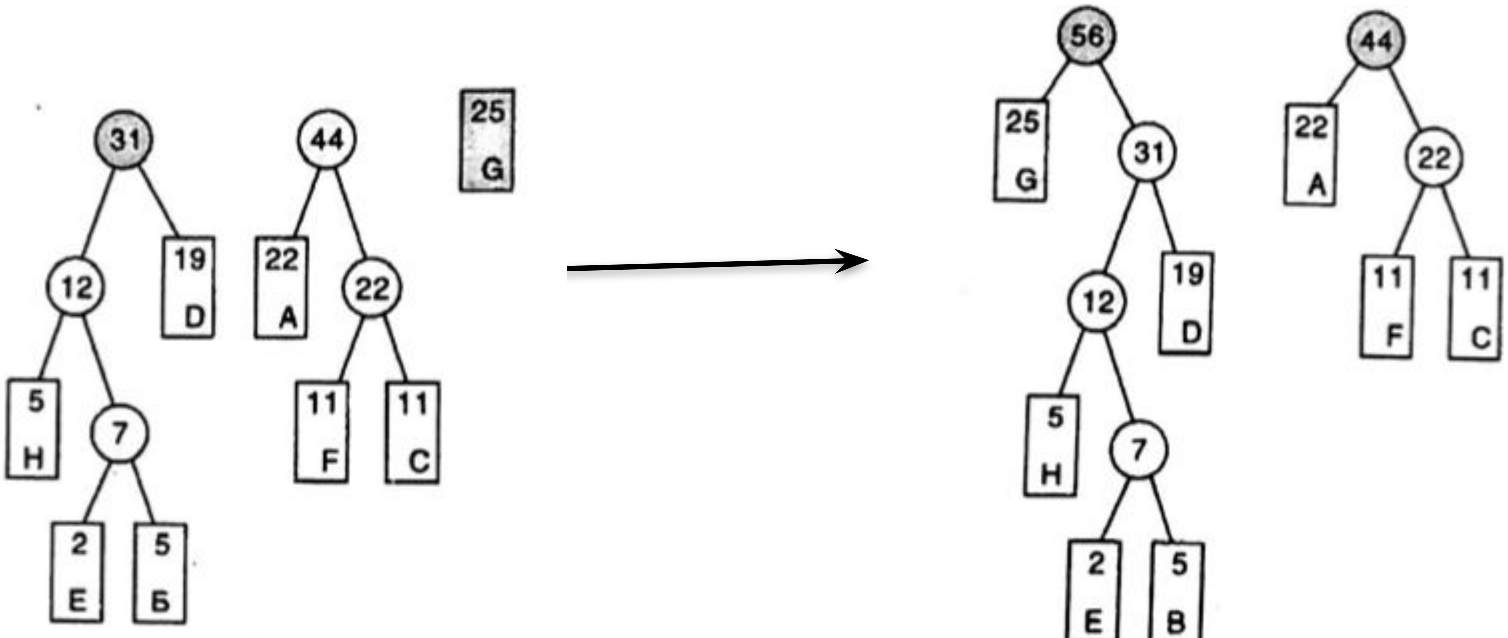
- Huffman's Algorithm



(e) (f)

