Chapter 1

Introduction and Overview

1.1 INTRODUCTION

This chapter introduces the subject of data structures and presents an overview of the content of

Since (d) and (f) may contain a few or many items, they may lead to variable-length records. Also, (e) may contain many items, unless it asks only for the highest level obtained.

1.4 Data base systems will be only briefly covered in this text. Why?

"Data base systems" refers to data stored in the secondary memory of the computer. The implementation and analysis of data structures in the secondary memory are very different from those in the main memory of the computer. This text is primarily concerned with data structures in main memory, not secondary memory.

Data Structures and Operations

1.5 Give a brief description of (a) traversing, (b) sorting and (c) searching.

(a) Accessing and processing each record exactly once
(b) Arranging the data in some given order
(c) Finding the location of the record with a given key or keys

1.6 Give a brief description of (a) inserting and (b) deleting.

(a) Adding a new record to the data structure, usually keeping a particular ordering
(b) Removing a particular record from the data structure

1.7 Consider the linear array NAME in Fig. 1.15, which is sorted alphabetically.
(a) Find NAME[2], NAME[4] and NAME[7].
(b) Suppose Davis is to be inserted into the array. How many names must be moved to new locations.
(c) Suppose Gupta is to be deleted from the array. How many names must be moved to new locations?

	NAME
1	Adam
2	Clark
3	Evans
4	Gupta
5	Jones
6	Lane
7	Pace
8	Smith

Fig. 1.15

(a) Here NAME[K] is the *k*th name in the list. Hence,

NAME[2] = Clark, NAME[4] = Gupta, NAME[7] = Pace

(b) Since Davis will be assigned to NAME[3], the names Evans through Smith must be moved. Hence six names are moved.
(c) The names Jones through Smith must be moved up the array. Hence four names must be moved.

1.8 Consider the linear array NAME in Fig. 1.16. The values of FIRST and LINK[K] in the figure determine a linear ordering of the names as follows. FIRST gives the location of the first name in the list, and LINK[K] gives the location of the name following NAME[K], with 0 denoting the end of the list. Find the linear ordering of the names.

	CITY
1	Atlanta
2	Boston
3	Chicago
4	Miami
5	Philadelphia

(a)

	NUMBER	ORIG	DEST
1	701	2	3
2	702	3	2
3	705	5	3
4	708	3	4
5	711	2	5
6	712	5	2
7	713	5	1
8	715	1	4
9	717	5	4
10	718	4	5

(b)

Fig. 1.19

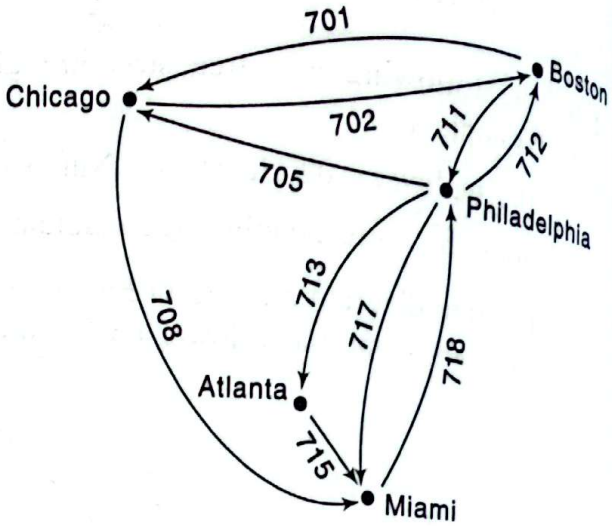


Fig. 1.20

Complexity; Space-Time Tradeoffs

1.13 Briefly describe the notions of (a) the complexity of an algorithm and (b) the space-time tradeoff of algorithms.

(a) The complexity of an algorithm is a function $f(n)$ which measures the time and/or space used by an algorithm in terms of the input size n.

(b) The space-time tradeoff refers to a choice between algorithmic solutions of a data processing problem that allows one to decrease the running time of an algorithmic solution by increasing the space to store the data and vice versa.

1.14 Suppose a data set S contains n elements.

(a) Compare the running time T_1 of the linear search algorithm with the running time T_2 of the binary search algorithm when (i) $n = 1000$ and (ii) $n = 10\,000$.

(b) Discuss searching for a given item in S when S is stored as a linked list.

(a) Recall (Sec. 1.5) that the expected running of the linear search algorithm is $f(n) = n/2$ and that the binary search algorithm is $f(n) = \log_2 n$. Accordingly, (i) for $n = 1000$, $T_1 = 500$ but $T_2 = \log_2 1000 \approx 10$; and (ii) for $n = 10\,000$, $T_1 = 5000$ but $T_2 = \log_2 10\,000 \approx 14$.

(b) The binary search algorithm assumes that one can directly access the middle element in the set S. But one cannot directly access the middle element in a linked list. Hence one may have to use a linear search algorithm when S is stored as a linked list.

1.15 Consider the data in Fig. 1.19, which gives the different flights of an airline. Discuss different ways of storing the data so as to decrease the time in executing the following:

(a) Find the origin and destination of a flight, given the flight number.

(b) Given city A and city B, find whether there is a flight from A to B, and if there is, find its flight number.

(a) Store the data of Fig. 1.19(b) in arrays ORIG and DEST where the subscript is the flight number, as pictured in Fig. 1.21(a).

(b) Store the data of Fig. 1.19(b) in a two-dimensional array FLIGHT where FLIGHT[J, K] contains the flight number of the flight from CITY[J] to CITY[K], or contains 0 when there is no such flight, as pictured in Fig. 1.21(b).

	ORIG	DEST
701	2	3
702	3	2
703	0	0
704	0	0
705	5	3
706	0	0
⋮	⋮	⋮
715	1	4
716	0	0
717	5	4
718	4	5

(a)

FLIGHT	1	2	3	4	5
1	0	0	0	715	0
2	0	0	701	0	711
3	0	702	0	708	0
4	0	0	0	0	718
5	713	712	705	717	0

(b)

Fig. 1.21

1.16 Suppose an airline serves n cities with s flights. Discuss drawbacks to the data representations used in Fig. 1.21(a) and Fig. 1.21(b).

(a) Suppose the flight numbers are spaced very far apart; i.e. suppose the ratio of the number s of flights to the number of memory locations is very small, e.g. approximately 0.05. Then the extra storage space may not be worth the expense.

(b) Suppose the ratio of the number s of flights to the number n of memory locations in the array FLIGHT is very small, i.e. that the array FLIGHT is one that contains a large number of zeros (such an array is called a sparse matrix). Then the extra storage space may not be worth the expense.

1.17 List examples of linear data structures.

Arrays, linked lists, stacks and queues are examples of linear data structures.

1.18 Define Abstract Data Type. Explain it briefly.

An abstract data type can be defined as a data declaration packaged together with the operations that are meaningful for the data type. In other words, we encapsulate the data and the operations on the data, and then we hide them from the user.

Chapter 4

Arrays, Records and Pointers

[3]	5	0	-15
[4]	1	1	11
[5]	2	1	3
[6]	3	2	-6
[7]	0	4	91
[8]	2	5	28

SOLVED PROBLEMS

Linear Arrays

4.1 Consider the linear arrays AAA(5:50), BBB(–5:10) and CCC(18).

(a) Find the number of elements in each array.

(b) Suppose *Base*(AAA) = 300 and w = 4 words per memory cell for AAA. Find the address of AAA[15], AAA[35] and AAA[55].

(a) The number of elements is equal to the length; hence use the formula

$$\text{Length} = \text{UB} - \text{LB} + 1$$

Accordingly,

$$\text{Length(AAA)} = 50 - 5 + 1 = 46$$
$$\text{Length(BBB)} = 10 - (-5) + 1 = 16$$
$$\text{Length(CCC)} = 18 - 1 + 1 = 18$$

Note that Length(CCC) = UB, since LB = 1.

(b) Use the formula

$$\text{LOC(AAA[K])} = \textit{Base}(\text{AAA}) + w(\text{K} - \text{LB})$$

Hence:

$$\text{LOC(AAA[15])} = 300 + 4(15 - 5) = 340$$
$$\text{LOC(AAA[35])} = 300 + 4(35 - 5) = 420$$

AAA[55] is not an element of AAA, since 55 exceeds UB = 50.

4.2 Suppose a company keeps a linear array YEAR(1920:1970) such that YEAR[K] contains the number of employees born in year K. Write a module for each of the following tasks:

(a) To print each of the years in which no employee was born.

(b) To find the number NNN of years in which no employee was born.

(c) To find the number N50 of employees who will be at least 50 years old at the end of the year. (Assume 1984 is the current year.)

(d) To find the number NL of employees who will be at least L years old at the end of the year. (Assume 1984 is the current year.)

Each module traverses the array.

(a) 1. Repeat for K = 1920 to 1970:
If YEAR[K] = 0, then: Write: K.
[End of loop.]
2. Return.

(b) 1. Set NNN := 0.
2. Repeat for K = 1920 to 1970:
If YEAR[K] = 0, then: Set NNN := NNN + 1.
[End of loop.]
3. Return.

(c) We want the number of employees born in 1934 or earlier.
1. Set N50 := 0.
2. Repeat for K = 1920 to 1934:
Set N50 := N50 + YEAR[K].
[End of loop.]
3. Return.

(d) We want the number of employees born in year 1984 – L or earlier.
1. Set NL := 0 and LLL := 1984 – L
2. Repeat for K = 1920 to LLL:
Set NL := NL + YEAR[K].
[End of loop.]
3. Return.

4.3 Suppose a 10-element array A contains the values $a_1, a_2, \ldots, a_{10}$. Find the values in A after each loop.

(a) Repeat for K = 1 to 9:
Set A[K + 1] := A[K].
[End of loop.]

(b) Repeat for K = 9 to 1 by –1:
Set A[K + 1] := A[9].
[End of loop.]

Note that the index K runs from 1 to 9 in part (a) but in reverse order from 9 back to 1 in part (b).

(a) First A[2] := A[1] sets A[2] = a_1, the value of A[1].
Then A[3] := A[2] sets A[3] = a_1, the current value of A[2].
Then A[4] := A[3] sets A[4] = a_1, the current value of A[3]. And so on.
Thus every element of A will have the value x_1, the original value of A[1].

(b) First A[10] : = A[9] sets A[10] = a_9.
Then A[9]:= A[8] sets A[9] = a_8.
Then A[8]: = A[7] sets A[8] = a_7. And so on.
Thus every value in A will move to the next location. At the end of the loop, we still have A[1] = x_1.

Remark: This example illustrates the reason that, in the insertion algorithm, Algorithm 4.4, the elements are moved downward in reverse order, as in loop (b) above.

(a) C(Hobbs) = 6, since Hobbs is compared with each name, beginning with Allen, until Hobbs is found in NAME[6].
C(Morgan) = 10, since Morgan appears in NAME[10].
C(Fisher) = 15, since Fisher is initially placed in NAME[15] and then Fisher is compared with every name until it is found in NAME[15]. Hence the search is unsuccessful.

(b) Observe that NAME is alphabetized. Accordingly, the linear search can stop after a given name XXX is compared with a name YYY such that XXX < YYY (i.e., such that, alphabetically, XXX comes before YYY). With this algorithm, C(Fisher) = 5, since the search can stop after Fisher is compared with Goodman in NAME[5].

4.6 Suppose the binary search algorithm, Algorithm 4.6, is applied to the array NAME in Fig. 4.30 to find the location of Goodman. Find the ends BEG and END and the middle MID for the test segment in each step of the algorithm.

Recall that MID = INT((BEG + END)/2), where INT means integer value.
Step 1. Here BEG = 1 [Allen] and END = 14 [Walton), so MID = 7 [Irwin].
Step 2. Since Goodman < Irwin, reset END = 6. Hence MID = 3 [Dickens].
Step 3. Since Goodman > Dickens, reset BEG = 4. Hence MID = 5 [Goodman].
We have found the location LOC = 5 of Goodman in the array. Observe that, there were $C = 3$ comparisons.

4.7 Modify the binary search algorithm, Algorithm 4.6, so that it becomes a search and insertion algorithm.

There is no change in the first four steps of the algorithm. The algorithm transfers control to Step 5 only when ITEM does not appear in DATA. In such a case, ITEM is inserted before or after DATA[MID] according to whether ITEM < DATA[MID] or ITEM > DATA[MID]. The algorithm follows.

Algorithm P4.7: (Binary Search and Insertion) DATA is a sorted array with N elements, and ITEM is a given item of information. This algorithm finds the location LOC of ITEM in DATA or inserts ITEM in its proper place in DATA.

Steps 1 through 4. Same as in Algorithm 4.6.
5. If ITEM < DATA[MID), then:
   Set LOC := MID.
   Else:
   Set LOC := MID + 1.
   [End of If structure.]
6. Insert ITEM into DATA[LOC] using Algorithm 4.2.
7. Exit.

4.8 Suppose A is a sorted array with 200 elements, and suppose a given element x appears with the same probability in any place in A. Find the worst-case running time $f(n)$ and the average-case running time $g(n)$ to find x in A using the binary search algorithm.

For any value of k, let n_k denote the number of those elements in A that will require k comparisons to be located in A. Then:

k:	1	2	3	4	5	6	7	8
n_k:	1	2	4	8	16	32	64	73

The 73 comes from the fact that $1 + 2 + 4 + \ldots + 64 = 127$ so there are only $200 - 127 = 73$ elements left. The worst-case running time $f(n) = 8$. The average-case running time $g(n)$ is obtained as follows:

$$g(n) = \frac{1}{n}\sum_{k=1}^{8} k \cdot n_k$$

$$= \frac{1\cdot 1 + 2\cdot 2 + 3\cdot 4 + 4\cdot 8 + 5\cdot 16 + 6\cdot 32 + 7\cdot 64 + 8\cdot 73}{200}$$

$$= \frac{1353}{200} = 6.765$$

Observe that, for the binary search, the average-case and worst-case running times are approximately equal.

4.9 Using the bubble sort algorithm, Algorithm 4.4, find the number C of comparisons and the number D of interchanges which alphabetize the $n = 6$ letters in PEOPLE.

The sequences of pairs of letters which are compared in each of the $n - 1 = 5$ passes follow: a square indicates that the pair of letters is compared and interchanged, and a circle indicates that the pair of letters is compared but not interchanged.

Pass 1. [P E] O P L E, E [P O] P L E, E O (P P) L E
E O P [P L] E E O P L [P E] E O P L E P

Pass 2. (E O) P L E P, E (O P) L E P, E O [P L] E P
E O L [P E] P, E O L E P P

Pass 3. (E O) L E P P, E [O L] E P P, E L [O E] P P
E L E O P P

Pass 4. (E L) E O P P, E [L E] O P P, E E L O P P

Pass 5. (E E) L O P P, E E L O P P

Since $n = 6$, the number of comparisons will be $C = 5 + 4 + 3 + 2 + 1 = 15$. The number D of interchanges depends also on the data, as well as on the number n of elements. In this case $D = 9$.

4.10 Prove the following identity, which is used in the analysis of various sorting and searching algorithms:

$$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

2. Repeat for I = 1 to N:
3. Repeat for J = 1 to N:
If A[I, J] ≠ 0, then: Set NUM := NUM + 1.
[End of inner loop.]
[End of outer loop.]
4. Return.

(b) 1. Set SUM := 0.
2. Repeat for J = 2 to N:
3. Repeat for I = 1 to J – 1:
Set SUM := SUM + A[I, J].
[End of inner Step 3 loop.]
4. Return.

(c) 1. Set PROD := 1. [This is analogous to setting SUM = 0.]
2. Repeat for K = 1 to N:
Set PROD := PROD*A[K, K].
[End of loop.]
3. Return.

4.13 Consider an n-square tridiagonal array A as shown in Fig. 4.31. Note that A has n elements on the diagonal and $n - 1$ elements above and $n - 1$ elements below the diagonal. Hence A contains at most $3n - 2$ nonzero elements. Suppose we want to store A in a linear array B as indicated by the arrows in Fig. 4.31; i.e.,

$$B[1] = a_{11}, \quad B[2] = a_{12}, \quad B[3] = a_{21}, \quad B[4] = a_{22}, \quad \ldots$$

Find the formula that will give us L in terms of J and K such that

$$B[L] = A[J, K]$$

(so that one can access the value of A[J, K] from the array B).

