Data Structure

تركيب بيانات
الفرقة الثالثة علوم إحصاء وعلوم الحاسب

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

رقم المحاضرة

رقم المحاضرة	التاريخ
المحاضرة 1	2020-02
المحاضرة 2	2020-03-03
المحاضرة 3	2020-03-10
المحاضرة 4	2020-03-17
المحاضرة 5	2020-03-24
المحاضرة 6	
المحاضرة 7	
المحاضرة 8	
المحاضرة 9	
المحاضرة 10	
المحاضرة 11	

Chapter 5:
Linked Lists Part I

Chapter 5: Linked Lists

5.1. Introduction

5.2. Linked Lists

1. Example 5.1

5.3. Representation of Linked Lists in Memory

1. Example 5.2 to 5.5

5.4. Traversing a linked list

1. Algorithm 5.1:
2. Example 5.6 and 5.7

Chapter 5: Linked Lists

5.5. Searching a linked list

1. Algorithm 5.2
2. Example 5.8
3. Algorithm 5.3
4. Example 5.9

5.6. Memory Allocation; Garbage Collection

1. Example 5.10 to 5.12
2. Garbage Collection
3. Overflow and underflow

Linked List outline

- Why linked lists
- Linked lists
- Representation of Linked lists in memory
- Traversing a linked lists
- Memory allocation: Garbage collection
- Overflow and Underflow
- Basic operations of linked lists
  - Insert, find, delete, print, etc.
- Variations of linked lists
  - Circular linked lists
  - Doubly linked lists

5.1 Introduction

5.1 Introduction

- Data processing frequently involves storing and processing data organized into lists.
- One way to store such data is by means of arrays.
- **Arrays have certain disadvantages**. It is relatively expensive **to insert and delete elements** in an array.

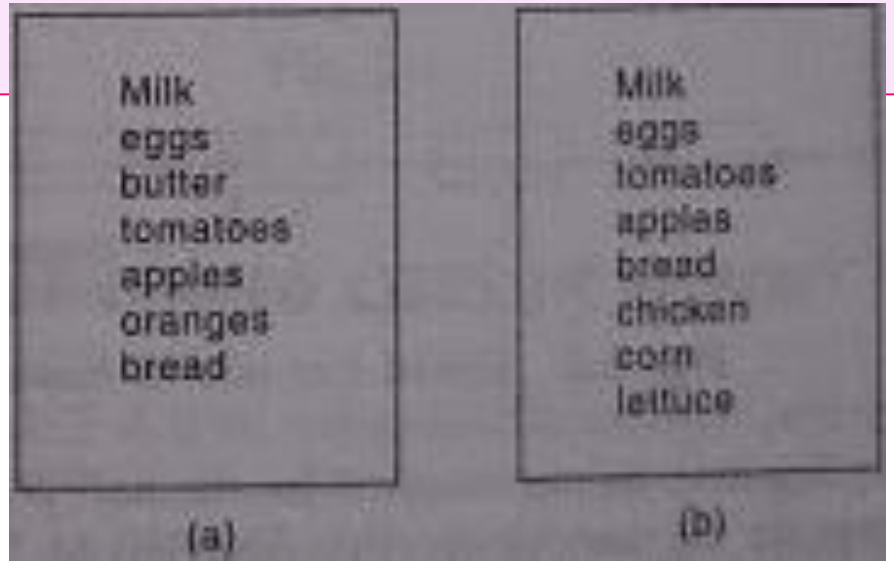


Fig. 5-1

Disadvantages of using Arrays

- Since an array occupies a block of memory space, one can't simply change the size of an array when an additional space is required.
- For this reason, arrays are called dense lists and are said to be static data structures.

Using Linked Lists

- Another way of storing a list in memory is to have each element in the list contain a field, called a link or pointer which contains the address of the next element in the list.
- Thus successive elements in the list need not occupy adjacent space in memory.
- This will make it easier to insert and delete elements in the list.

Using Linked Lists

- If one were mainly interested in searching through data for inserting and deleting, one would not store the data in an array but rather in a list using pointers.
- This later type of data structure is called a linked list which is the subject of this chapter.