Path Lengths; Huffman's Algorithm

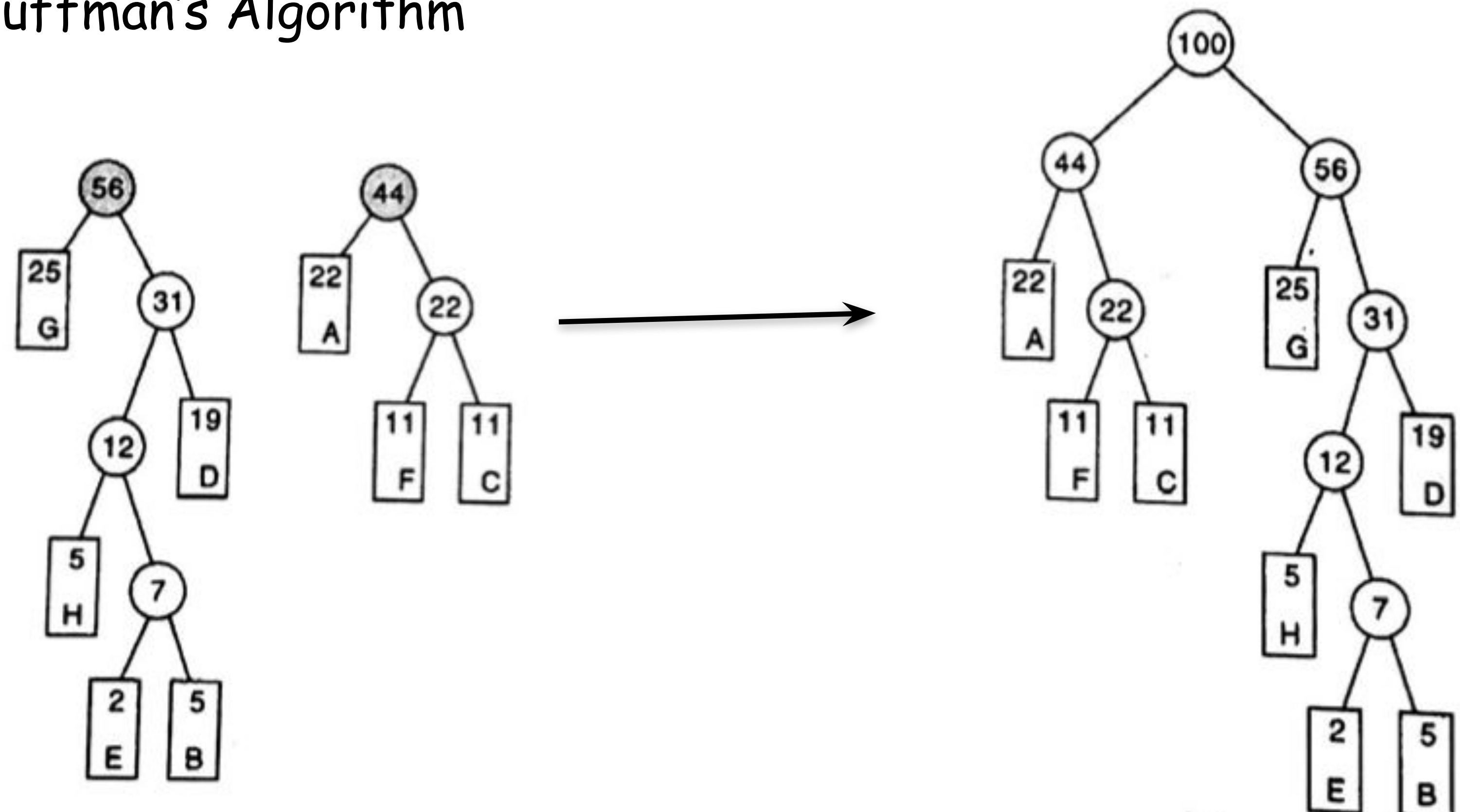
- Huffman's Algorithm



(f)

Path Lengths; Huffman's Algorithm

- Huffman's Algorithm



(h)

Path Lengths; Huffman's Algorithm

- Application to Coding
  - Data item don't occur with the same probability.
  - Then memory space may be **conserved** by using **variable-length string**
  - Items which occur frequently are assigned shorter string
  - Item which occur infrequently are assigned longer string.