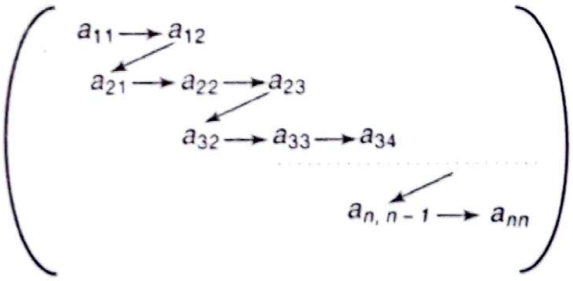


Fig. 4.31 Tridiagonal Array

Note that there are 3(J – 2) + 2 elements above A[J, K] and K – J + 1 elements to the left of A[J, K].

Hence

$$L = [3(J - 2) + 2] + [K - J + 1] + 1 = 2J + K - 2$$

The following C program demonstrates how the non-zero elements of a 5-square tridiagonal array are stored in a linear array:

Program 4.15

```
#include <stdio.h>
#include <conio.h>

void main()
{
int A[5][5];
int B[50];
int J,K,L;
int N=5;
clrscr();

for(J=0;J<N;J++)
 for(K=0;K<N;K++)
  A[J][K]=0;

for(L=0;L<50;L++)
 B[L]=0;

A[0][0]=2;
A[0][1]=22;

A[1][0]=-13;
A[1][1]=77;
A[1][2]=4;

A[2][1]=3;
A[2][2]=87;
A[2][3]=99;

A[3][2]=42;
A[3][3]=6;
A[3][4]=9;

A[4][3]=7;
A[4][4]=29;

for(J=0;J<N;J++)
 for(K=0;K<N;K++)
 {
  if(A[J][K]!=0)
  {
   L=2*(J+1)+K+1-2-1;
```

```
    B[L]=A[J][K];
   }
  }

printf("5-Square Tridiagonal Array A:\n\n");
printf("%d  %d\n\n",A[0][0],A[0][1]);
for(J=1,K=1;J<=3&&K<=3;J++,K++)
{
 L=J;
 while(L!=0)
 {
  printf("\t");
  L=L-1;
 }
 printf("%d  %d  %d\n\n",A[J][K-1],A[J][K],A[J][K+1]);
}
printf("\t\t\t\t%d  %d\n\n",A[4][3],A[4][4]);

printf("5-Square Tridiagonal Array A stored in Linear Array B:\n\n");
L=0;
while(B[L]!=0)
{
 printf("B[%d]  =  %d\n",L,B[L]);
 L=L+1;
}

getch();
}
```

Output:

```
5-Square Tridiagonal Array A:

2  22

        -13  77  4

                3  87  99

                        42  6  9

                                7  29

5-Square Tridiagonal Array A stored in Linear Array B:

B[0]  =  2
B[1]  =  22
```

```
B[2]  = -13
B[3]  = 77
B[4]  = 4
B[5]  = 3
B[6]  = 87
B[7]  = 99
B[8]  = 42
B[9]  = 6
B[10]  = 9
B[11]. = 7
B[12]  = 29
```

4.14 An n-square matrix array A is said to be *symmetric* if A[J, K] = A[K, J] for all J and K.

(a) Which of the following matrices are symmetric?

$$\begin{pmatrix} 2 & -3 & 5 \\ -3 & -2 & 4 \\ 5 & 6 & 8 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & -7 \\ 3 & 6 & -1 \\ -7 & -1 & 2 \end{pmatrix}$$

(b) Describe an efficient way of storing a symmetric matrix A in memory.

(c) Suppose A and B are two n-square symmetric matrices. Describe an efficient way of storing A and B in memory.

(a) The first matrix is not symmetric, since $a_{23} = 4$ but $a_{32} = 6$. The second matrix is not a square matrix so it cannot be symmetric, by definition. The third matrix is symmetric.

(b) Since A[J, K] = A[K, J], we need only store those elements of A which lie on or below the diagonal. This can be done in the same way as that for triangular matrices described in Example 4.25.

(c) First note that, for a symmetric matrix, we need store only either those elements on or below the diagonal or those on or above the diagonal. Therefore, A and B can be stored in an $n \times (n + 1)$ array C as pictured in Fig. 4.32, where C[J, K] = A[J, K] when $J \geq K$ but C[J, K] = B[J, K −1] when $J < K$.

$$\begin{pmatrix} a_{11} & b_{11} & b_{12} & b_{13} & \cdots & b_{1,n-1} & b_{1n} \\ a_{21} & a_{22} & b_{22} & b_{23} & \cdots & b_{2,n-1} & b_{2n} \\ a_{31} & a_{32} & a_{33} & b_{33} & \cdots & b_{3,n-1} & b_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & & a_{nn} & b_{nn} \end{pmatrix}$$

Fig. 4.32

Chapter 5

Linked Lists

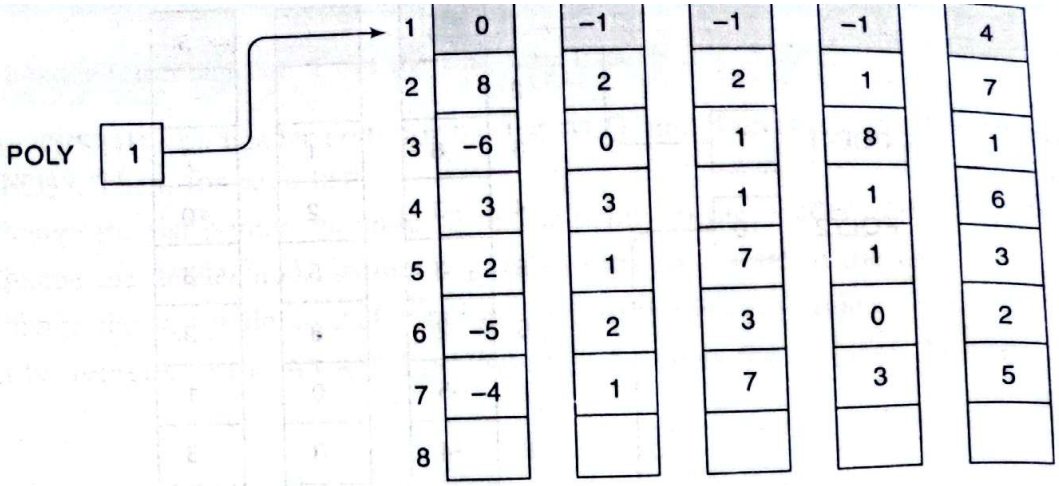


Fig. 5.50

5.10 Discuss the advantages, if any, of a two-way list over a one-way list for each of the following operations:

(a) Traversing the list to process each node
(b) Deleting a node whose location LOC is given
(c) Searching an unsorted list for a given element ITEM
(d) Searching a sorted list for a given element ITEM
(e) Inserting a node before the node with a given location LOC
(f) Inserting a node after the node with a given location LOC

(a) There is no advantage.
(b) The location of the preceding node is needed. The two-way list contains this information, whereas with a one-way list we must traverse the list.
(c) There is no advantage.
(d) There is no advantage unless we know that ITEM must appear at the end of the list, in which case we traverse the list backward. For example, if we are searching for Walker in an alphabetical listing, it may be quicker to traverse the list backward.
(e) As in part (b), the two-way list is more efficient.
(f) There is no advantage.

Remark: Generally speaking, a two-way list is not much more useful than a one-way list except in special circumstances.

5.11 Suppose LIST is a header (circular) list in memory. Write an algorithm which deletes the last node from LIST. (Compare with Solved Problem 5.5.)

The algorithm is the same as Algorithm P5.5, except now we can omit the special case

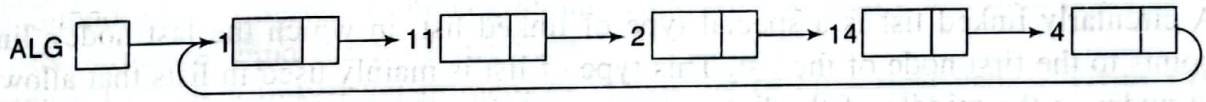
when LIST has only one node. That is, we can immediately define SAVE when LIST is not empty.

Algorithm P5.11: DELLSTH(INFO, LINK, START, AVAIL)
This algorithm deletes the last node from the header list.

1. [List empty?] If LINK[START] = NULL, then: Write: UNDERFLOW, and Exit.
2. Set PTR := LINK[START] and SAVE := START. [Initializes pointers.]
3. Repeat while LINK[PTR] ≠ START: [Traverses list seeking last node.]
   Set SAVE := PTR and PTR := LINK[PTR]. [Updates SAVE and PTR.]
   [End of loop.]
4. Set LINK[SAVE] := LINK[PTR]. [Removes last node.]
5. Set LINK[PTR] := AVAIL and AVAIL := PTR. [Returns node to AVAIL list.]
6. Exit.

5.12 Form two-way lists from the one-way header lists in Fig. 5.48.

Traverse the list ALG in the forward direction to obtain:



We require the backward pointers. These are calculated node by node. For example, the last node (with location LOC = 4) must point to the next-to-last node (with location LOC = 14). Hence

$$BACK[4] = 14$$

The next-to-last node (with location LOC = 14) must point to the preceding node (with location LOC = 2). Hence

$$BACK[14] = 2$$

And so on. The header node (with location LOC = 1) must point to the last node (with location 4). Hence

$$BACK[1] = 4$$

A similar procedure is done with the list GEOM. Figure 5.51 pictures the two-way lists. Note that there is no difference between the arrays LINK and FORW. That is, only the array BACK need be calculated.

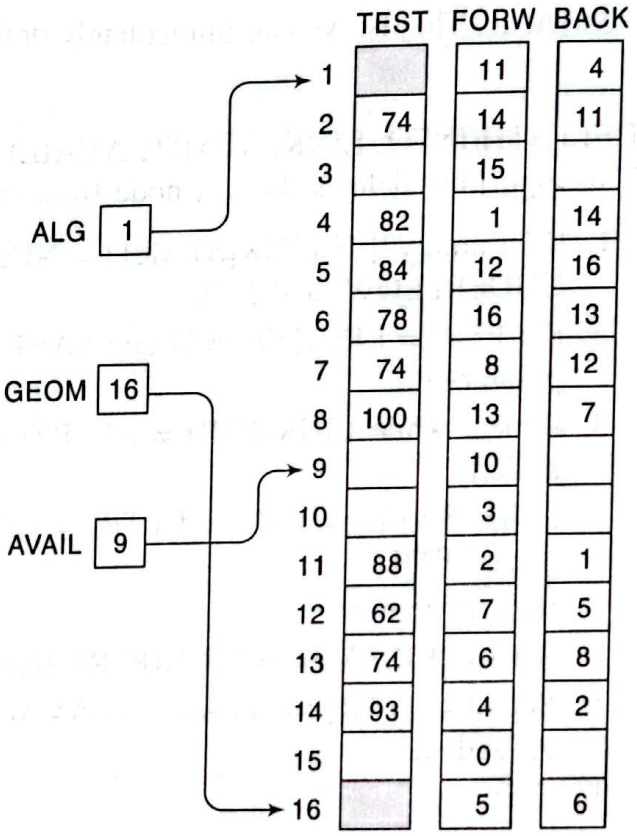


Fig. 5.51

5.13 Define a circularly linked list. Explain.

A circularly linked list is a special type of linked list, in which the last node's link field points to the first node of the list. This type of list is mainly used in lists that allow access to nodes in the middle of the list without starting at the beginning.

SUPPLEMENTARY PROBLEMS

Linked Lists

5.1 Figure 5.52 is a list of five hospital patients and their room numbers. (a) Fill in values for NSTART and NLINK so that they form an alphabetical listing of the names. (b) Fill in values for RSTART and RLINK so that they form an ordering of the room numbers.

NSTART

RSTART

	NAME	ROOM	NLINK	RLINK
1	Brown	650		
2	Smith	422		
3	Adams	704		
4	Jones	462		
5	Burns	632		

Fig. 5.52

Chapter 6

Stacks, Queues, Recursion

6. Set avrgWaitTime to stats totWaitTime / stats numCust
7. Print ("Average waiting time: " avrgWaitTime)
8. Print ("Maximum size of the queue: " stats maxQueueSize)
9. Exit

Summary

Once again we see the time-space tradeoff when choosing between different data structures for a given problem. The array representation of a priority queue is more time-efficient than the one-way list. This is because when adding an element to a one-way list, one must perform a linear search on the list. On the other hand, the one-way list representation of the priority queue may be more space-efficient than the array representation. This is because in using the array representation, overflow occurs when the number of elements in any single priority level exceeds the capacity for that level, but in using the one-way list, overflow occurs only when the total number of elements exceeds the *total* capacity. Another alternative is to use a linked list for each priority level.

SOLVED PROBLEMS

STACKS

6.1 Consider the following stack of characters, where STACK is allocated N = 8 memory cells:

STACK: A, C, D, F, K, ___, ___, ___,

(For notational convenience, we use "___" to denote an empty memory cell.) Describe the stack as the following operations take place:

(a) POP(STACK, ITEM)
(b) POP(STACK, ITEM)
(c) PUSH(STACK, L)
(d) PUSH(STACK, P)
(e) POP(STACK, ITEM)
(f) PUSH(STACK, R)
(g) PUSH(STACK, S)
(h) POP(STACK, ITEM)

The POP procedure always deletes the top element from the stack, and the PUSH procedure always adds the new element to the top of the stack. Accordingly:

(a) STACK: A, C, D, F, ___, ___, ___, ___
(b) STACK: A, C, D, ___, ___, ___, ___, ___
(c) STACK: A, C, D, L, ___, ___, ___, ___
(d) STACK: A, C, D, L, P, ___, ___, ___
(e) STACK: A, C, D, L, ___, ___, ___, ___
(f) STACK: A, C, D, L, R, ___, ___, ___
(g) STACK: A, C, D, L, R, S, ___, ___
(h) STACK: A, C, D, L, R, ___, ___, ___

6.2 Consider the data in Problem 6.1. (a) When will overflow occur? (b) When will C be deleted before D?

(a) Since STACK has been allocated N = 8 memory cells, overflow will occur when STACK contains 8 elements and there is a PUSH operation to add another element to STACK.
(b) Since STACK is implemented as a stack, C will never be deleted before D.

6.3 Consider the following stack, where STACK is allocated N = 6 memory cells: _____

STACK: AAA, DDD, EEE, FFF, GGG, _____

Describe the stack as the following operations take place: (a) PUSH(STACK, KKK), (b) POP(STACK, ITEM), (c) PUSH(STACK, LLL), (d) PUSH(STACK, SSS), (e) POP(STACK, ITEM) and (f) PUSH(STACK, TTT).

(a) KKK is added to the top of STACK, yielding

STACK: AAA, DDD, EEE, FFF, GGG, KKK

(b) The top element is removed from STACK, yielding

STACK: AAA, DDD, EEE, FFF, GGG, _____

(c) LLL is added to the top of STACK, yielding

STACK: AAA, DDD, EEE, FFF, GGG, LLL

(d) Overflow occurs, since STACK is full and another element SSS is to be added to STACK. No further operations can take place until the overflow is resolved—by adding additional space for STACK, for example.

6.4 Suppose STACK is allocated N = 6 memory cells and initially STACK is empty, or, in other words, TOP = 0. Find the output of the following module:

1. Set AAA := 2 and BBB := 5.
2. Call PUSH(STACK, AAA).
   Call PUSH(STACK, 4).
   Call PUSH(STACK, BBB + 2).
   Call PUSH(STACK, 9).
   Call PUSH(STACK, AAA + BBB).
3. Repeat while TOP ≠ 0:
   Call POP(STACK, ITEM).
   Write: ITEM.
   [End of loop.]
4. Return.

Step 1. Sets AAA = 2 and BBB = 5.

Step 2. Pushes AAA = 2, 4, BBB + 2 = 7, 9 and AAA + BBB = 7 onto STACK, yielding

STACK: 2, 4, 7, 9, 7, __

Step 3. Pops and prints the elements of STACK until STACK is empty. Since the top element is always popped, the output consists of the following sequence:

7, 9, 7, 4, 2

Observe that this is the reverse of the order in which the elements were added to STACK.

6.11 Translate, by inspection and hand, each infix expression into its equivalent prefix expression:

(a) (A – B) * (D / E)

(b) (A + B ↑ D)/(E – F) + G

Is there any relationship between the prefix expressions and the equivalent postfix expressions obtained in Solved Problem 6.7.

Using the order in which the operators are executed, translate each operator from infix to prefix notation.

(a) (A – B)*(D/E) = [–AB]*[/DE] = * – A B / D E

(b) (A + B ↑ D)/(E – F) + G = (A + [↑ BD])/[–EF] + G

= [+ A ↑ BD]/[–EF] + G

= [/ + A ↑ BD – EF] + G

= + / + A ↑ B D – E F G

The prefix expression is not the reverse of the postfix expression. However, the order of the operands—A, B, D and E in part (a) and A, B, D, E, F and G in part (b)—is the same for all three expressions, infix, postfix and prefix.