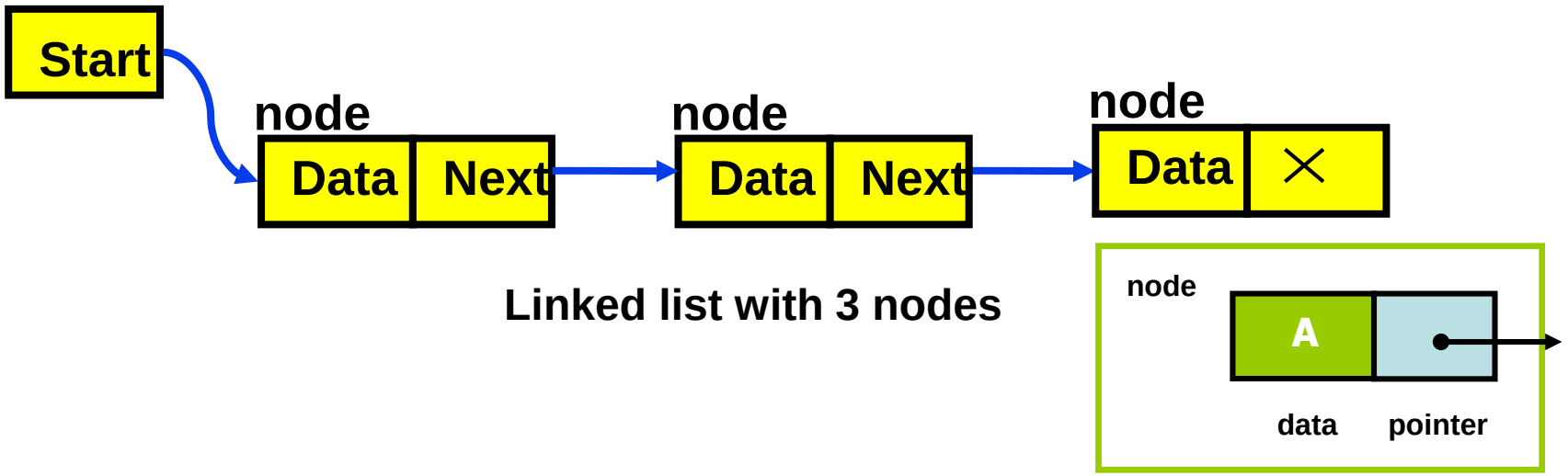
5.1 Introduction: Why linked lists

- Disadvantages of arrays as storage data structures:
  - slow insertion in ordered array
  - Fixed size
- Linked lists solve some of these problems
- Linked lists are general purpose storage data structures.

5.2 Linked List

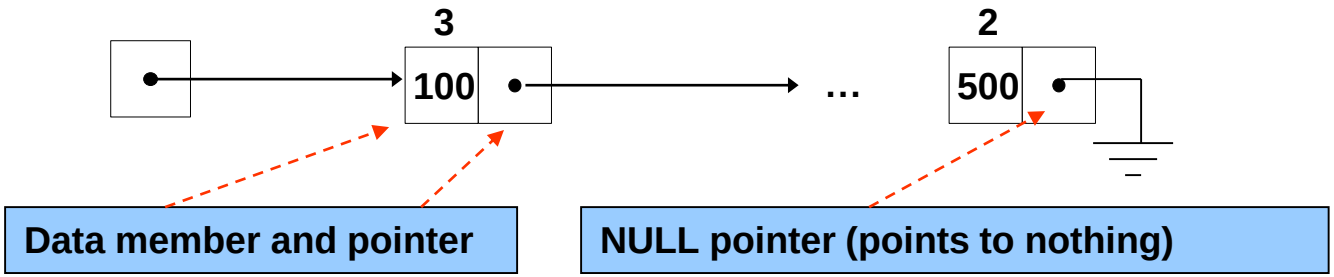
5.2 Linked List

- A linked list, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers.
- Each node is divided into two parts:
- The first part contains the information of the element, and
- The second part, called the link field or next pointer field, contains the address of the next node in the list see fig 5.2 and the next figure.
- The pointer of the last node contains a special value, called the null pointer.
- A pointer variable – called START which contains the address of the first node.
- A special case is the list that has no nodes, such a list is called the null list or empty list and is denoted by the null pointer in the variable START.



Linked list with 3 nodes

5.2 Linked List



- A schematic diagram of a linked list with 2 nodes,

5.2 Linked List

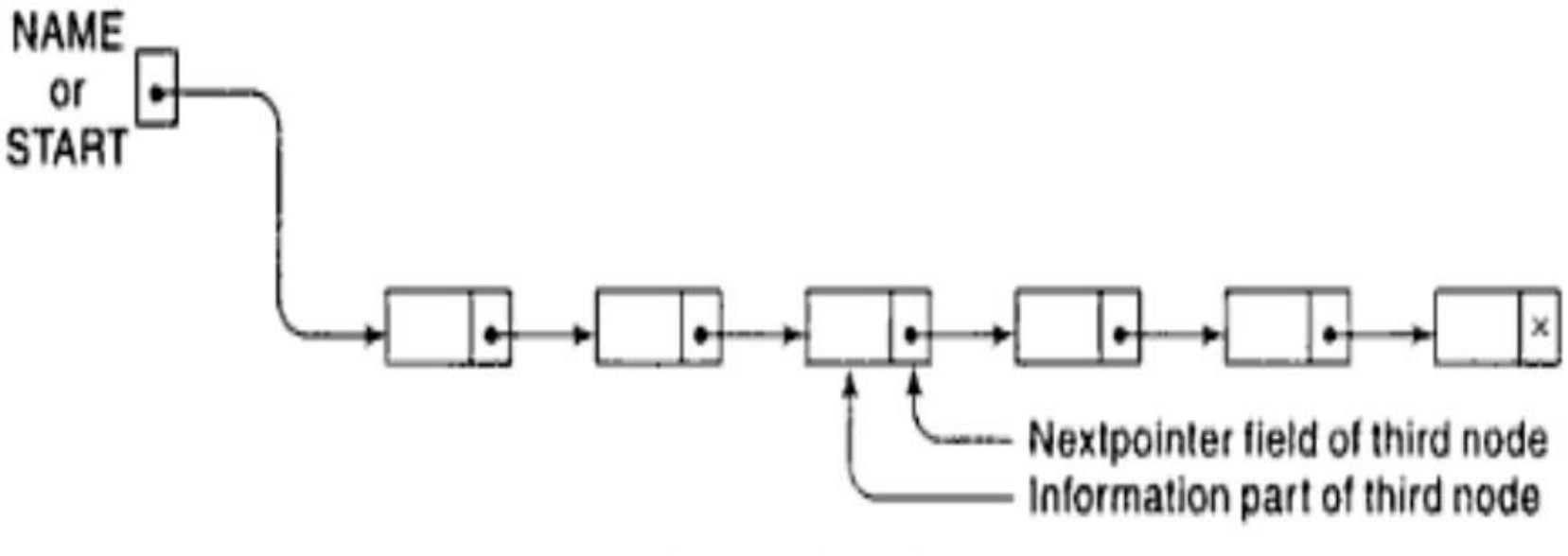


Fig. 5-2 Linked List with 6 Nodes

5.2 Linked List

- In actual practice, 0 or a negative number is used for the null pointer.
- The null pointer, denoted by x in the diagram, signals the end of the list.
- The linked list also contains a list pointer variable-called START or NAME which contains the address of the first node in the list; hence there is an arrow drawn from START to the first node.
- Clearly, we need only this address in START to trace through the list.