Path Lengths; Huffman's Algorithm

- Application to Coding
  - Z, U and V are used frequently
  - X and Y are used infrequently

U: 00 V: 01 W: 100 X: 1010 Y: 1011 Z: 11

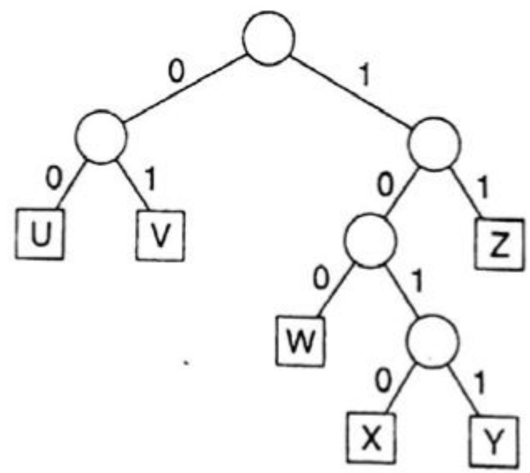


Fig. 7.66

General Trees

General Trees

- A nonempty finite set T 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 tree $T_1, T_2, \ldots, T_m$
- The trees $T_1, T_2, \ldots, T_m$ are called subtrees of the root R
- The root of $T_1, T_2, \ldots, T_m$ are called successors of R.

General Trees

- If N is a node with successors S_1, S_2, …, S_m , then N is called the parent of the S_i ‘s.
- S_i ‘s are called children of N
- S_i ‘s are called siblings of each other.

Figure 7.68 pictures a general tree *T* with 13 nodes,

A, B, C, D, E, F, G, H, J, K, L, M, N

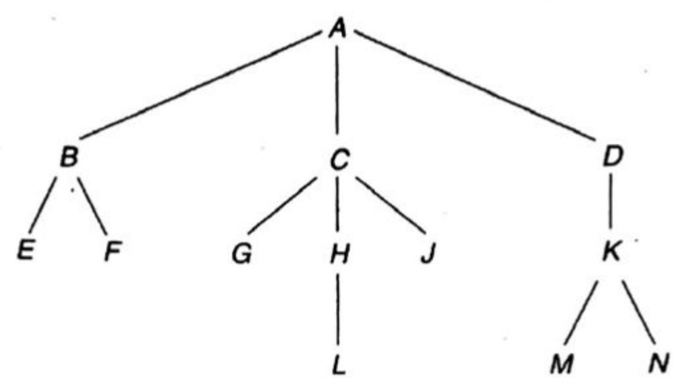


Fig. 7.68

General Trees

- Difference between general tree and binary tree
    1) A binary tree T' may be empty but a general tree T is nonempty
    2) Suppose a node N has only one child. Then the child is distinguished as a left child or right child in a binary tree T', but no such distinction exists in a general tree T.

General Trees

- Two Properties of a Tree
  - T has a distinguished node R called the root of T
  - T is ordered that is the children of each node N of T have a specific order.

- A forest F is defined to be an ordered collection of zero or more distinct tree.
  - If we delete the root R from a general tree T, then we obtain the forest F consisting of the subtrees of R (which may be empty)
  - Conversely If F is a forest, then we may adjoin a node R to F to form a general tree T where R is the root of T.

General Trees (Computer Representation)

- Suppose T is a general Tree.
- Uses three parallel arrays INFO, CHILD (or DOWN) and SIBL (or HORZ), and a pointer variable ROOT as follows. First of all, each node N of T will correspond to a location K such that
    1) INFO[K] contains the data at node N.
    2) CHILD[K] contains the location of the first child of N. The condition CHILD[K} = NULL indicates that N has no children.
    3) SIBL[K] contains the location of the next sibling of N. The condition SIBL[K] = NULL indicates that N is the last child of its parents.