Quicksort

6.12 Suppose S is the following list of 14 alphabetic characters:

(D) A T A S T R U C T U R E (S)

Suppose the characters in S are to be sorted alphabetically. Use the quicksort algorithm to find the final position of the first character D.

Beginning with the last character S, scan the list from right to left until finding a character which precedes D alphabetically. It is C. Interchange D and C to obtain the list:

(C) A T A S T R U (D) T U R E S

Beginning with this C, scan the list toward D, i.e., from left to right, until finding a character which succeeds D alphabetically. It is T. Interchange D and T to obtain the list:

C A (D) A S (T) R U T T U R E S

Beginning with this T, scan the list toward D until finding a character which precedes D. It is A. Interchange D and A to obtain the list:

C A (A) (D) S T R U T T U R E S

Beginning with this A, scan the list toward D until finding a character which succeeds D. There is no such letter. This means D is in its final position. Furthermore, the letters before D form a sublist consisting of all letters preceding D alphabetically. and the letters after D form a sublist consisting of all the letters succeeding D alphabetically, as follows:

C A A (D) S T R U T T U R E S

Sublist Sublist

Sorting S is now reduced to sorting each sublist.

6.13 Suppose S consists of the following $n = 5$ letters:

(A) B C D (E)

Find the number C of comparisons to sort S using quicksort. What general conclusion can one make, if any?

Beginning with E, it takes $n - 1 = 4$ comparisons to recognize that the first letter A is already in its correct position. Sorting S is now reduced to sorting the following sublist with $n - 1 = 4$ letters:

A (B C D E)

Beginning with E, it takes $n - 2 = 3$ comparisons to recognize that the first letter B in the sublist is already in its correct position. Sorting S is now reduced to sorting the following sublist with $n - 2 = 3$ letters:

A B (C D E)

Similarly, it takes $n - 3 = 2$ comparisons to recognize that the letter C is in its correct position, and it takes $n - 4 = 1$ comparison to recognize that the letter D is in its correct position. Since only one letter is left, the list is now known to be sorted. Altogether we have:

$$C = 4 + 3 + 2 + 1 = 10 \text{ comparisons}$$

Similarly, using quicksort, it takes

$$C = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + 0(n) = O(n^2)$$

comparisons to sort a list with n elements when the list is already sorted. (This can be shown to be the worst case for quicksort.)

6.14 Consider the quicksort algorithm. (a) Can the arrays LOWER and UPPER be implemented as queues rather than as stacks? Why? (b) How much extra space is needed for the quicksort algorithm, or, in other words, what is the space complexity of the algorithm?

(a) Since the order in which the subsets are sorted does not matter, LOWER and UPPER can be implemented as queues, or even deques, rather than as stacks.

(b) Quicksort algorithm is an "in-place" algorithm; that is, the elements remain in their places except for interchanges. The extra space is required mainly for the stacks LOWER and UPPER. On the average, the extra space required for the algorithm is proportional to $\log n$, where n is the number of elements to be sorted.

Recursion

6.15 Let a and b denote positive integers. Suppose a function Q is defined recursively as follows:

$$Q(a, b) = \begin{cases} 0 & \text{if } a < b \\ Q(a-b, b) + 1 & \text{if } b \le a \end{cases}$$

(a) Find the value of $Q(2, 3)$ and $Q(14, 3)$.

(b) What does this function do? Find $Q(5861, 7)$.

(a) $Q(2, 3) = 0$ since $2 < 3$

$$\begin{aligned} Q(14, 3) &= Q(11, 3) + 1 \\ &= [Q(8, 3) + 1] + 1 = Q(8, 3) + 2 \\ &= [Q(5, 3) + 1] + 2 = Q(5, 3) + 3 \\ &= [Q(2, 3) + 1] + 3 = Q(2, 3) + 4 \\ &= 0 + 4 = 4 \end{aligned}$$

(b) Each time b is subtracted from a, the values of Q is increased by 1. Hence $Q(a, b)$ finds the quotient when a is divided by b. Thus,

$$Q(5861, 7) = 837$$

6.16 Let n denote a positive integer. Suppose a function L is defined. recursively as follows:

$$L(n) = \begin{cases} 0 & \text{if } n = 1 \\ L(\lfloor n/2 \rfloor + 1) & \text{if } n > 1 \end{cases}$$

(Here $\lfloor k \rfloor$ denotes the "floor" of k, that is, the greatest integer which does not exceed k. See Sec. 2.2.)

(a) Find $L(25)$.

(b) What does this function do?

(a) $L(25) = L(12) + 1$

$$\begin{aligned} &= [L(6) + 1] + 1 = L(6) + 2 \\ &= [L(3) + 1] + 2 = L(3) + 3 \\ &= [L(1) + 1] + 3 = L(1) + 4 \\ &= 0 + 4 = 4 \end{aligned}$$

(b) Each time n is divided by 2, the value of L is increased by 1. Hence L is the greatest integer such that

$$2^L \leq n$$

Accordingly, this function finds

$$L = \lfloor \log_2 n \rfloor$$

6.17 Suppose the Fibonacci numbers $F_{11} = 89$ and $F_{12} = 144$ are given.

(a) Should one use recursion or iteration to obtain F_{16}? Find F_{16}.

(b) Write an iterative procedure to obtain the first N Fibonacci numbers F[1], F[2],..., F[N], where N > 2. (Compare this with the recursive Procedure 6.10.)

(a) The Fibonacci numbers should be evaluated by using iteration (that is, by evaluating from the bottom up), rather than by using recursion (that is, evaluating from the top down).

Recall that each Fibonacci number is the sum of the two preceding Fibonacci numbers. Beginning with F_{11} and F_{12} we have

$$F_{13} = 89 + 144 = 233, \quad F_{14} = 144 + 233 = 377, \quad F_{15} = 233 + 377 = 610$$

and hence

$$F_{16} = 377 + 610 = 987$$

(b) **Procedure P6.17:** FIBONACCI(F, N)

This procedure finds the first N Fibonacci numbers and assigns them to an array F.

1. Set F[1] := 1 and F[2] := 1.
2. Repeat for L = 3 to N:
   Set F[L] := F[L − 1] + F[L − 2].
   [End of loop.]
3. Return.

(We emphasize that this iterative procedure is much more efficient than the recursive Procedure 6.10.)

6.18 Use the definition of the Ackermann function (Definition 6.3) to find $A(1, 3)$.

We have the following 15 steps:

1. $A(1, 3) = A(0, A(1, 2))$
2. $A(1, 2) = A(0, A(1, 1))$
3. $A(1, 1) = A(0, A(1, 0))$
4. $A(1, 0) = A(0, 1)$
5. $A(0, 1) = 1 + 1 = 2$
6. $A(1, 0) = 2$
7. $A(1, 1) = A(0, 2)$
8. $A(0, 2) = 2 + 1 = 3$
9. $A(1, 1) = 3$
10. $A(1, 2) = A(0, 3)$
11. $A(0, 3) = 3 + 1 = 4$
12. $A(1, 2) = 4$
13. $A(1, 3) = A(0, 4)$
14. $A(0, 4) = 4 + 1 = 5$
15. $A(1, 3) = 5$

The forward indention indicates that we are postponing an evaluation and are recalling the definition, and the backward indention indicates that we are backtracking.

Observe that the first formula in Definition 6.3 is used in Steps 5, 8, 11 and 14, the second formula in Step 4 and the third formula in Steps 1, 2 and 3. In the other Steps we are backtracking with substitutions.

6.19 Suppose a recursive procedure P contains only one recursive call:

Step K. Call P.

Indicate the reason that the stack STADD (for the return addresses) is not necessary.

Since there is only one recursive call, control will always be transferred to Step K + 1 on a Return, except for the final Return to the main program. Accordingly, instead of maintaining the stack STADD (and the local variable ADD), we simply write

(c) Go to Step K + 1

6.26 A linked queue Q and an AVAIL list maintained as a linked stack, are as shown in Fig. 6.38. Trace the contents of the memory after the execution of the following operations on the linked queue Q.

	INFO	LINK
23	56	NULL
24	8	29
25	12	34
26	5	NULL
27	76	30
28	123	31
29	09	33
30	45	23
31	23	26
32	56	25
33	78	28
34	123	24

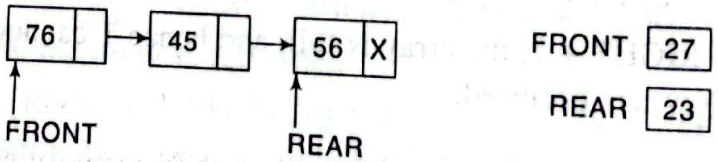
Fig. 6.38

AVAIL: [32] Linked queue Q: FRONT: [27] REAR: [23]

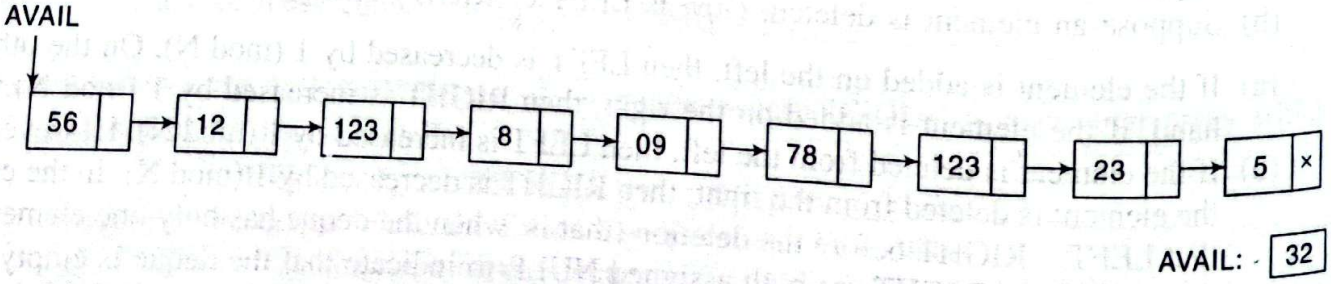
(i) Insert 567
(ii) Delete
(iii) Delete
(iv) Insert 67

It is easier to show the memory contents after extracting the linked queue Q and AVAIL list and performing the operations on the lists.

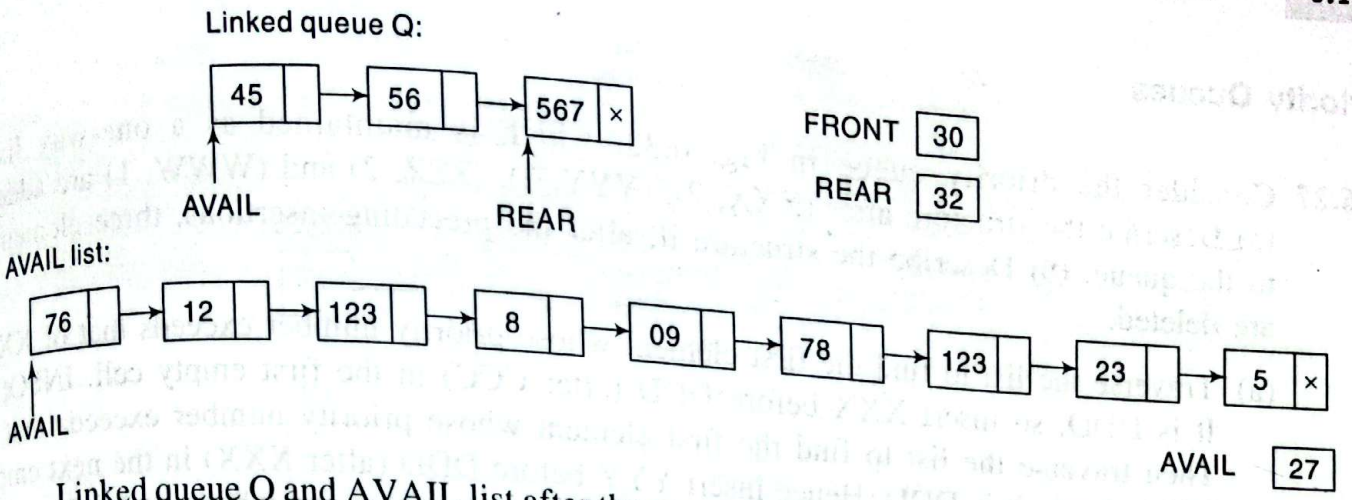
Linked queue Q:



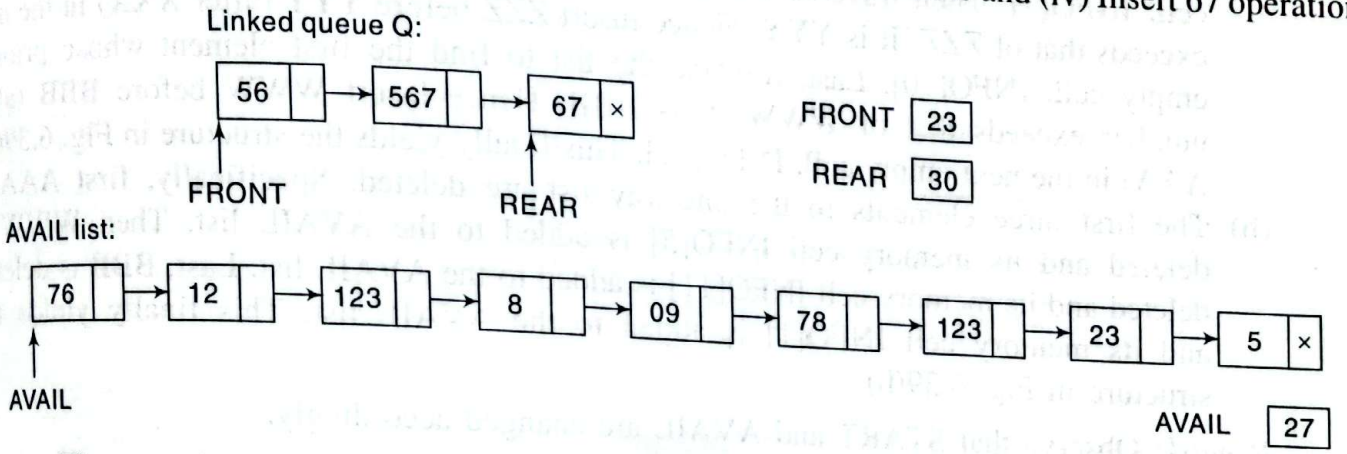
AVAIL list:



Linked queue Q and AVAIL list after the execution of (i) Insert 567 and (ii) Delete operations



Linked queue Q and AVAIL list after the execution of (iii) Delete and (iv) Insert 67 operations

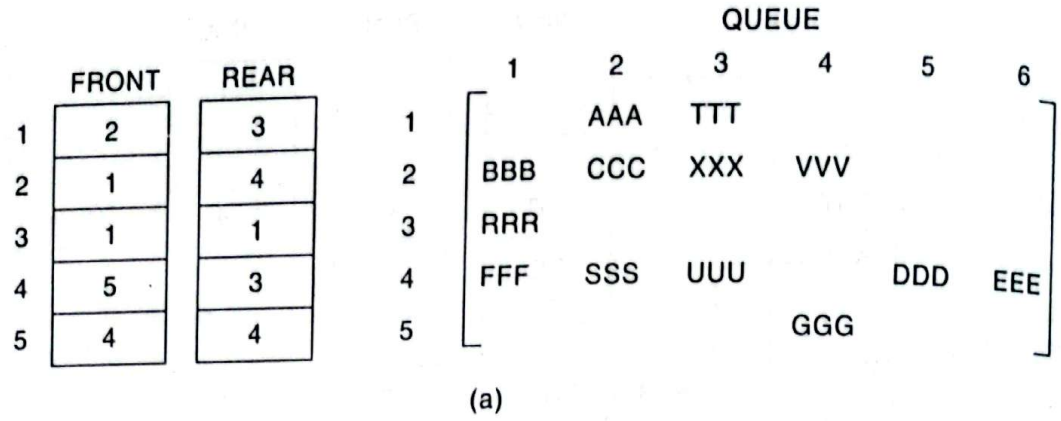


The snapshot of the memory after the execution of the operations is shown below:

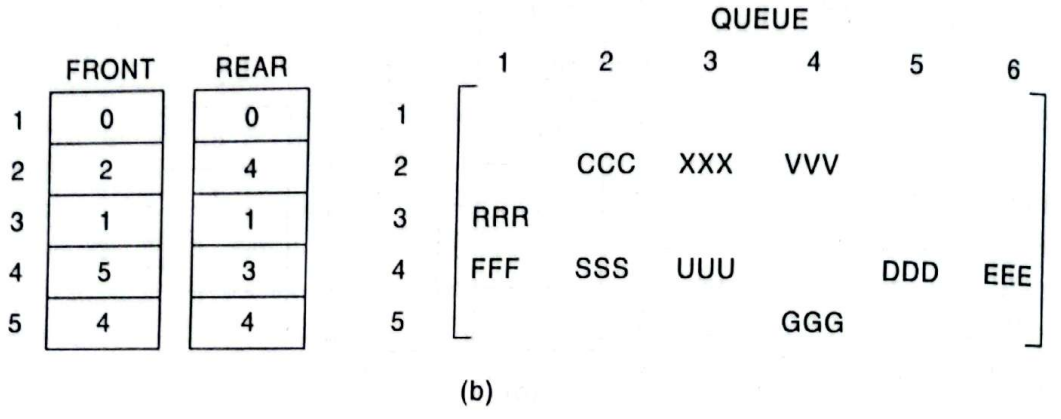
	INFO	LINK
23	56	32
24	8	29
25	12	34
26	5	NULL
27	76	25
28	123	31
29	09	33
30	67	NULL
31	23	26
32	567	30
33	78	28
34	123	24

AVAIL: 27 Linked queue Q: FRONT: 23 REAR: 30

Observe how the insertions into the linked list Q calls for a pop operation from the AVAIL list maintained as a linked stack and the deletions from Q call for push operations into the AVAIL list.