Array versus Linked Lists

- **Linked lists** are more complex to code and manage than arrays, but they have some distinct advantages.

- **Dynamic:** a linked list can easily grow and shrink in size.
  - We don't need to know how many nodes will be in the list. They are created in memory as needed.
  - In contrast, the size of a C++ array is fixed at compilation time.

- **Easy and fast insertions and deletions**
  - To insert or delete an element in an array, we need to copy to temporary variables to make room for new elements or close the gap caused by deleted elements.
  - With a linked list, no need to move other nodes. Only need to reset some pointers.

5.2 Linked Lists

- **A linked list organizes a collection of data items (elements) such that elements can easily be added to and deleted from any position in the list.**
- **Only references to next elements are updated in insertion and deletion operations.**
- **There is no need to copy or move large blocks of data to facilitate insertion and deletion of elements.**
- **Lists grow dynamically.**

EXAMPLE 5.1

A hospital ward contains 12 beds, of which 9 are occupied as shown in Fig. 5-3. Suppose we want an alphabetical listing of the patients.

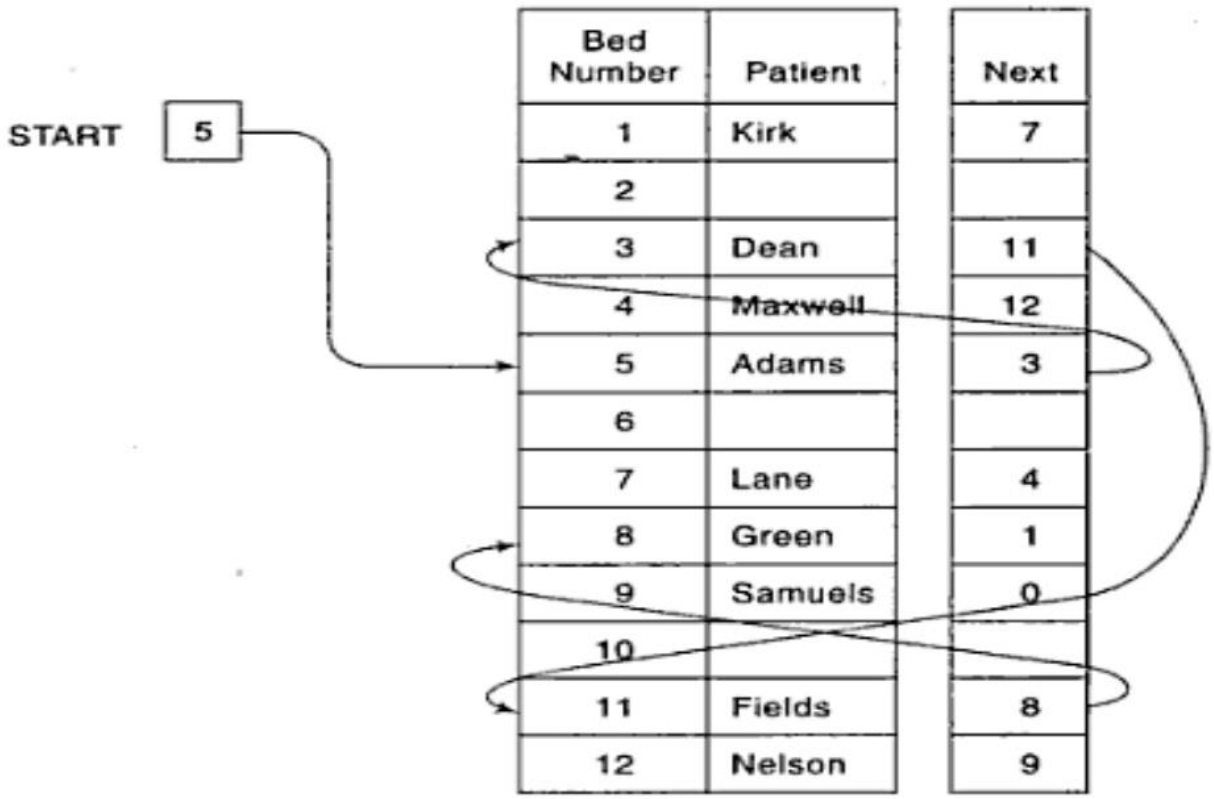


Fig. 5-3

EXAMPLE 5.1

- This listing may be given by the pointer field, called Next in the figure.
- We use the variable START to point to the first patient.
- Hence START contains 5, since the first patient, Adams, occupies bed 5.
- Also, Adams's pointer is equal to 3, since Dean, the next patient, occupies bed 3; Dean's pointer is 11, since Fields, the next patient, occupies bed 11; and so on.
- The entry for the last patient (Samuels) contains the null pointer, denoted by O. (Some arrows have been drawn to indicate the listing of the first few patients.)