General Trees (Computer Representation)

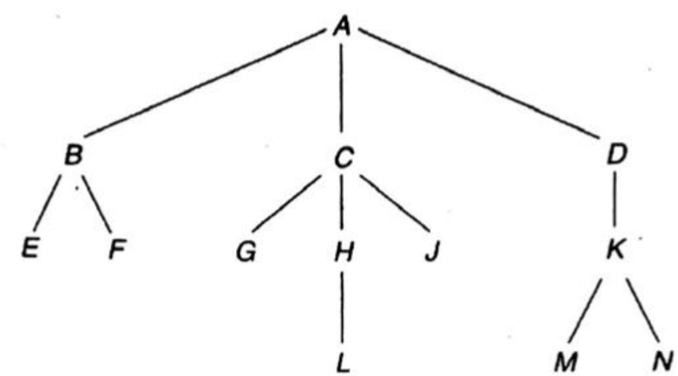
	INFO
1	
2	A
3	B
4	C
5	
6	G
7	H
8	J
9	N
10	M
11	L
12	K
13	
14	F
15	E
16	D

(a)

	CHILD	SIBL
1	5	
2	3	0
3	15	4
4	6	16
5	13	
6	0	7
7	11	8
8	0	0
9	0	0
10	0	9
11	0	0
12	10	0
13	0	
14	0	0
15	0	14
16	12	0

ROOT = 2, AVAIL = 13

(b)



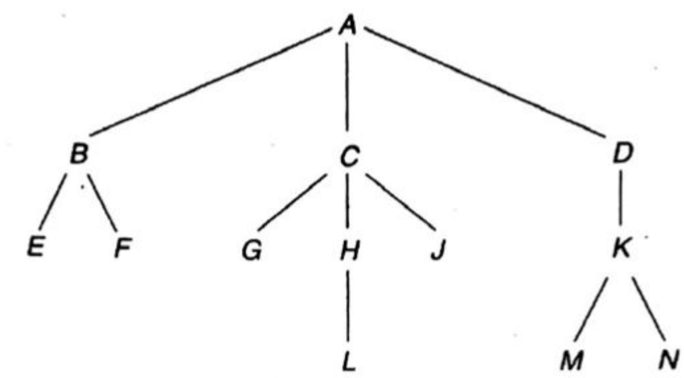
General Trees

(Correspondence between general tree and binary tree)

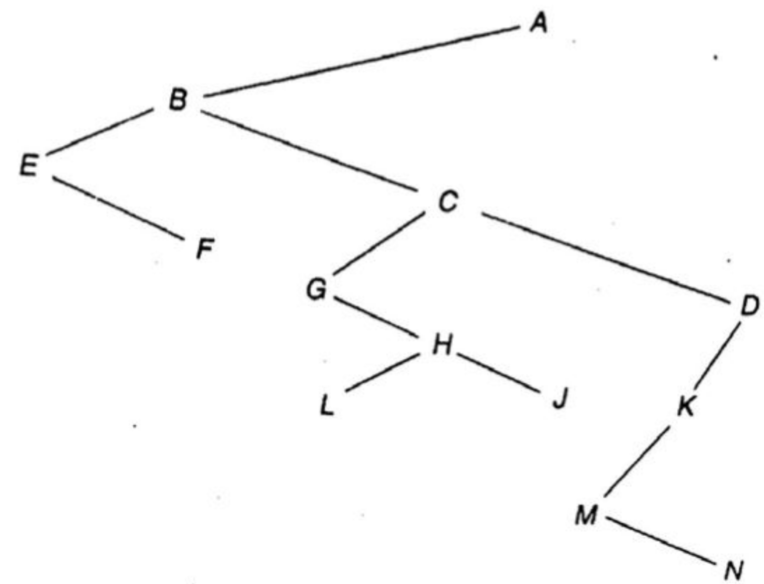
- Suppose T is a general tree.
- Then we may assign a unique binary tree T' to T as follows.
  - First of all, the node of binary tree T' will be the same as the nodes of the general tree T and the root of T' will be the root of T.
  - Let N be an arbitrary node of the binary tree T'
    - Then the left child of N in T' will be the first child of the node N in the general tree T.
    - The right child of N in T' will be the next sibling of N in the general tree T.

General Trees

(Correspondence between general tree and binary tree)



General Tree T



Binary tree T'

Any Query?