(a)



(b)

Fig. 6.40

6.29 Which data structure is used to perform recursion?

Stack Because of its LIFO (Last In First Out) property, it remembers its 'caller'; so the function knows where to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls.

Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, stack is to be used.

6.30 List two significant applications of a queue.

Two important applications of queues are categorizing data, and queue simulation.

6.31 Differentiate between enqueue and dequeue.

The insert operation of the queue structure is called enqueue, and the delete operation is called dequeue.

6.32 Define a circular queue.

A circular queue, by definition, is a queue in which the element next to the last element is the first element.

6.33 Sort the given values using Quick Sort.

65	70	75	80	85	60	55	50	45

Sorting takes place from the pivot value, which is the first value of the given elements—this is marked bold. The values at the left pointer and right pointer are indicated using L and R respectively.

65	70^{L}	75	80	85	60	55	50	45^{R}

Since pivot is not yet changed, the same process is continued after interchanging the values at L and R positions

65	45	75^{L}	80	85	60	55	50^{R}	70
65	45	50	80^{L}	85	60	55^{R}	75	70
65	45	50	55	85^{L}	60^{R}	80	75	70
65	45	50	55	60^{R}	85^{L}	80	75	70

When the L and R pointers cross each other, the pivot value is interchanged with the value at right pointer. If the pivot is changed, it means that the pivot has occupied its original position in the sorted order (shown in bold italics) and hence two different arrays are formed—one from start of the original array to the pivot position–1 and the other from pivot position +1 to end.

$\mathbf{60}^{L}$	45	50	55^{R}	***65***	$\mathbf{85}^{L}$	80	75	70^{R}
$\mathbf{55}^{L}$	45	50^{R}	***60***	***65***	$\mathbf{70}^{R}$	80^{L}	75	***85***
$\mathbf{50}^{L}$	45^{R}	***55***	***60***	***65***	***70***	80^{L}	75^{R}	***85***

In the next pass, we get the sorted form of the array.

45	***50***	***55***	***60***	***65***	***70***	***75***	***80***	***85***

SUPPLEMENTARY PROBLEMS

STACKS

6.1 Consider the following stack of city names:

STACK: London, Berlin, Rome, Paris, _____, _____

(a) Describe the stack as the following operations take place:

(i) PUSH(STACK, Athens),
(ii) POP(STACK, ITEM)
(iii) POP(STACK, ITEM)
(iv) PUSH(STACK, Madrid)
(v) PUSH(STACK, Moscow)
(vi) POP(STACK, ITEM)

(b) Describe the stack if the operation POP(STACK, ITEM) deletes London.

6.2 Consider the following stack where STACK is allocated N = 4 memory cells:

STACK: AAA, BBB, _____, _____

E: 6 + 2 ↑ 3 ↑ 2 − 4 * 5

Evaluate the expression E, (a) assuming that exponentiation is performed from left to right, as are the other operations, and (b) assuming that exponentiation is performed from right to left.

6.10 Consider each of the following postfix expressions:

P_1: 5, 3, +, 2, *, 6, 9, 7, −, /, −

P_2: 3, 5, +, 6, 4, −, *, 4, 1, −, 2, ↑, +

P_3: 3, 1, +, 2, ↑, 7, 4, −, 2, *, +, 5, −

Translate, by inspection and hand, each expression into infix notation and then evaluate.

6.11 Evaluate each postfix expression in Supplemetary Problem 6.10, using Algorithm 6.5.

6.12 Use Algorithm 6.6 to translate each infix expression into its equivalent postfix expression:

(a) (A − B)/((D + E) * F) (b) ((A + B)/D) ↑ ((E − F) * G)

(Compare with Supplementary Problem 6.6.)

Recursion

6.13 Let J and K be integers and suppose Q(J, K) is recursively defined by

$$Q(J, K) = \begin{cases} 5 & \text{if } J < K \\ Q(J - K, K + 2) + J & \text{if } J \geq K \end{cases}$$

Find Q(2, 7), Q(5, 3) and Q(15, 2)

6.14 Let A and B be nonnegative integers. Suppose a function GCD is recursively defined as follows:

$$GCD(A, B) = \begin{cases} GCD(B, A) & \text{if } A < B \\ A & \text{if } B = 0 \\ GCD(B, MOD(A, B)) & \text{otherwise} \end{cases}$$

(Here MOD(A, B), read "A modulo B," denotes the remainder when A is divided by B.) (a) Find GCD(6, 15), GCD(20, 28) and GCD(540, 168). (b) What does this function do?

6.15 Let N be an integer and suppose H(N) is recursively defined by

$$H(N) = \begin{cases} 3 * N & \text{if } N < 5 \\ 2 * H(N - 5) + 7 & \text{otherwise} \end{cases}$$

(a) Find the base criteria of H and (b) find H(2), H(8) and H(24).

6.16 Use Definition 6.3 (of the Ackermann function) to find $A(2, 2)$.

6.17 Let M and N be integers and suppose F(M, N) is recursively defined by

$$F(M, N) = \begin{cases} 1 & \text{if } M = 0 \text{ or } M \geq N \geq 1 \\ F(M - 1, N) + F(M - 1, N - 1) & \text{otherwise} \end{cases}$$

(a) Find (4, 2), F(1, 5) and F(2, 4). (b) When is F(M, N) undefined?

6.18 Let A be an integer array with N elements. Suppose X is an integer function defined by

$$X(K) = X(A, N, K) = \begin{cases} 0 & \text{if } K = 0 \\ X(K-1) + A(K) & \text{if } 0 < K \le N \\ X(K-1) & \text{if } K > N \end{cases}$$

Find X(5) for each of the following arrays:
(a) N = 8, A: 3, 7, –2, 5, 6, –4, 2, 7 (b) N = 3, A: 2, 7, –4
What does this function do?

6.19 Show that the recursive solution to the Towers of Hanoi problem in Sec. 6.8 requires $f(n) = 2^n - 1$ moves for n disks. Show that no other solution uses fewer than $f(n)$ moves.

6.20 Suppose S is a string with N characters. Let SUB(S, J, L) denote the substring of S beginning in the position J and having length L. Let A//B denote the concatenation of strings A and B. Suppose REV(S, N) is recursively defined by

$$REV(S, N) = \begin{cases} S & \text{if } N = 1 \\ SUB(S, N, 1)//REV(SUB(S, 1, N-1) & \text{otherwise} \end{cases}$$

(a) Find REV(S, N) when (i) N = 3, S = *abc* and (ii) N = 5, S = *ababc*. (b) What does this function do?

Queues; Deques

6.21 Consider the following queue where QUEUE is allocated 6 memory cells:

FRONT = 2, REAR = 5 QUEUE: _____, London, Berlin, Rome, Paris, _____

Describe the queue, including FRONT and REAR, as the following operations take place: (a) Athens is added, (b) two cities are deleted, (c) Madrid is added, (d) Moscow is added, (e) three cities are deleted and (f) Oslo is added.

6.22 Consider the following deque where DEQUE is allocated 6 memory cells:

LEFT = 2, RIGHT = 5 DEQUE: _____, London, Berlin, Rome, Paris, _____

Describe the deque, including LEFT and RIGHT, as the following operations take place:
(a) Athens is added on the left.
(b) Two cities are deleted from the right.
(c) Madrid is added on the left.
(d) Moscow is added on the right.
(e) Two cities are deleted from the right.
(f) A city is deleted from the left.
(g) Oslo is added on the left.

6.23 Suppose a queue is maintained by a circular array QUEUE with N = 12 memory cells. Find the number of elements in QUEUE if (a) FRONT = 4, REAR = 8; (b) FRONT = 10, REAR = 3; and (c) FRONT = 5, REAR = 6 and then two elements are deleted.

6.24 Consider the priority queue in Fig. 6.42(b), which is maintained as a one-way list.
(a) Describe the structure if two elements are deleted.
(b) Describe the structure if, after the preceding deletions, the elements (RRR, 3), (SSS, 1), (TIT, 3) and (UUU, 2) are added to the queue.
(c) Describe the structure if, after the preceding insertions, three elements are deleted.

Chapter 7

Trees

SOLVED PROBLEMS

Binary Trees

7.1 Suppose T is the binary tree stored in memory as in Fig. 7.93. Draw the diagram of T.

The tree T is drawn from its root R downward as follows:

(a) The root R is obtained from the value of the pointer ROOT. Note that ROOT = 5. Hence INFO[5] = 60 is the root R of T.

(b) The left child of R is obtained from the left pointer field of R. Note that LEFT[5] = 2. Hence INFO[2] = 30 is the left child of R.

(c) The right child of R is obtained from the right pointer field of R. Note that RIGHT[5] = 6. Hence INFO[6] = 70 is the right child of R.

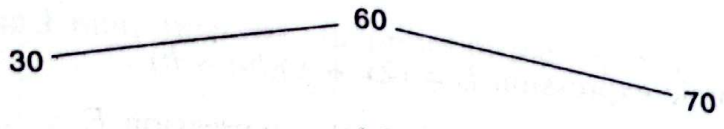
We can now draw the top part of the tree as pictured in Fig. 7.94(a). Repeating the above process with each new node, we finally obtain the required tree T in Fig. 7.94(b).

ROOT: 5

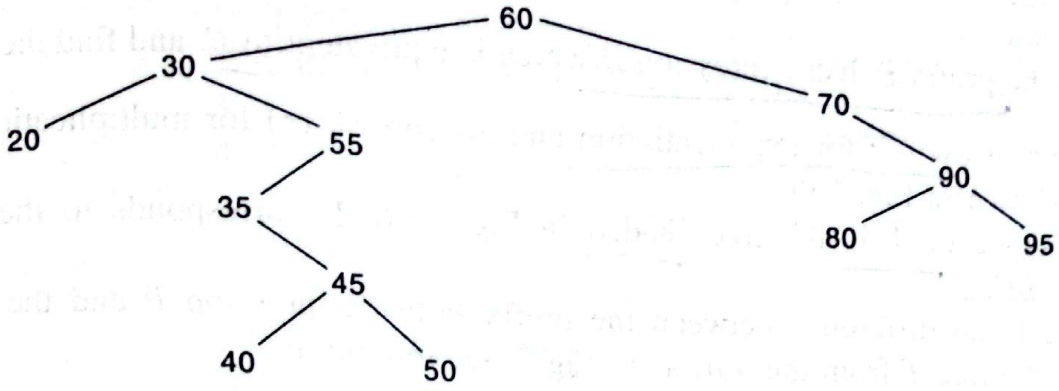
AVAIL: 9

	INFO	LEFT	RIGHT
1	20	0	0
2	30	1	13
3	40	0	0
4	50	0	0
5	60	2	6
6	70	0	8
7	80	0	0
8	90	7	14
9		10	
10		0	
11	35	0	12
12	45	3	4
13	55	11	0
14	95	0	0

Fig. 7.93



(a)



(b)

Fig. 7.94

7.2 A binary tree T has 9 nodes. The inorder and preorder traversals of T yield the following sequences of nodes:

Inorder:	E	A	C	K	F	H	D	B	G
Preorder:	F	A	E	K	C	D	H	G	B

Draw the tree T.

The tree T is drawn from its root downward as follows.

(a) The root of T is obtained by choosing the first node in its preorder. Thus F is the root of T.

(b) The left child of the node F is obtained as follows. First use the inorder of T to find the nodes in the left subtree T_1 of F. Thus T_1 consists of the nodes E, A, C and K. Then the left child of F is obtained by choosing the first node in the preorder of T_1 (which appears in the preorder of T). Thus A is the left son of F.

(c) Similarly, the right subtree T_2 of F consists of the nodes H, D, B and G, and D is the root of T_2, that is, D is the right child of F.
Repeating the above process with each new node, we finally obtain the required tree in Fig. 7.95.

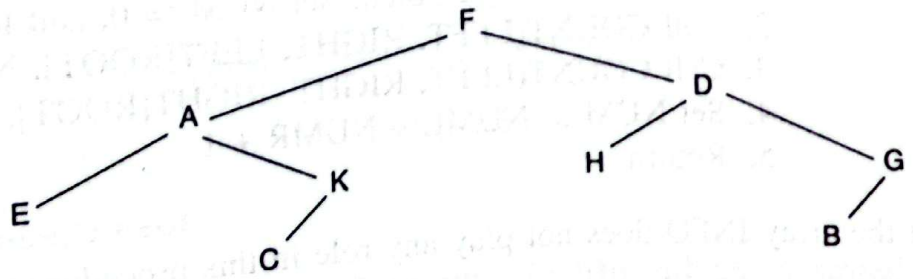


Fig. 7.95

The depth DEP of T is 1 more than the maximum of the depths of the left and right subtrees of T. Accordingly:

Procedure P7.5: DEPTH(LEFT, RIGHT, ROOT, DEP)
This procedure finds the depth DEP of a binary tree T in memory.

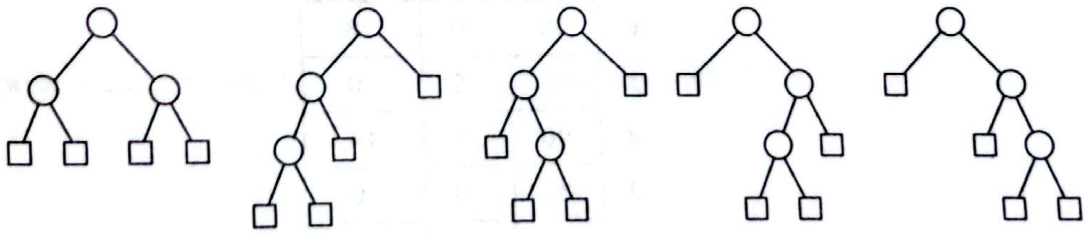
1. If ROOT = NULL, then: Set DEP := 0, and Return.
2. Call DEPTH(LEFT, RIGHT, LEFT[ROOT], DEPL).
3. Call DEPTH(LEFT, RIGHT, RIGHT[ROOT], DEPR).
4. If DEPL ≥ DEPR, then:
   Set DEP := DEPL + 1.
   Else:
   Set DEP := DEPR + 1.
   [End of If structure.]
5. Return.

(Observe that the array INFO does not play any role in this procedure.)

7.6 Draw all the possible nonsimilar trees T where:
(a) T is a binary tree with 3 nodes.
(b) T is a 2-tree with 4 external nodes.

(a) There are five such trees, which are pictured in Fig. 7.97(a).
(b) Each 2-tree with 4 external nodes is determined by a binary tree with 3 nodes, i.e., by a tree in part (a). Thus there are five such trees, which are pictured in Fig. 7.97(b).

(a) Binary trees with 3 nodes



(b) Extended binary trees with 4 external nodes

Fig. 7.97

Binary Search Trees; Heaps

7.7 Consider the binary search tree T in Fig. 7.94(b), which is stored in memory as in Fig. 7.93. Suppose ITEM = 33 is added to the tree T. (a) Find the new tree T. (b) Which changes occur in Fig. 7.93?

The tree is now a heap. The dotted edges indicate that an exchange has taken place. The unshaded area indicates that part of the tree which forms a heap. Observe that the heap is created from the top down (although individual elements move up the tree).