5.3 Representation Of Linked Lists In Memory

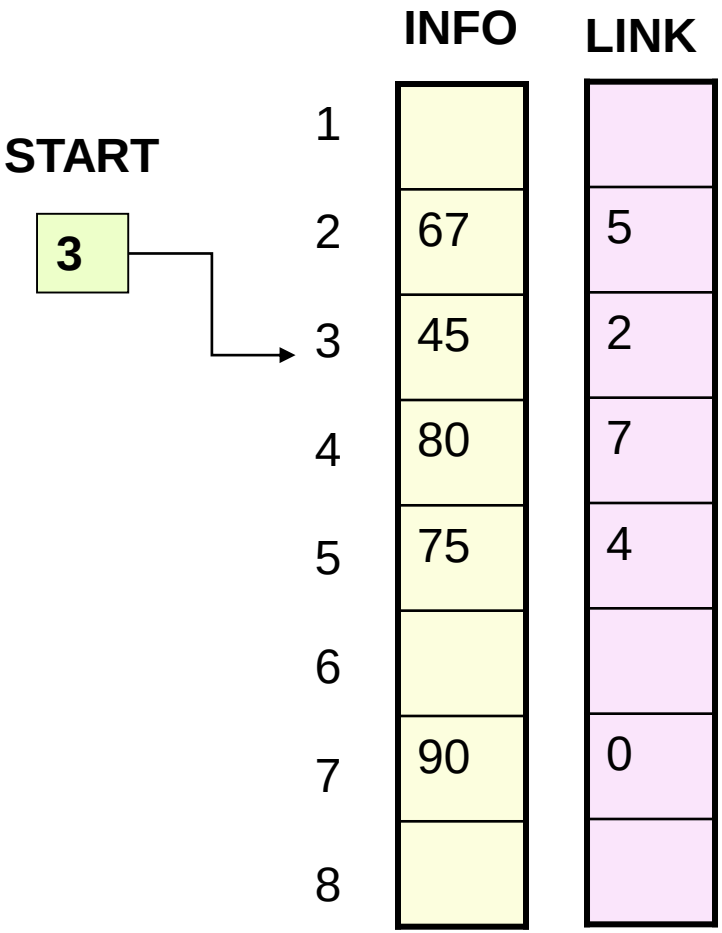
5.3 REPRESENTATION OF LINKED LISTS IN MEMORY

- Let LIST be a linked list.
- Then LIST will be maintained in memory, unless otherwise specified or implied, as follows.
- First of all, LIST requires two linear arrays
- we will call them here INFO and LINK such that INFO[K] and LINK[K] contain, respectively, the information part and the next pointer field of a node of LIST.
- As noted above, LIST also requires ,a variable name- such as ST ART which contains the location of the beginning of the list,
- and a next pointer sentinel-denoted by NULL-which indicates the end of the list.

5.3 REPRESENTATION OF LINKED LISTS IN MEMORY

- Since the subscripts of the arrays INFO and LINK will usually be positive, we will choose NULL = 0, unless otherwise stated.
- The following examples of linked lists indicate that the nodes of a list need not occupy adjacent elements in the arrays INFO and LINK, and that more than one list may be maintained in the same linear arrays INFO and LINK.
- However, each list must have its own pointer variable giving the location of its first node.

Example



START=3, INFO[3]=45

LINK[3]=2, INFO[2]=67

LINK[2]=5, INFO[5]=75

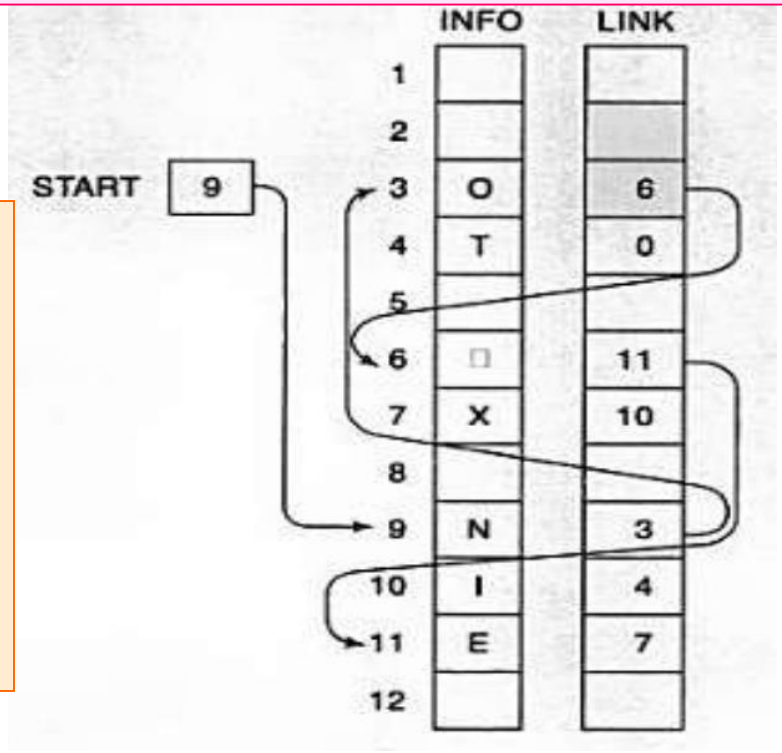
LINK[5]=4, INFO[4]=80

LINK[4]=7, INFO[7]=90

LINK[7]=0, NULL value, So the list has ended

EXAMPLE 5.2

- Figure 5-4 pictures a linked list in memory where each node of the list contains a single character.
- We can obtain the actual list of characters, or, in other words, the string, as follows:



START=9, INFO[9]=N is the first character.
LINK[9]=3, INFO[3]=0 is the second character.
LINK[3]=6, INFO[6]= (blank) is third character.
LINK[6]=11, INFO[I1]=E is the fourth character.
LINK[I1]=7, INFO[7] =X is the fifth character.
LINK(7] = 10, INFO[1O] = I is sixth character.
LINK[1O] = 4, so INFO[4] = T is seventh character.
LINK[4] = 0, the NULL! value, so the list has ended.

Fig. 5-4

EXAMPLE 5.3

- Figure 5-5 pictures how two lists of test scores, here ALG and GEOM, may be maintained in memory where the nodes of both lists are stored in the same linear arrays TEST and LINK.
- Observe that the names of the lists are also used as the list pointer variables.
- Here ALG contains 11, the location of its first node, and GEOM contains 5, the location of its first node.