AVL Search Trees, *m*-Way Search Trees, B Trees

7.13 Find which of the following is a (i) Binary search tree (ii) AVL search tree (iii) Skewed binary search tree (iv) Binary tree (neither of (i), (ii) and (iii))

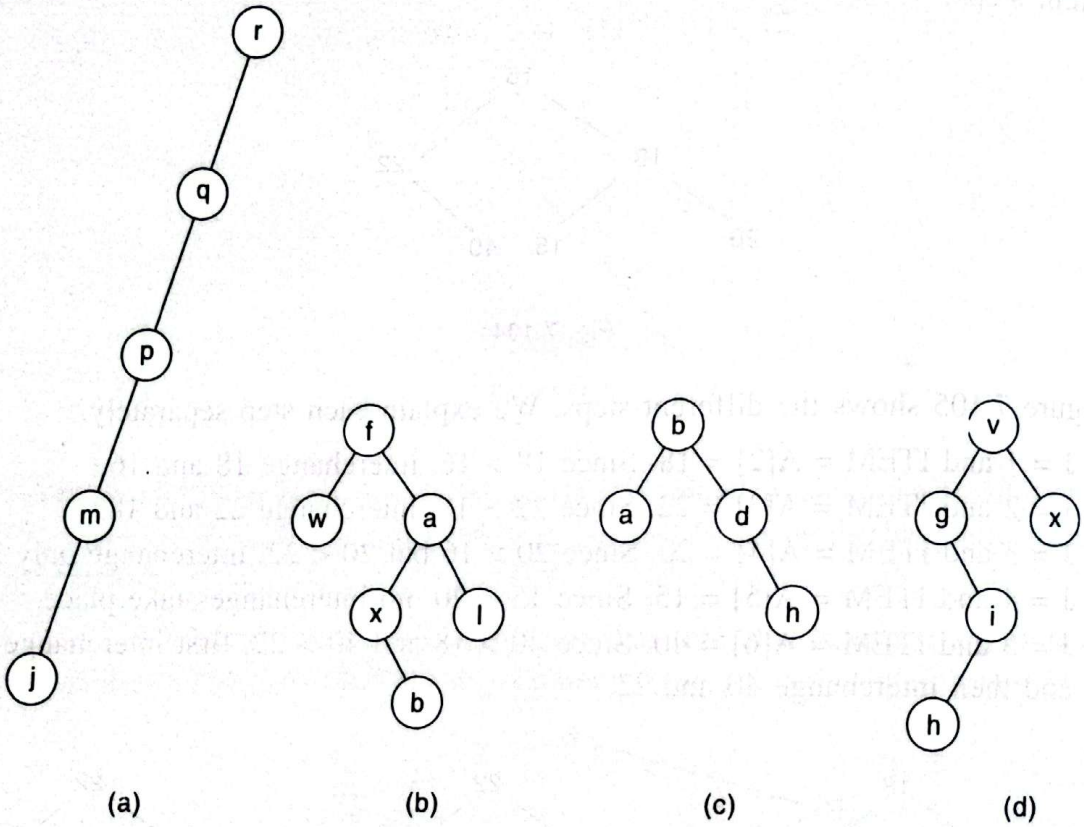


Fig. 7.106

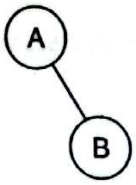
(a) Skewed binary search tree
(b) Binary tree
(c) AVL search tree
(d) Binary search tree

7.14 Insert the following keys in the order shown to construct an AVL search tree.

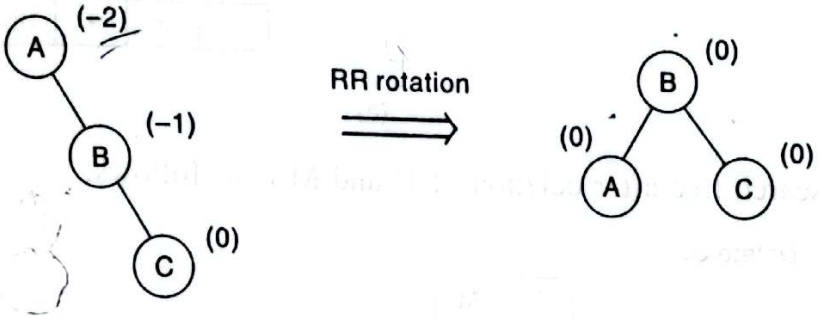
A, B, C, D, E

Delete the last two keys in the order of Last in First out.

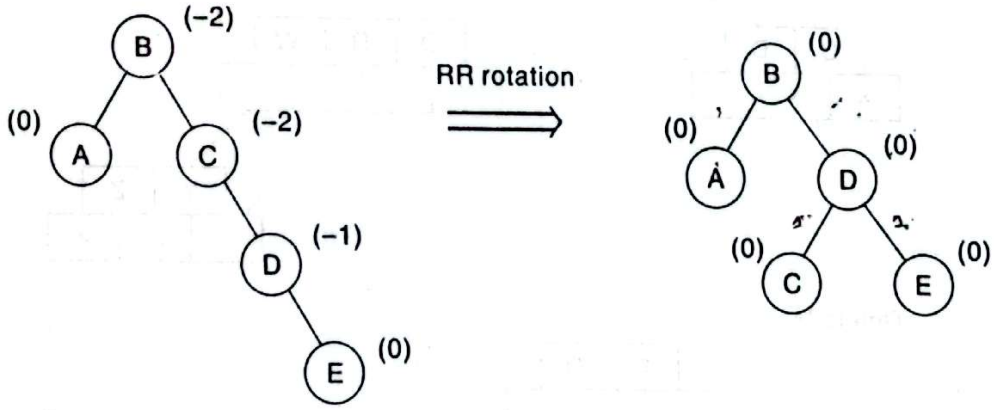
Insert A, B



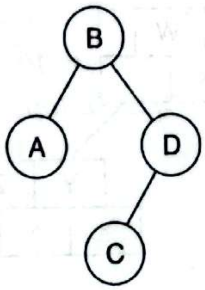
Insert C



Insert D, E



Delete E



Delete D

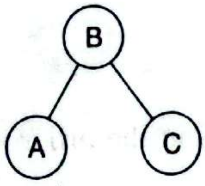
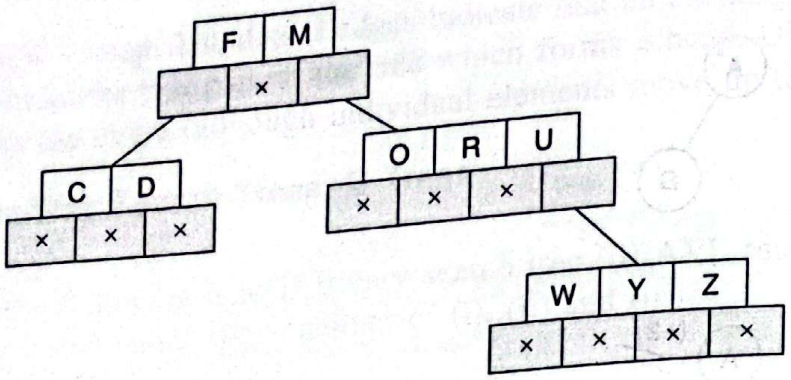


Fig. 7.107

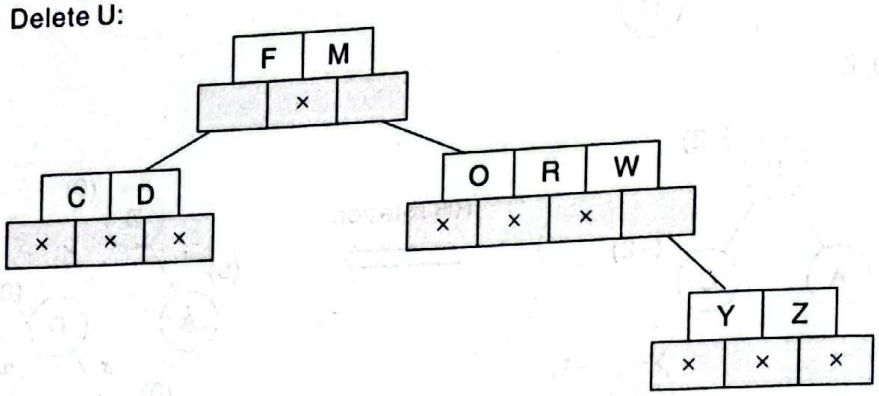
7.15 In the following 4-way search tree trace the tree after deletion of (i) U and (ii) M.



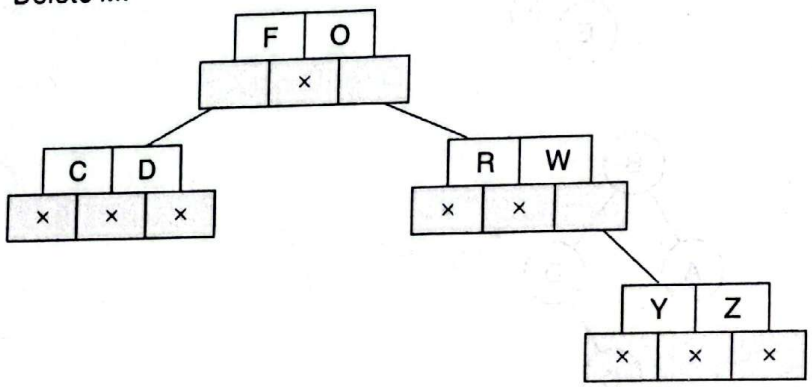
(a)

The 4-way search tree after deletion of U and M is as follows:

Delete U:



Delete M:



(b)

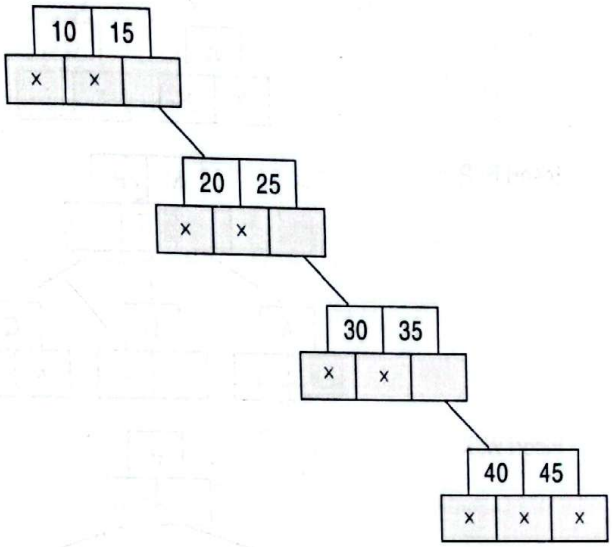
Fig. 7.108

7.16 Construct a 3-way search tree for the list of keys in the order shown below. What are your observations?

List A: 10, 15, 20, 25, 30, 35, 40, 45.
List B: 20, 35, 40, 10, 15, 25, 30, 45.

The corresponding trees are shown in Fig. 7.109.

3-way search tree for List A:



3-way search tree for List B:

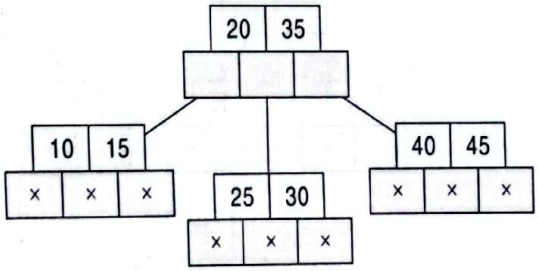


Fig. 7.109

The observation is that for the same set of keys, various m-way search trees of varying heights can be constructed. The height of an m-way search tree varies from a low of $\log_m(n + 1)$ to a high of n. For minimum number of accesses related to the efficient retrieval of keys, it is essential that the height of an m-way search tree is close to $\log_m(n + 1)$. In the problem, while the 3-way search tree corresponding to List B is of balanced height with a value equal to $\log_m(n + 1)$, the same corresponding to List A shows an imbalance in height.

7.17 Construct a B-tree of order 3 (2–3 tree) by inserting the following keys in the order shown into an empty B-tree:

M Q A N P W X T G E J

The 2–3 tree constructed is shown in Fig. 7.110.

7.18 In the B-tree of order 3 given in Fig. 7.111(a) delete 75, 60.

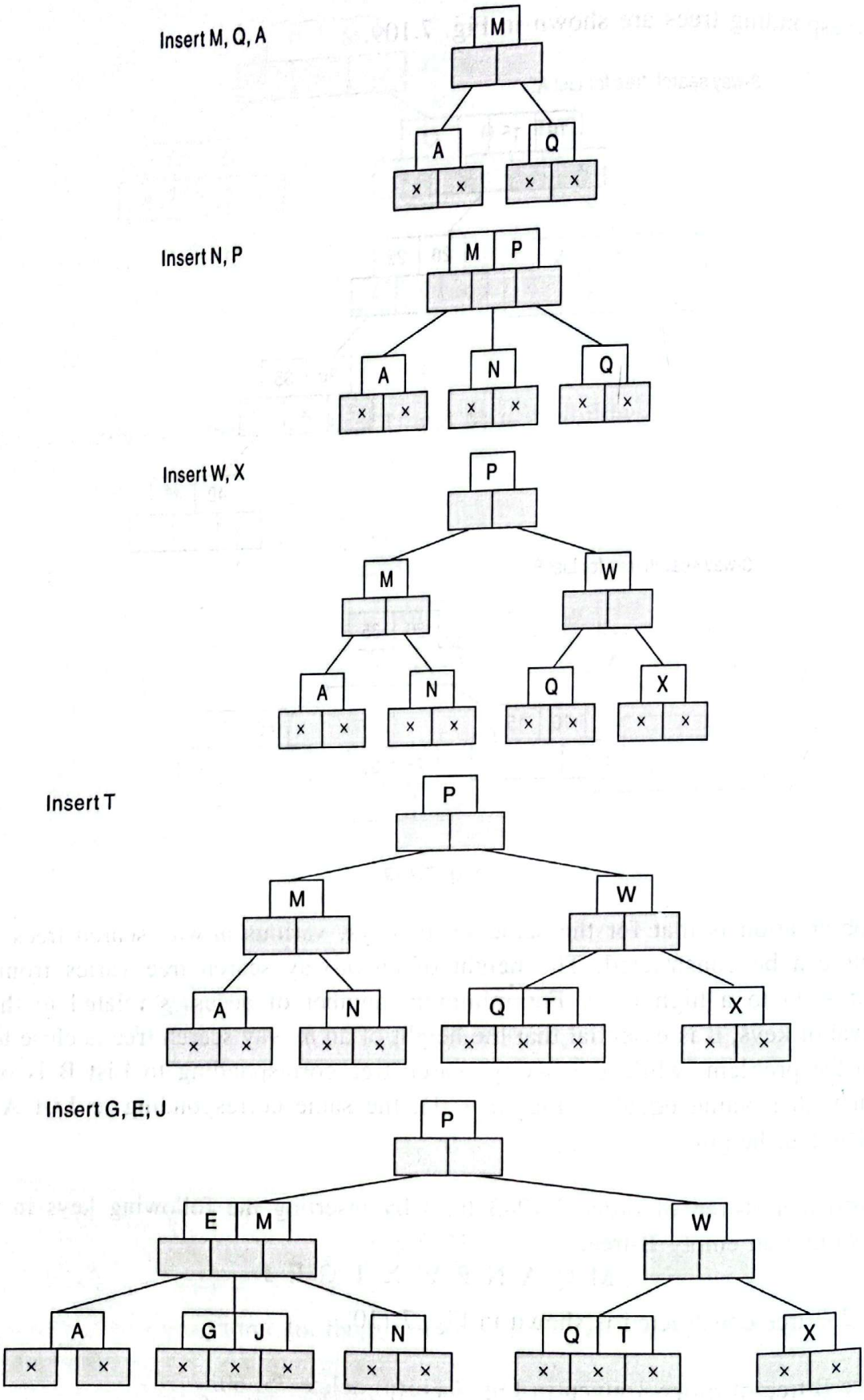

Insert M, Q, A
M
A
Q
Insert N, P
M P
A
N
Q
Insert W, X
P
M
W
A
N
Q
X
Insert T
P
M
W
A
N
Q T
X
Insert G, E; J
P
E M
W
A
G J
N
Q T
X

Fig. 7.110

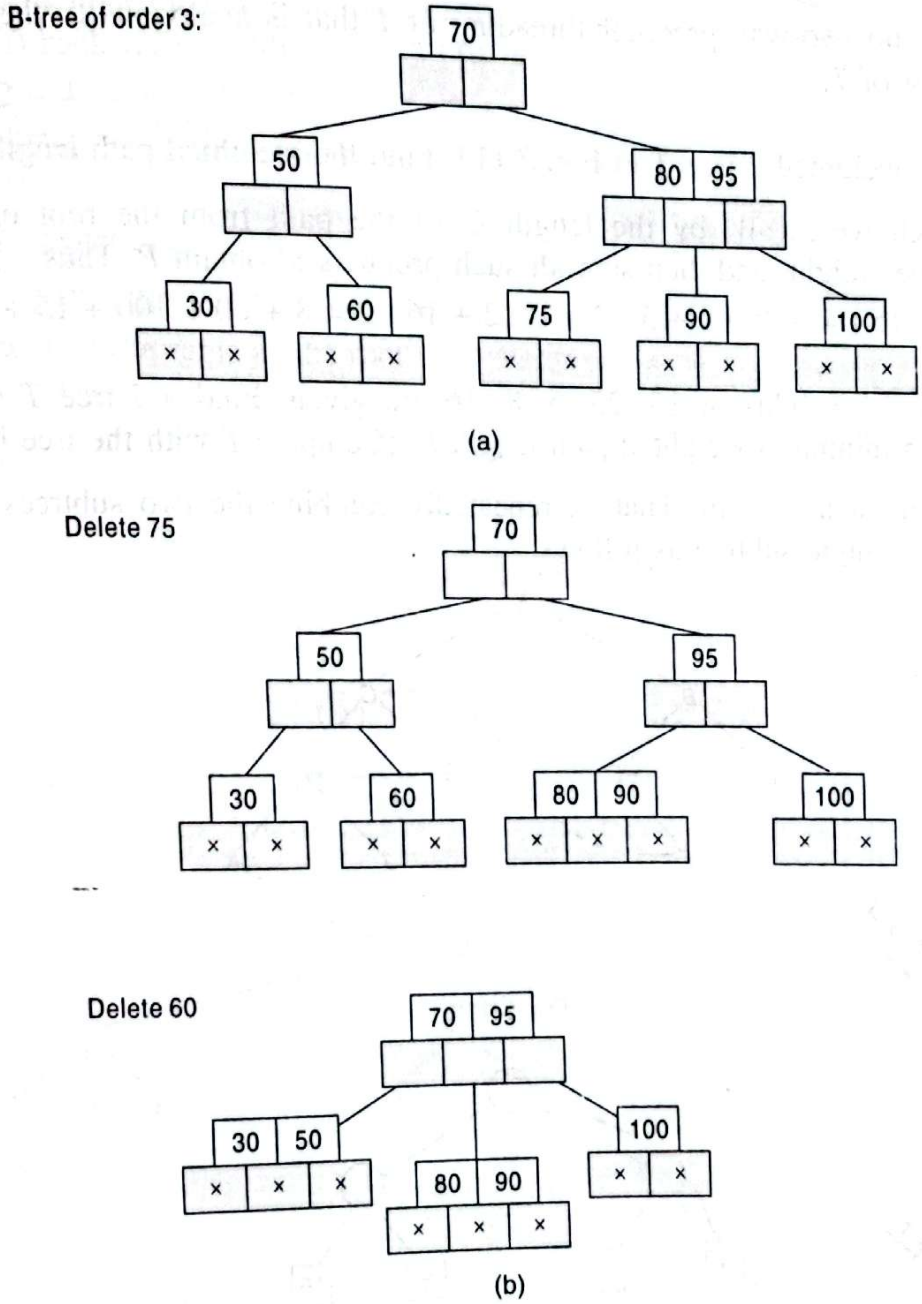


Fig. 7.111

Miscellaneous Problems

7.1 Consider the binary tree T in Fig. 7.1. (a) Find the one-way preorder threading of T. (b) Find the two-way preorder threading of T.

(a) Replace the right null subtree of a terminal node N by a thread pointing to the successor of N in the preorder traversal of T. Thus there is a thread from D to E, since E is visited after D in the preorder traversal of T. Similarly, there is a thread from F to C, from G to H and from L to K. The threaded tree appears in Fig. 7.112. The terminal node K has no thread, since it is the last node in the preorder traversal of T. (On the other hand, if T had a header node Z, then there would be a thread from K back to Z.)

(b) There is no two-way preorder threading of T that is analogous to the two-way inorder threading of T.

7.20 Consider the weighted 2-tree T in Fig. 7.113. Find the weighted path length P of the tree T.

Multiply each weight W_i by the length L_i of the path from the root of T to the node containing the weight, and then sum all such products to obtain P. Thus:

$$P = 4 \cdot 2 + 15 \cdot 4 + 25 \cdot 4 + 5 \cdot 3 + 8 \cdot 2 + 16 \cdot 2 = 8 + 60 + 100 + 15 + 16 + 32 = 231$$

7.21 Suppose the six weights 4, 15, 25, 5, 8, 16 are given. Find a 2-tree T with the given weights and a minimum weighted path length P. (Compare T with the tree in Fig. 7.113.)

Use the Huffman algorithm. That is, repeatedly combine the two subtrees with minimum weights into a single subtree as follows:

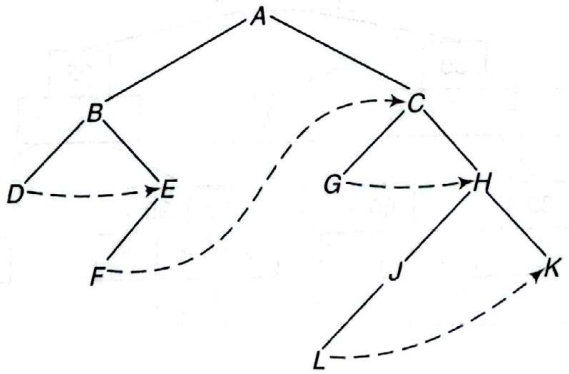


Fig. 7.112

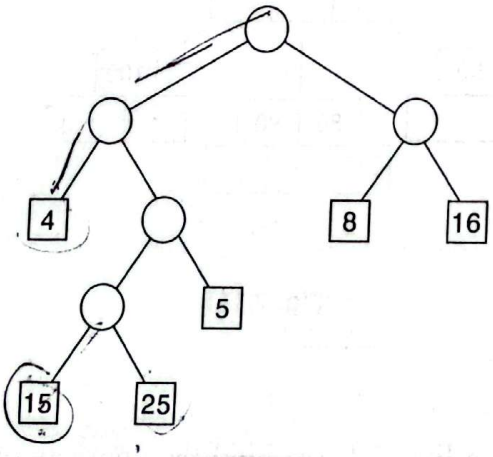


Fig. 7.113

(a) 4, 15, 25, 5, 8, 16
(b) 15, 25, 9, 8, 16
(c) 15, 25, 17, 16
(d) 25, 17, 31
(e) 42, 31
(f) 73

(The circled number indicates the root of the new subtree in the step.) The tree T is drawn from Step (f) backward, yielding Fig. 7.114. With the tree T, compute

$$P = 25 \cdot 2 + 4 \cdot 4 + 5 \cdot 4 + 8 \cdot 3 + 15 \cdot 2 + 16 \cdot 2 = 50 + 16 + 20 + 24 + 30 + 32 = 172$$

(The tree in Fig. 7.113 has weighted path length 231.)

7.22 Consider the general tree T in Fig. 7.115(a). Find the corresponding binary tree T′.

The nodes of T′ will be the same as the nodes of the general tree T, and in particular, the root of T′ will be the same as the root of T. Furthermore, if N is a node in the binary tree T′,

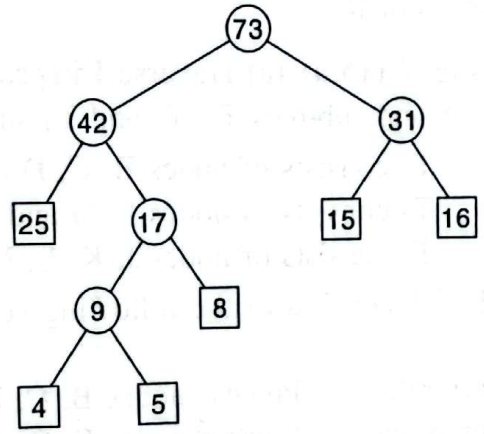
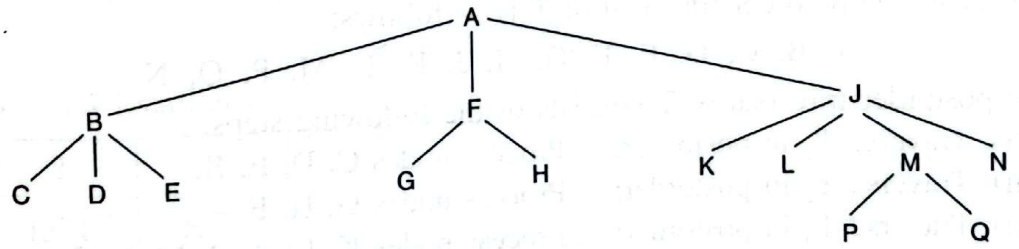
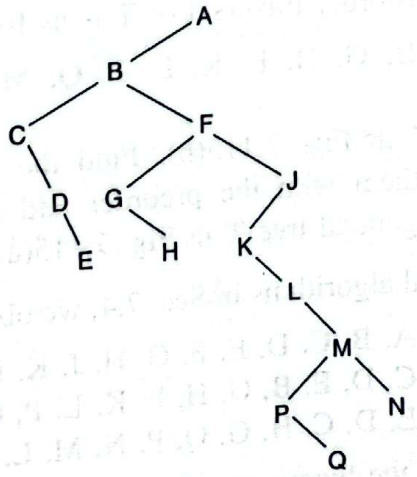


Fig. 7.114



(a) General tree T



(b) Binary tree T′

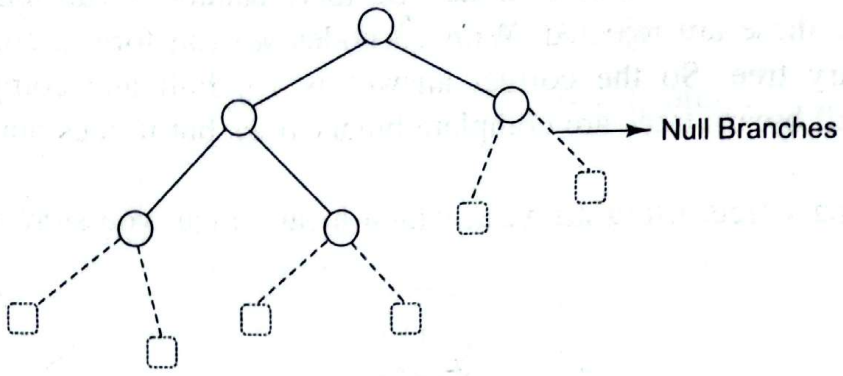
Fig. 7.115

7.28 A binary tree with 20 nodes has _____ NULL branches.

21

Explanation:

Let us take a tree with 5 nodes ($n = 5$)



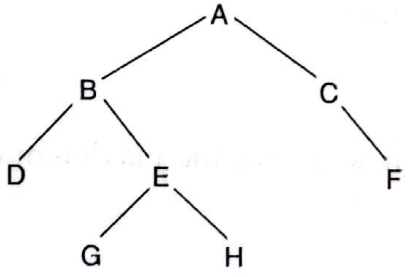
It will have only 6 (i.e., 5+1) null branches. Generally, a binary tree with n nodes has exactly n+1 NULL nodes.

SUPPLEMENTARY PROBLEMS

Binary Trees

7.1 Consider the tree T in Fig. 7.116(a).

(a) Fill in the values for ROOT, LEFT and RIGHT in Fig. 7.116(b) so that T will be stored in memory.



(a)

ROOT

AVAIL

5

	INFO	LEFT	RIGHT
1	A		
2	C		
3	D		
4	G		
5		6	
6		0	
7	H		
8	F		
9	E		
10	B		

(b)

Fig. 7.116

(b) Find (i) the depth D of T, (ii) the number of null subtrees and (iii) the descendants of node B.

7.2 List the nodes of the tree T in Fig. 7.116(a) in (a) preorder, (b) inorder and (c) postorder.

7.3 Draw the diagram of the tree T in Fig. 7.117.

7.4 Suppose the following sequences list the nodes of a binary tree T in preorder and inorder, respectively:

Preorder: G, B, Q, A, C, K, F, P, D, E, R, H
Inorder: Q, B, K, C, F, A, G, P, E, D, H, R

Draw the diagram of the tree.

7.5 Suppose a binary tree T is in memory and an ITEM of information is given.

(a) Write a procedure which finds the location LOC of ITEM in T (assuming the elements of T are distinct).

(b) Write a procedure which finds the location LOC of ITEM and the location PAR of the parent of ITEM in T.

ROOT
14

AVAIL
8

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

Fig. 7.117

(c) Write a procedure which finds the number NUM of times ITEM appears in T (assuming the elements of T are not necessarily distinct).

Remark: T is not necessarily a binary search tree.

7.6 Suppose a binary tree T is in memory. Write a nonrecursive procedure for each of the following:
(a) Finding the number of nodes in T.
(b) Finding the depth D of T.
(c) Finding the number of terminal nodes in T.

7.7 Suppose a binary tree T is in memory. Write a procedure which deletes all the terminal nodes in T.

7.8 Suppose ROOTA points to a binary tree T_1 in memory. Write a procedure which makes a copy T_2 of the tree T_1 using ROOTB as a pointer.

Binary Search Trees

7.9 Suppose the following eight numbers are inserted in order into an empty binary search tree T:

50, 33, 44, 22, 77, 35, 60, 40

Draw the tree T.

7.10 Consider the binary search tree T in Fig. 7.118. Draw the tree T if each of the following operations is applied to the original tree T. (That is, the operations are applied independently, not successively.)
(a) Node 20 is added to T.
(b) Node 15 is added to T.
(c) Node 88 is added to T.
(d) Node 22 is deleted from T.
(e) Node 25 is deleted from T.
(f) Node 75 is deleted from T.

7.11 Consider the binary search tree T in Fig. 7.117. Draw the final tree T if the six operations in Problem 7.10 are applied one after the other (not independently) to T.

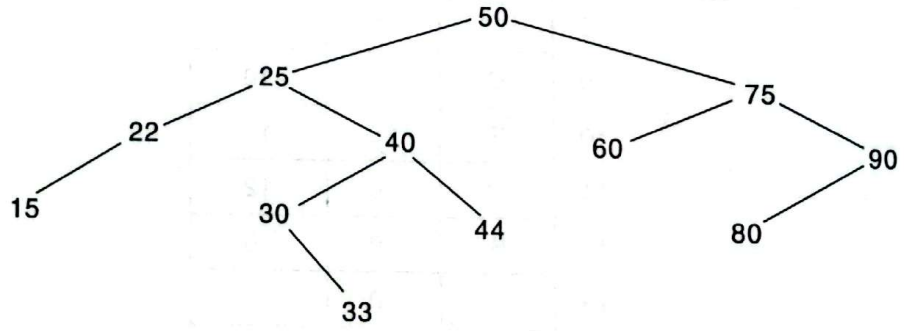


Fig. 7.118

7.12 Draw the binary search tree T in Fig. 7.119.

7.13 Consider the binary search tree T in Fig. 7.119. Describe the changes in INFO, LEFT, RIGHT, ROOT and AVAIL if each of the following operations is applied independently (not successively) to T.
(a) Davis is added to T.
(b) Harris is added to T.
(c) Smith is added to T.
(d) Parker is deleted from T.
(e) Fox is deleted from T.
(f) Murphy is deleted from T.

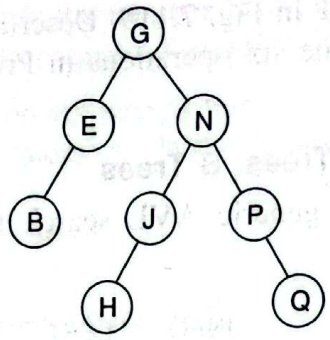


Fig. 7.121

7.17 Insert the following keys in the order shown below into an initially empty *m*-way search tree of order (i) 5 (ii) 4 (iii) 3

G S F L Q X Z V R A I J W

7.18 Define preorder traversal of an *m*-way search tree to be as follows:
Visit all the keys in the root node first followed by visiting all nodes in the subtrees beginning from left to right recursively in preorder.
List the keys in the preorder for the 4-way search tree constructed in Supplementary Problem 7.17.

7.19 A post order traversal of a B-tree of order *m* may be defined as below:
Recursively traverse all the subtrees of the root of the B-tree from the left to the right and traverse all keys in the root node.
Execute post order traversal for the B-tree of order 3 constructed in Solved Problem 7.17.