EXAMPLE 5.3

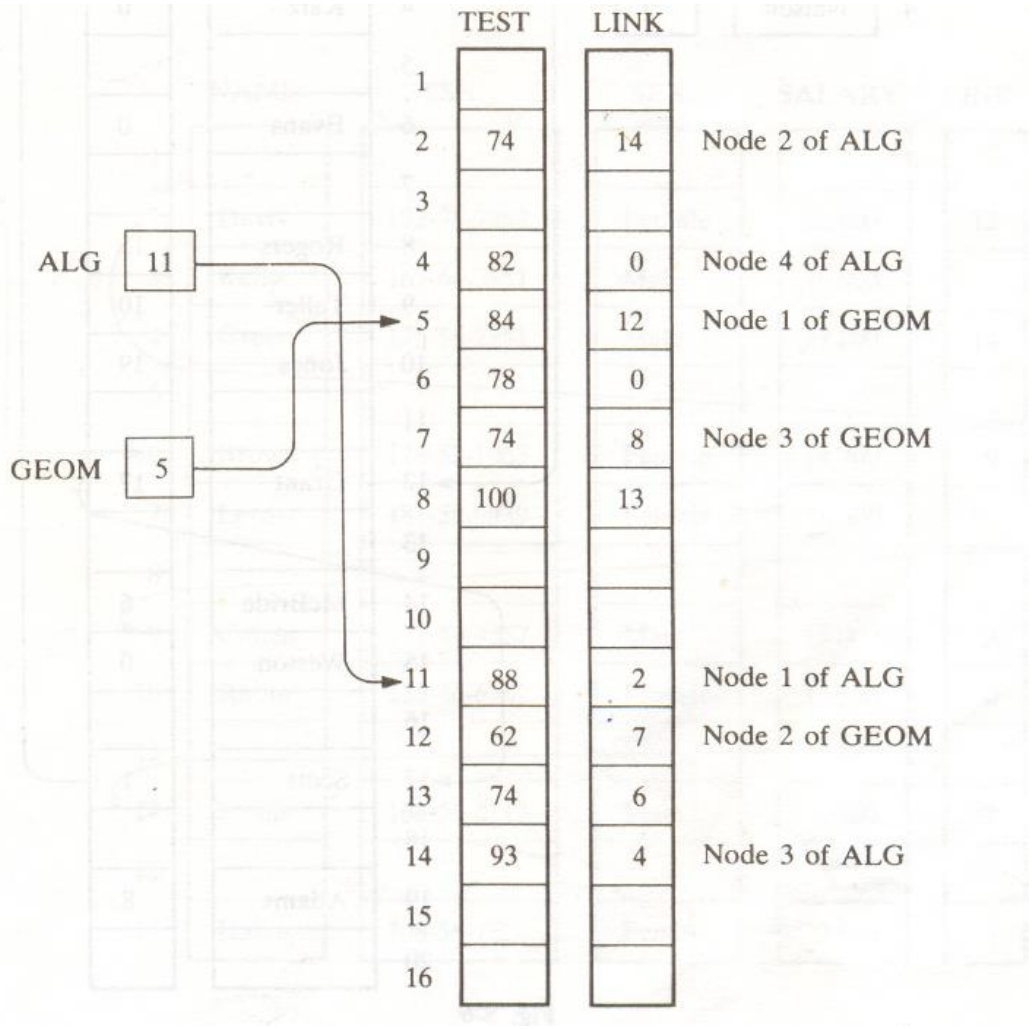


Fig. 5-5

EXAMPLE 5.3

- Following the pointers, we see that ALG consists of the test scores

  88, 74, 93, 82

  and GEOM consists of the test scores

  84, 62, 74, 100, 74, 78

  (The nodes of ALG and some of the nodes of GEOM are explicitly labeled in the diagram.)

EXAMPLE 5.4

- Suppose a brokerage firm شركة الوساطة المالية has four brokers السماسرة and each broker has his own list of customers. Such data may be. organized as in Fig.5-6.
- That is, all four lists of customers appear in the same array CUSTOMER, and an array LINK contains the next pointer fields of the nodes of the lists.
- There is also an array BROKER which contains the list of brokers, and a pointer array POINT such that POINT[K) points to the beginning of the list of customers of BROKER[K).