7.20 Are B-trees of order 2 full binary trees? If so, how?

7.21 Consider the binary tree T in Fig. 7.116(a).
(a) Draw the one-way inorder threading of T.
(b) Draw the one-way preorder threading of T.
(c) Draw the two-way inorder threading of T.
In each case, show how the threaded tree will appear in memory using the data in Fig. 7.116(b).

7.22 Consider the complete tree T with N = 10 nodes in Fig. 7.131. Suppose a maxheap is formed out of T by applying

Call INSHEAP(A, J, A[J + 1])

for J = 1, 2, …, N – 1. (Assume T is stored sequentially in the array A.) Find the final maxheap.

7.23 Repeat Supplementary Problem 7.22 for the tree T in Fig. 7.122, except now form a minheap out of T instead of a maxheap.

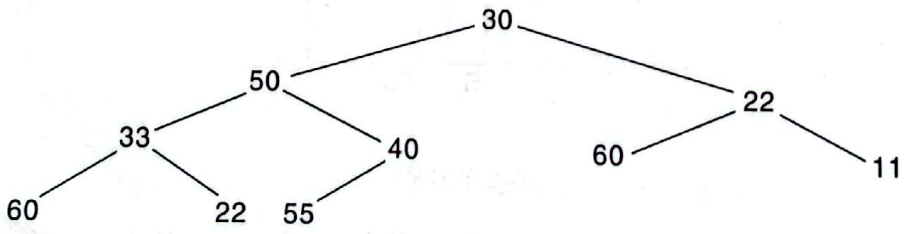


Fig. 7.122

7.24 Draw the 2-tree corresponding to each of the following algebraic expressions:
(a) $E_1 = (a - 3b)(2x - y)^3$
(b) $E_2 = (2a + 5b)^3(x - 7y)^4$

7.25 Consider the 2-tree in Fig. 7.123. Find the Huffman coding for the seven letters determined by the tree T.

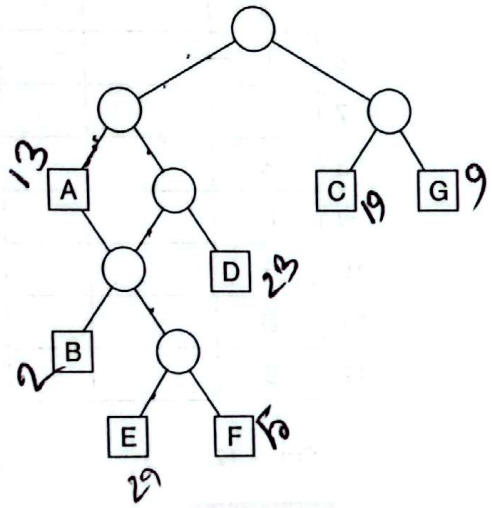


Fig. 7.123

7.26 Suppose the 7 data items A, B, ..., G are assigned the following weights:

(A, 13), (B, 2), (C, 19), (D, 23), (E, 29), (F, 5), (G, 9)

Find the weighted path length P of the tree in Fig. 7.123.

7.27 Using the data in Problem 7.6, find a 2-tree with a minimum weighted path length P. Wha is the Huffman coding for the 7 letters using this new tree?

7.28 Consider the forest F in Fig. 7.124, which consists of three trees with roots A, B and C respectively.

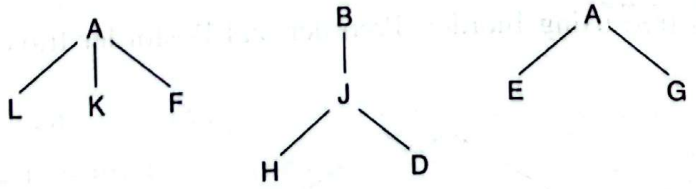


Fig. 7.124 Forest F

(a) Find the binary tree F′ corresponding to the forest F.
(b) Fill in values for ROOT, CHILD and SIB in Fig. 7.124 so that F will be stored i memory.

7.29 Suppose T is a complete tree with n nodes and depth D. Prove (a) $2^{D-1} - 1 < n \leq 2^D - 1$ an (b) $D \approx \log_2 n$.

ROOT

	INFO	CHILD	SIB
1	A		
2	C		
3	E		
4	G		
5	J		
6	L		
7			
8	K		
9	H		
10	F		
11	D		
12	B		

Fig. 7.125

Hint: Use the following identity with $x = 2$:

$$1 + x + x^2 + x^3 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

7.30 Suppose T is an extended binary tree. Prove:

(a) $N_E = N_I + 1$, where N_E is the number of external nodes and N_I is the number of internal nodes.

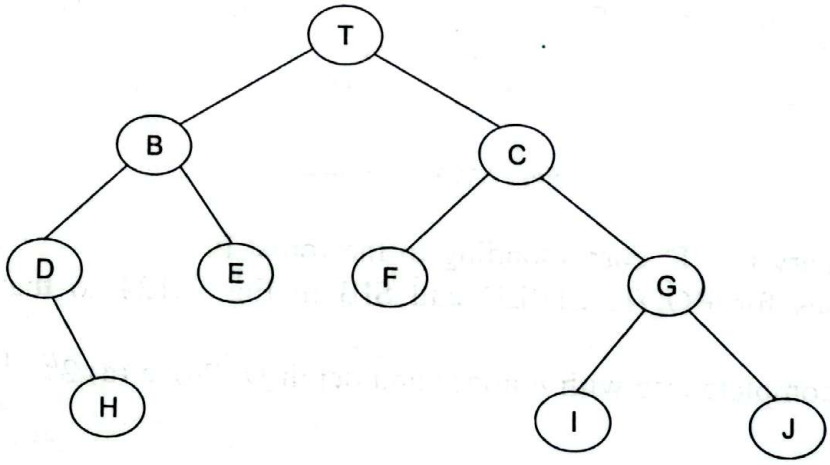
(b) $L_E = L_I + 2n$, where L_E is the external path length, L_I is the internal path length and n is the number of internal nodes.

7.31 You are given data in the sequence—91 24 6 7 11 8 21 4 5 16 19 20 76. Draw a B-tree of order 3 by inserting the data.

7.32 Draw a binary tree for the following expression:

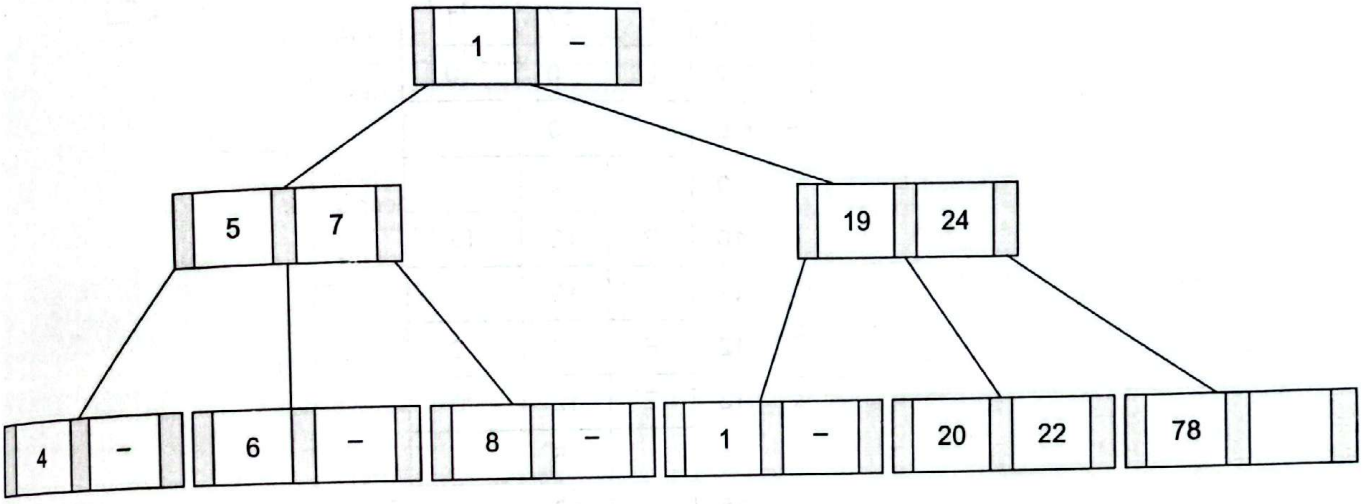
A * B – (C + D) * (P / Q)

7.33 Traverse the given tree using Inorder, Preorder and Postorder traversals.



Inorder: D H B E A F C I G J
Preorder: A B D H E C F G I J
Postorder: H D E B F I J G C A

7.34 Draw the B-tree of order 3 created by inserting the following data arriving in sequence—92 24 6 7 11 8 22 4 5 16 19 20 78



7.35 Suppose we traverse in a chess board right and down from one corner to another corner. Determine how many possible ways will be there.

7.36 Sort the following using the heap sort algorithm.

35 45 25 11 6 85 17 35

PROGRAMMING PROBLEMS

Programming Problems 7.1 to 7.3 refer to the tree T in Fig. 7.1, which is stored in memory as in Fig. 7.126.

7.1 Write a program which prints the nodes of T in (a) preorder, (b) inorder and (c) postorder.

7.2 Write a program which prints the terminal nodes of T in (a) preorder (b) inorder and (c) postorder.

(*Note:* All three lists should be the same.)

7.3 Write a program which makes a copy T' of T using ROOTB as a pointer. Test the program by printing the nodes of T' in preorder and inorder and comparing the lists with those obtained in Programming Problem 7.1.

Chapter 8

Graphs and Their Applications

8.5 Draw all (nonsimilar) trees with exactly 6 nodes. (A graph G is *similar* to a graph G' if there is a one-to-one correspondence between the set V of nodes of G and the set V' of nodes of G' such that (u, v) is an edge in G if and only if the corresponding pair (u', v') of nodes is an edge in G'.)

There are six such trees, which are exhibited in Fig. 8.31. The first tree has diameter 5, the next two diameter 4, the next two diameter 3 and the last one diameter 2. Any other tree with 6 nodes will be similar to one of these trees.

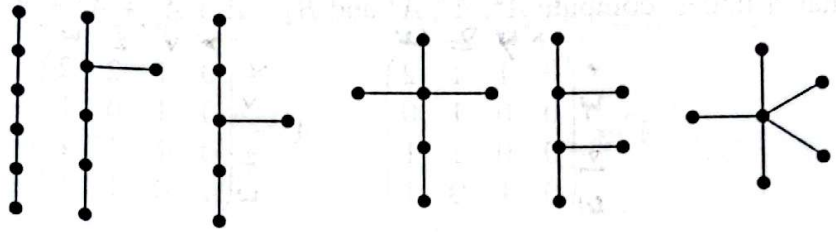


Fig. 8.31

8.6 Find all spanning trees of the graph G shown in Fig. 8.32(a). (A tree T is called a *spanning tree* of a connected graph G if T has the same nodes as G and all the edges of T are contained among the edges of G.)

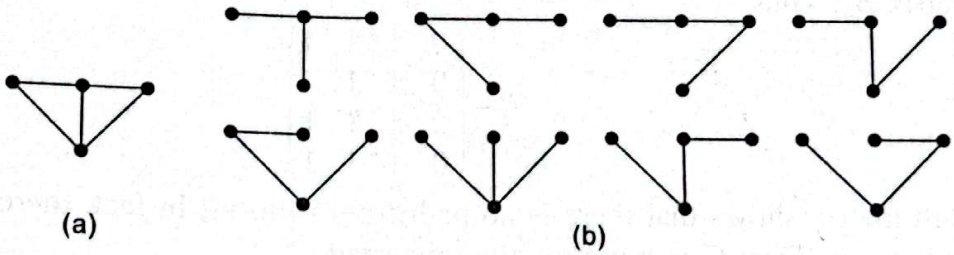


Fig. 8.32

There are eight such spanning trees, as shown in Fig. 8.32(b). Since G has 4 nodes, each spanning tree T must have $4 - 1 = 3$ edges. Thus each spanning tree can be obtained by deleting 2 of the 5 edges of G. This can be done in 10 ways, except that two of them lead to disconnected graphs. Hence the eight spanning trees shown are all the spanning trees of G.

Sequential Representation of Graphs

8.7 Consider the graph G in Fig. 8.30. Suppose the nodes are stored in memory in an array DATA as follows:

DATA: X, Y, Z, W

(a) Find the adjacency matrix A of the graph G.
(b) Find the path matrix P of G using powers of the adjacency matrix A.
(c) Is G strongly connected?

(a) The nodes are normally ordered according to the way they appear in memory; that is, we assume $v_1 = X$, $v_2 = Y$ $v_3 = Z$ and $v_4 = W$. The adjacency matrix A of G follows:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Here $a_{ij} = 1$ if there is a node from v_i to v_j otherwise, $a_{ij} = 0$.

(b) Since G has 4 nodes, compute A^2, A^3, A^4 and $B_4 = A + A^2 + A^3 + A^4$:

$$A^2 = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad A^3 = \begin{pmatrix} 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$A^4 = \begin{pmatrix} 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{pmatrix} \qquad B_4 = \begin{pmatrix} 0 & 5 & 6 & 8 \\ 0 & 1 & 2 & 3 \\ 0 & 3 & 3 & 5 \\ 0 & 2 & 3 & 5 \end{pmatrix}$$

The path matrix P is now obtained by setting $p_{ij} = 1$ wherever there is a nonzero entry in the matrix B_4. Thus

$$P = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

(c) The path matrix shows that there is no path from v_2 to v_1, In fact, there is no path from any node to v_1. Thus G is not strongly connected.

8.8 Consider the graph G in Fig. 8.30 and its adjacency matrix A obtained in Problem 8.7. Find the path matrix P of G using Warshall's algorithm rather than the powers of A.

Compute the matrices P_0, P_1, P_2, P_3 and P_4 where initially $P_0 = A$ and

$$P_k[i, j] = P_{k-1}[i, j] \vee (P_{k-1}[i, j] \wedge P_{k-1}[k, j])$$

That is,

$$P_k[i, j] = 1 \quad \text{if} \quad P_{k-1}[i, j] = 1 \quad \text{or both} \quad P_{k-1}[i, k] = 1 \quad \text{and} \quad P_{k-1}[k, j] = 1$$

Then:

$$P_1 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$P_3 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \qquad P_4 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

Observe that $P_0 = P_1 = P_2 = A$. The changes in P_3 occur for the following reasons:

$P_3(4, 2) = 1$ because $P_2(4, 3) = 1$ and $P_2(3, 2) = 1$

$P_3(4, 4) = 1$ because $P_2(4, 3) = 1$ and $P_2(3, 4) = 1$

The changes in P_4 occur similarly. The last matrix, P_4, is the required path matrix P of the graph G.

8.9 Consider the (undirected) weighted graph G in Fig. 8.33. Suppose the nodes are stored in memory in an array DATA as follows:

DATA: A, B, C, X, Y

Find the weight matrix $W = (w_{ij})$ of the graph G.

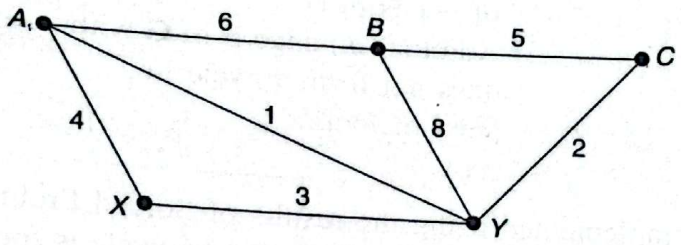


Fig. 8.33

Assuming $v_1 = A$, $v_2 = B$, $v_3 = C$, $v_4 = X$ and $v_5 = Y$, we arrive at the following weight matrix W of G:

$$W = \begin{array}{c} \\ A \\ B \\ C \\ X \\ Y \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & X & Y \end{array} \\ \begin{pmatrix} 0 & 6 & 0 & 4 & 1 \\ 6 & 0 & 5 & 0 & 8 \\ 0 & 5 & 0 & 0 & 2 \\ 4 & 0 & 0 & 0 & 3 \\ 1 & 8 & 2 & 3 & 0 \end{pmatrix} \end{array}$$

Here w_{ij} denotes the weight of the edge from v_i to v_j. Since G is undirected, W is a symmetric matrix, that is, $w_{ij} = w_{ji}$.

8.10 Suppose G is a graph (undirected) which is cycle-free, that is, without cycles. Let $P = (P_{ij})$ be the path matrix of G.

(a) When can an edge $[v_i, v_j]$ be added to G so that G is still cycle-free?

(b) How does the path matrix P change when an edge $[v_i, v_j]$ is added to G?