EXAMPLE 5.4

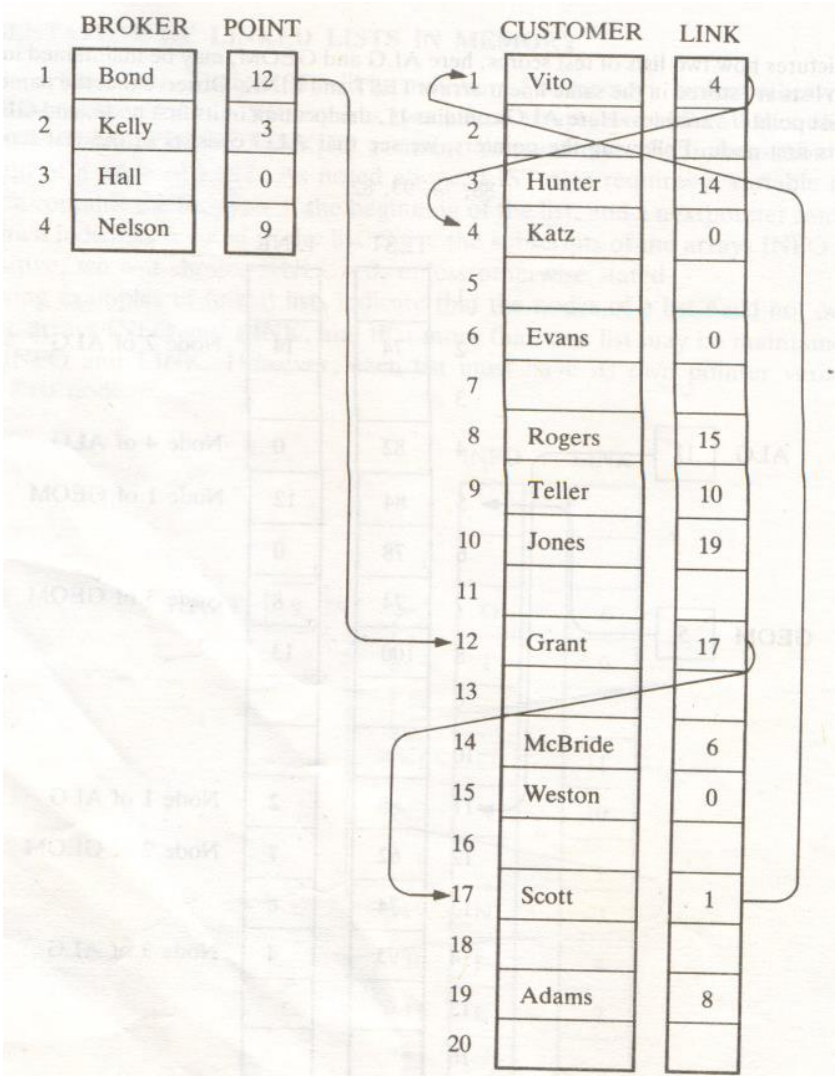


Fig. 5-6

EXAMPLE 5.4

- Accordingly, Bond's list of customers, as indicated by the arrows, consists of

  Grant, Scott, Vito, Katz

- Similarly, Kelly's list consists of

  Hunter, McBride, Evansand

- Nelson's list consists of

  Teller, Jones, Adams, Rogers, Weston

- Hall's list is the null list, since the null pointer 0 appears in POINT[3].

EXAMPLE 5.5

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

Name, Social Security Number, Sex, Monthly Salary

- Normally, four parallel arrays, say **NAME, SSN, SEX, SALARY**, are required to store the data as discussed in Sec. 4.12.
- Figure 5-7 shows how the data may be stored as a sorted (alphabetically) linked list using only an additional array LINK for the next pointer field of the list and the variable START to point to the first record in the list.
- Observe that 0 is used as the null pointer.

EXAMPLE 5.5

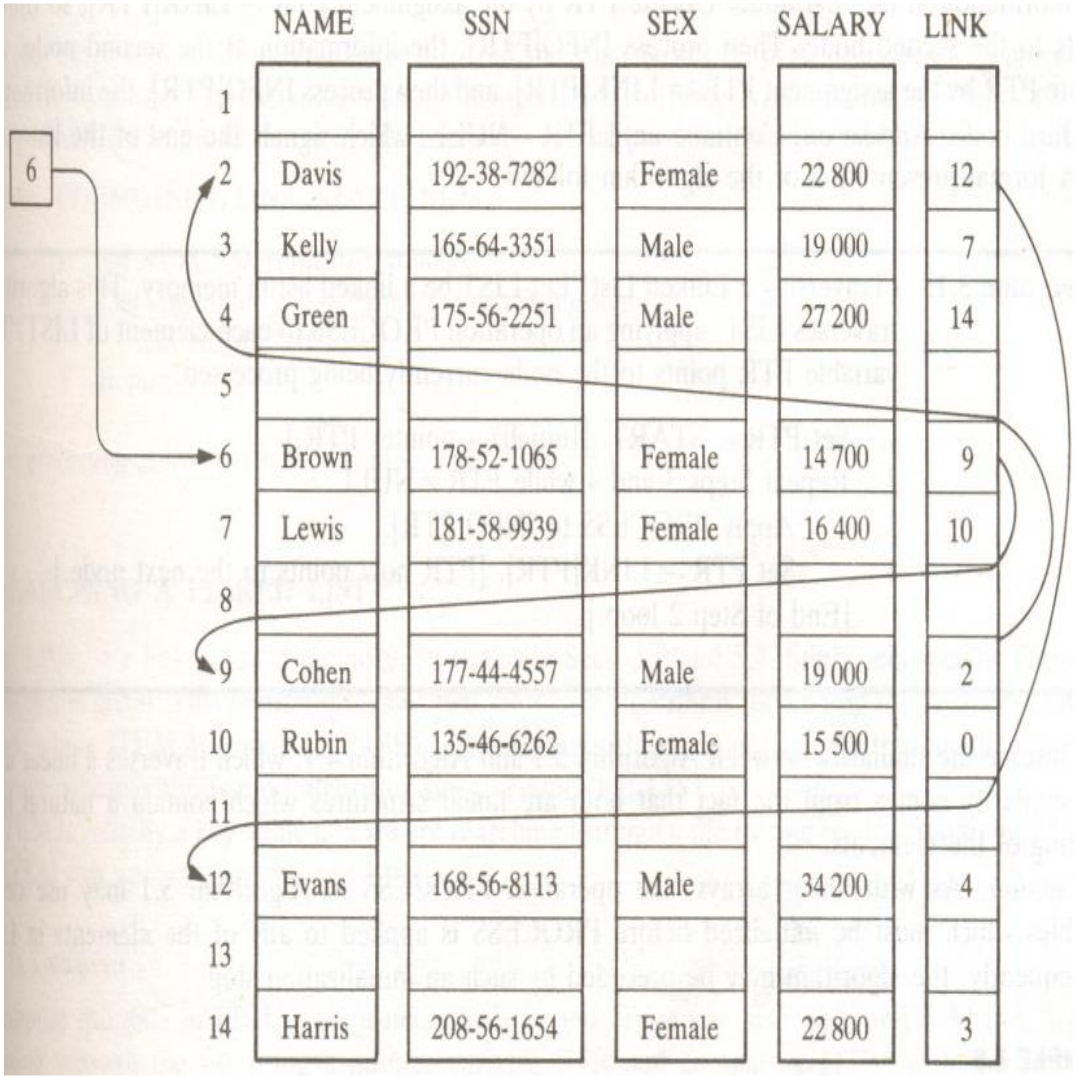


Fig. 5-7

Problem

Find the character strings stored in the four linked lists in Fig. 5-39.

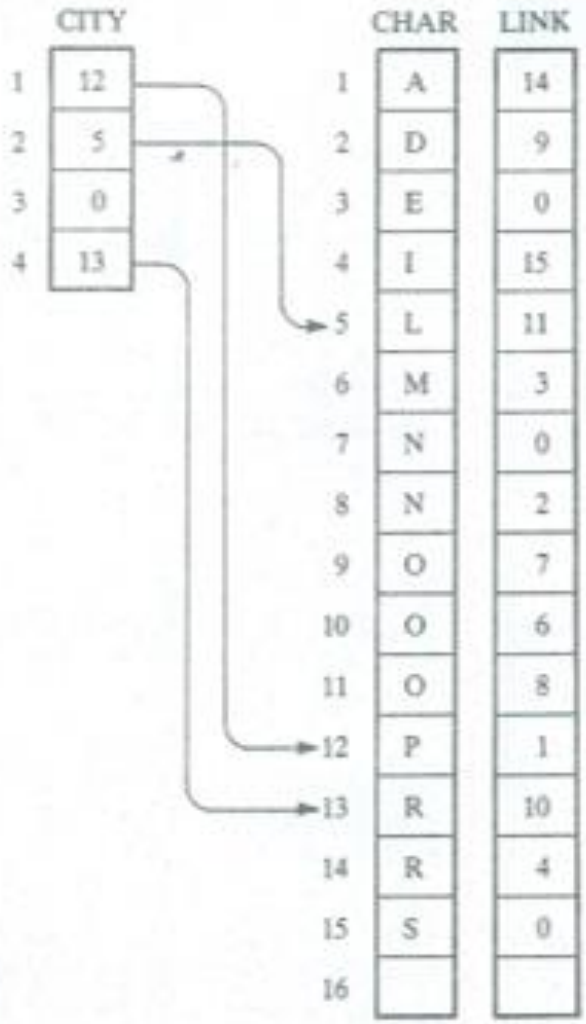


Fig. 5-39

Problem

The following list of names is assigned (in order) to a linear array INFO:

Mary, June, Barbara, Paula, Diana, Audrey, Karen, Nancy, Ruth, Eileen, Sandra, Helen

That is, INFO[1] = Mary, INFO[2] = June, . . . , INFO[12] = Helen. Assign values to an array LINK and a variable START so that INFO, LINK and START form an alphabetical listing of the names.

Problem

Figure 5-47 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. 5-47

5.4 Traversing a linked list

5.4 Traversing a linked lists

- Let LIST be a linked list in memory stored in linear arrays INFO and LINK with START pointing to the first element and NULL indicating the end of LIST.
- We want to traverse LIST in order to process each node exactly once.
- Pointer variable PTR points to the node that is currently being processed.
- LINK[PTR] points to the next node to be processed.
- Thus update PTR by the assignment

  PTR : =LINK[PTR]

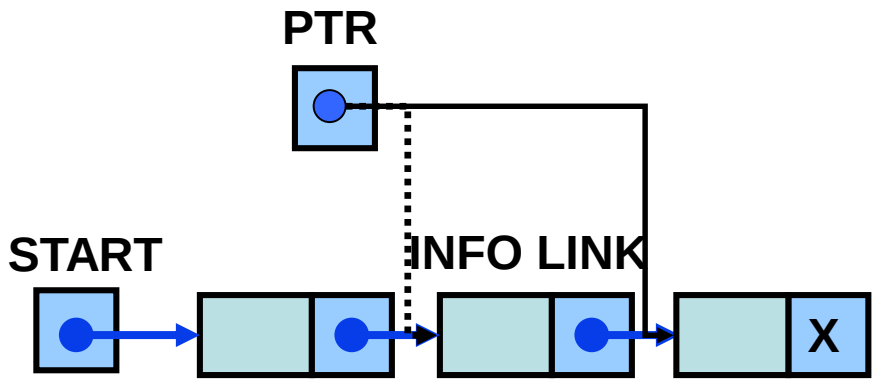


Fig : PTR : = LINK[PTR]

5.4 Traversing a linked lists

PTR : = LINK[PTR]

- moves the pointer to the next node in the list, as pictured in Fig.