(a) The edge $[v_i, v_j$ will form a cycle when it is added to G if and only if there already is a path between v_i and v_j. Hence the edge may be added to G when $P_{ij} = 0$.

(b) First set $p_{ij} = 1$, since the edge is a path from v_i to v_j. Also, set $p_{st} = 1$ if $p_{si} = 1$ and $p_{jt} = 1$. In other words, if there are both a path P_1 from v_s to v_i and a path P_2 from v_j to v_t, then P_1, $[v_i, v_j]$. P_2 will form a path from v_s to v_t.

8.11 A minimum spanning tree T of a weighted graph G is a spanning tree of G (see Problem 8.6) which has the minimum weight among all the spanning trees of G.

(a) Describe an algorithm to find a minimum spanning tree T of a weighted graph G.

(b) Find a minimum spanning tree T of the graph in Fig. 8.33.

(a) **Algorithm P8.11:** This algorithm finds a minimum spanning tree T of a weighted graph G.

1. Order all the edges of G according to increasing weights.
2. Initialize T to be a graph consisting of the same nodes as G and no edges.
3. Repeat the following M – 1 times, where M is the number of nodes in G:
   Add to T an edge E of G with minimum weight such that E does not form a cycle in T.
   [End of loop.]
4. Exit.

Step 3 may be implemented using the results of Solved Problem. 8.10. Problem 8.10(a) tells us which edge e may be added to T so that no cycle is formed—i.e., so that T is still cycle-free—and Problem 8.10(b) tells us how to keep track of the path matrix P of T as each edge e is added to T.

(b) Apply Algorithm P8.11 to obtain the minimum spanning tree T in Fig. 8.34. Although [A, X] has less weight than [B, C), we cannot add [A, X] to T, since it would form a cycle with [A, Y] and [Y, X].

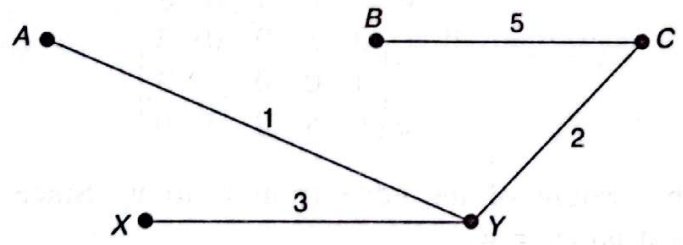


Fig. 8.34

8.12 Suppose a weighted graph G is maintained in memory by a node array DATA and a weight matrix W as follows:

DATA: X, Y, S, T

$$W = \begin{pmatrix} 0 & 0 & 3 & 0 \\ 5 & 0 & 1 & 7 \\ 2 & 0 & 0 & 4 \\ 0 & 6 & 8 & 0 \end{pmatrix}$$

Draw a picture of G.

The picture appears in Fig. 8.35. The nodes are labeled by the entries in DATA. Also, if $w_{ij} \neq 0$, then there is an edge from v_i to v_j with weight w_{ij}. (We assume $v_1 = X$, $v_2 = Y$, $v_3 = S$ and $v_4 = T$, the order in which the nodes appear in the array DATA.)

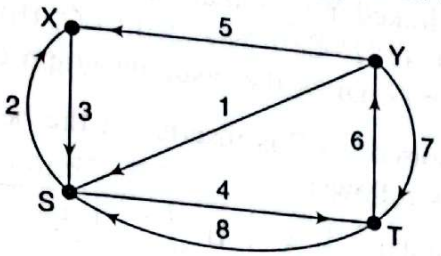


Fig. 8.35

Linked Representation of Graphs

8.13 A graph G is stored in memory as follows:

NODE	A	B		E		D	C	
NEXT	7	4	0	6	8	0	2	3
ADJ	1	2		5		7	9	
	1	2	3	4	5	6	7	8

START = 1, AVAILN = 5

DEST	2	6	4		6	7	4		4	6
LINK	10	3	6	0	0	0	0	4	0	0
	1	2	3	4	5	6	7	8	9	10

AVAILE = 8

Draw the graph G.

First find the neighbors of each NODE[K] by traversing its adjacency list, which has the pointer ADJ[K]. This yields:

A: 2(B) and 6(D)
B: 6(D), 4(E) and 7(C)
C: 4(E)
D: 4(E)
E: 6(D)

Then draw the diagram as in Fig. 8.36.

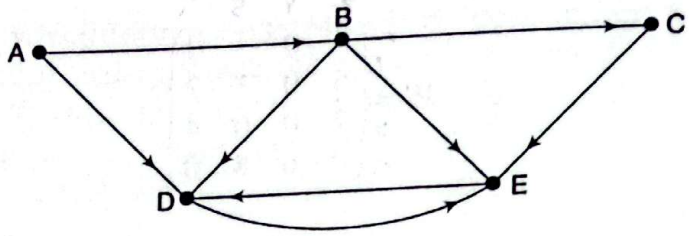


Fig. 8.36

8.14 Find the changes in the linked representation of the graph G in Problem 8.13 if the following operations occur: (a) Node F is added to G. (b) Edge (B, E) is deleted from G. (c) Edge (A, F) is added to G. Draw the resultant graph G.

(a) The node list is not sorted, so F is inserted at the beginning of the list, using the first available free node as follows:

START = 5
AVAILN = 8

	1	2	3	4	5	6	7	8
NODE	A	B		E	F	D	C	
NEXT	7	4	0	6	1	0	2	3
ADJ	1	2		5	0	7	9	

Observe that the edge list does not change.

(b) Delete LOC = 4 of node E from the adjacency list of node B as follows:

AVAILE = 3

	1	2	3	4	5	6	7	8	9	10
DEST	2	6			6	7	4		4	6
LINK	10	6	8	0	0	0	0	4	0	0

Observe that the node list does not change.

(c) The location LOC = 5 of the node F is inserted at the beginning of the adjacency list of the node A, using the first available free edge. The changes are as follows:

ADJ[1] = 3
AVAILE = 8

	1	2	3	4	5	6	7	8	9	10
DEST	2	6	5		6	7	4		4	6
LINK	10	6	1	0	0	0	0	4	0	0

The only change in the node list is the ADJ[1] = 3. (Observe that the shading indicates the changes in the lists.) The updated graph G appears in Fig. 8.37.

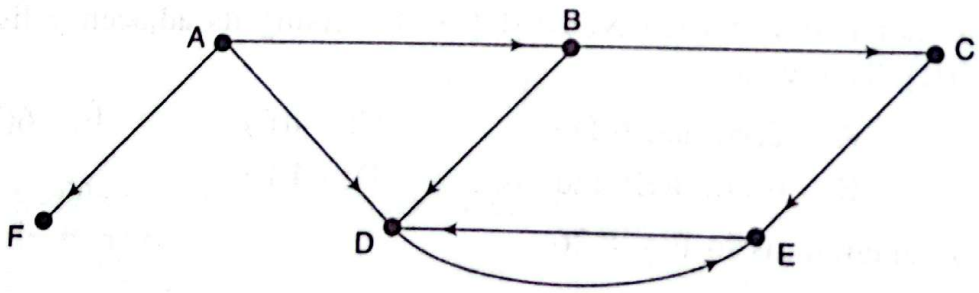


Fig. 8.37

SUPPLEMENTARY PROBLEMS

Graph Terminology

8.1 Consider the undirected graph G in Fig. 8.40. Find (a) all simple paths from node A to node H, (b) the diameter of G and (c) the degree of each node.

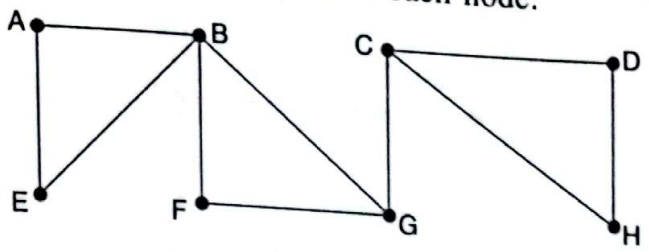


Fig. 8.40

8.2 Which of the multigraphs in Fig. 8.41 are (a) connected, (b) loop-free (i.e., without loops) and (c) graphs?

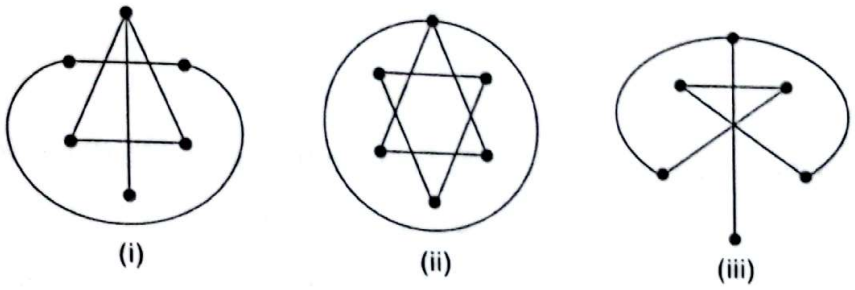


Fig. 8.41

8.3 Consider the directed graph G in Fig. 8.42. (a) Find the indegree and outdegree of each node. (b) Find the number of simple paths from v_1 to v_4. (c) Are there any sources or sinks?

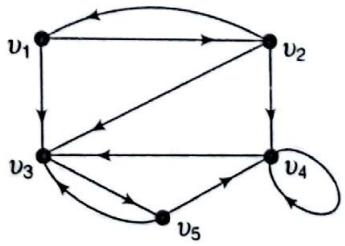


Fig. 8.42

8.4 Draw all (nonsimilar) trees with 5 or fewer nodes. (There are eight such trees.)

8.5 Find the number of spanning trees of the graph G in Fig. 8.43.

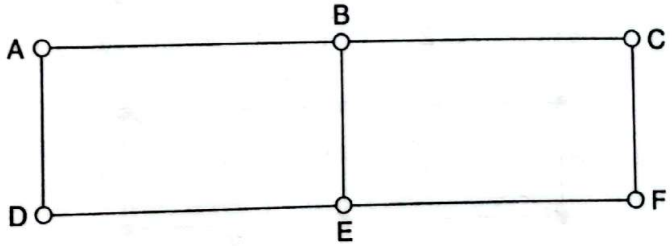


Fig. 8.43

Sequential Representation of Graphs; Weighted Graphs

8.6 Consider the graph G in Fig. 8.44. Suppose the nodes are stored in memory in an array DATA as follows:

DATA: X, Y, Z, S, T

(a) Find the adjacency matrix A of G. (b) Find the path matrix P or G. (c) Is G strongly connected?

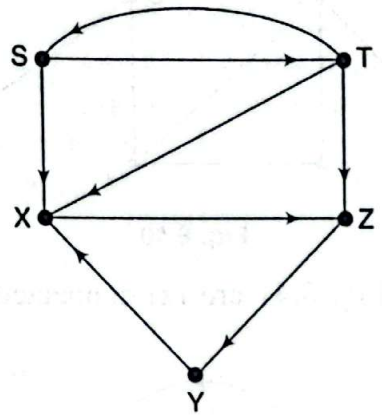


Fig. 8.44

8.7 Consider the weighted graph G in Fig. 8.45. Suppose the nodes are stored in an array DATA as follows:

DATA: X, Y, S, T

(a) Find the weight matrix W of G. (b) Find the matrix Q of shortest paths using Warshall's Algorithm 8.2.

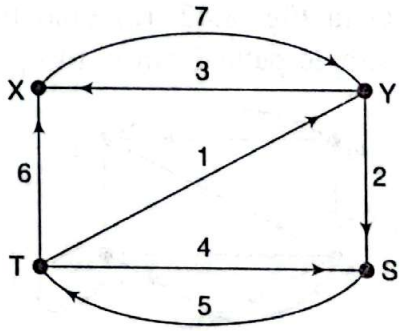


Fig. 8.45

8.8 Find a minimum spanning tree of the graph G in Fig. 8.46.

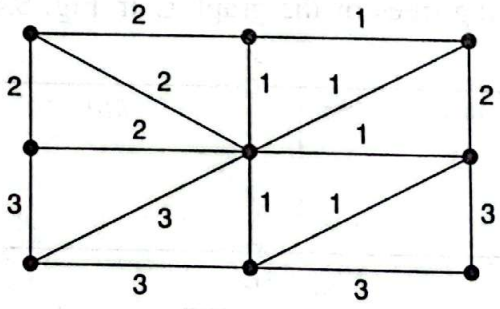


Fig. 8.46

8.9 The following is the incidence matrix M of an undirected graph G:

$$M = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

(Note that G has 5 nodes and 8 edges.) Draw G and find its adjacency matrix A.

8.10 The following is the adjacency matrix A of an undirected graph G:

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

(Note that G has 5 nodes.) Draw G and find its incidence matrix M.

Linked Representation of Graphs

8.11 Suppose a graph G is stored in memory as follows:

NODE	A		C	E		D		B
NEXT	4	0	8	0	7	3	2	1
ADJ	6		1	10		2		9
	1	2	3	4	5	6	7	8

START = 6, AVAILN = 5

DEST	8	8		1	4	3	3		6	3
LINK	5	7	8	0	0	0	0	0	4	0
	1	2	3	4	5	6	7	8	9	10

AVAILE = 3

Draw the graph G.

8.12 Find the changes in the linked representation of the graph G in Supplementary Problem 8.11 if edge (C, E) is deleted and edge (D, E) is inserted.

8.13 Find the changes in the linked representation of the graph G in Supplementary Problem 8.11 if a node F and the edges (E, F) and (F, D) are inserted into G.

8.14 Find the changes in the linked representation of the graph G in Supplementary Problem 8.11 if the node B is deleted from G.

Supplementary Problems 8.15 to 8.18 refer to a graph G which is maintained in memory by a linked representation:

GRAPH(NODE, NEXT, ADJ, START, AVAILN, DEST, LINK, AVAILE)

8.15 Write a procedure to supplement each of the following:

(a) Print the list of successors of a given node ND.

(b) Print the list of predecessors of a given node ND.

Chapter 9

Sorting and Searching

SUPPLEMENTARY PROBLEMS

Sorting

9.1 Write a subprogram RANDOM(DATA, N, K) which assigns N random integers between 1 and K to the array DATA.

9.2 Translate insertion sort into a subprogram INSERTSORT(A, N) which sorts the array A with N elements. Test the program using:
(a) 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66
(b) D, A, T, A, S, T, R, U, C, T, U, R, E, S

9.3 Translate insertion sort into a subprogram INSERTCOUNT(A, N, NUMB) which sorts the array A with N elements and which also counts the number NUMB of comparisons.

9.4 Write a program TESTINSERT(N, AVE) which repeats 500 times the procedure INSERTCOUNT(A, N, NUMB) and which finds the average AVE of the 500 values of NUMB. (Theoretically, AVE $\approx N^2/4$.) Use RANDOM(A, N, 5*N) from Problem 9.1 as each input. Test the program using N = 100 (so, theoretically, AVE $\approx N^2/4 = 2500$).

9.5 Translate quicksort into a subprogram QUICKCOUNT(A, N, NUMB) which sorts the array A with N elements and which also counts the number NUMB of comparisons. (See Sec. 6.5.)

9.6 Write a program TESTQUICKSORT(N, AVE) which repeats QUICKCOUNT(A, N, NUMB) 500 times and which finds the average AVE of the 500 values of NUMB. (Theoretically, AVE $\approx N \log_2 N$.) Use RANDOM(A, N, 5*N) from Problem 9.1 as each input. Test the program using N = 100 (so, theoretically, AVE ≈ 700).

9.7 Translate Procedure 9.2 into a subprogram MIN(A, LB, UB, LOC) which finds the location LOC of the smallest elements among A[LB], A[LB + 1], ..., A[UB].

9.8 Translate selection sort into a subprogram SELECTSORT(A, N) which sorts the array with N elements. Test the program using:
(a) 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66
(b) D, A, T, A, S, T, R, U, C, T, U, R, E, S

Searching, Hashing

9.9 Suppose an unsorted linked list is in memory. Write a procedure

SEARCH(INFO, LINK, START, ITEM, LOC)

which (a) finds the location LOC of ITEM in the list or sets LOC := NULL for an unsuccessful search and (b) when the search is successful, interchanges ITEM with the element in front of it. (Such a list is said to be *self-organizing*. It has the property that elements which are frequently accessed tend to move to the beginning of the list.)

9.10 Consider the following 4-digit employee numbers (see Example 9.10):

9614, 5882, 6713, 4409, 1825

Find the 2-digit hash address of each number using (a) the division method, with $m = 97$; (b) the midsquare method; (c) the folding method without reversing; and (d) the folding method with reversing.