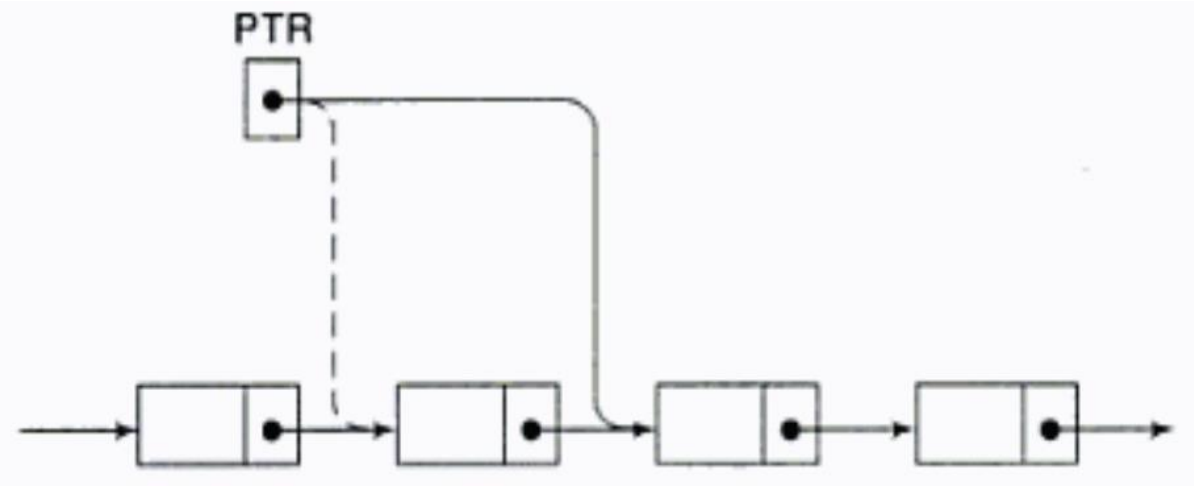


Fig. 5-8 PTR:=LINK[PTR]

Algorithm 5.1

- Algorithm 5.1: (Traversing a Linked List) Let LIST be a linked list in memory. This algorithm traverses LIST, applying an operation PROCESS to each element of LIST. The variable PTR points to the node currently being processed.

1. Set PTR:= START. [Initializes pointer PTR.]
2. Repeat Steps 3 and 4 while PTR != NULL.
3. Apply PROCESS to INFO[PTR].
4. Set PTR: = LINK[PTR]. [PTR now points to the next node.]

   [End of Step 2 loop.]
5. Exit.

5.4 Traversing a linked lists

- Initialize PTR or START.
- Then process INFO[PTR], the information at the first node.
- Update PTR by the assignment
- PTR: = LINK[PTR], so that PTR points to the second node. Then process INFO[PTR], the information at the second node.
- Again update PTR by the assignment PTR:= LINK[PTR], and then process INFO[PTR], the information at the third node.
- And so on.
- Continue until PTR = NULL, which signals the end of the list.

Example 5.6

- The following procedure prints the information at each node of a linked list. Since the procedure must traverse the list, it will be very similar to Algorithm 5.1.
- In other words, the procedure may be obtained by simply substituting the statement Write: INFO[PTR] for the processing step in Algorithm 5.1.

Example 5.6

For printing the information at each node of a linked list, must traverse the list.

Procedure 5.1 : PRINT(INFO, LINK, START)

Algorithm Prints the information at each node of the list.

1. **Set PTR : =START.**
2. **Repeat steps 3 and 4 while PTR : ≠ NULL:**
3. **Write : INFO[PTR].**
4. **Set PTR : =LINK[PTR].**
5. **Exit.**

Example 5.7

For Finding the number NUM of elements in a linked list, must traverse the list.

Procedure : COUNT(INFO, LINK, START, NUM)

1. **Set NUM: =0.**
2. **. Set PTR : =START.**
3. **Repeat steps 4 and 5 while PTR : ≠ NULL:**
4. **Set NUM : =NUM+1.**
5. **Set PTR : =LINK[PTR].**
6. **Exit.**

Problem

Let LIST be a linked list in memory. Write a procedure which

(*a*) Finds the number NUM of times a given ITEM occurs in LIST

(*b*) Finds the number NUM of nonzero elements in LIST

(*c*) Adds a given value K to each element in LIST

Each procedure uses Algorithm 5.1 to traverse the list.

Problem

Suppose LIST is a linked list in memory consisting of numerical values. Write a procedure for each of the following:

(*a*) Finding the maximum MAX of the values in LIST

(*b*) Finding the average MEAN of the values in LIST

(*c*) Finding the product PROD of the elements in LIST

d) Finding the Variance VAR of the values in LIST

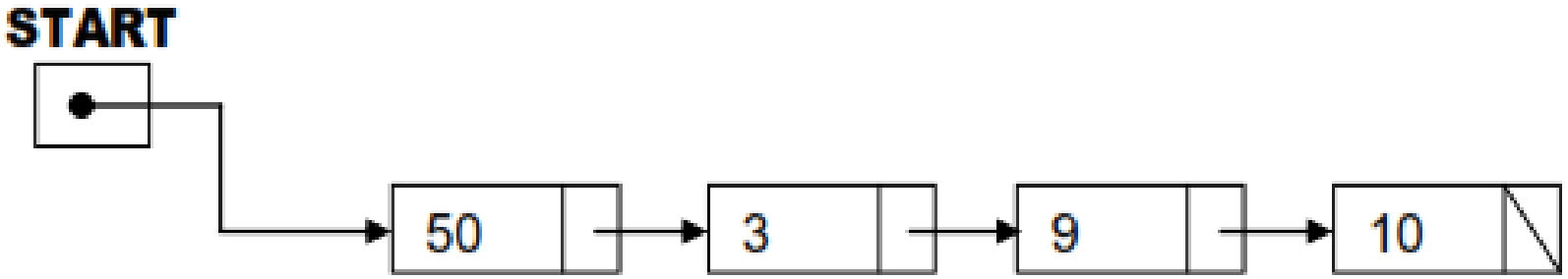
e) Finding the Range RANG of the values in LIST

5.5 Searching a Linked List

Algorithm 5.2 :LIST is unsorted

- Let LIST be a linked list in memory.
- We are given an ITEM of information.
- In this section we are going to discuss the two searching algorithms for finding the location LOC of the node where ITEM first appears in LIST.

LIST is unsorted



- The data in LIST are not necessarily sorted.
- Then one searches for ITEM in LIST by traversing through the list using a pointer variable PTR and comparing ITEM with the contents INFOR[PTR] of each node, one by one, of LIST.
- Before we update the pointer PTR by PTR := LINK[PTR]
- we require two tests. First we have to check whether we reached the end of the list; i.e., PTR = NULL
- If not, then we check to see whether INFO[PTR] = ITEM

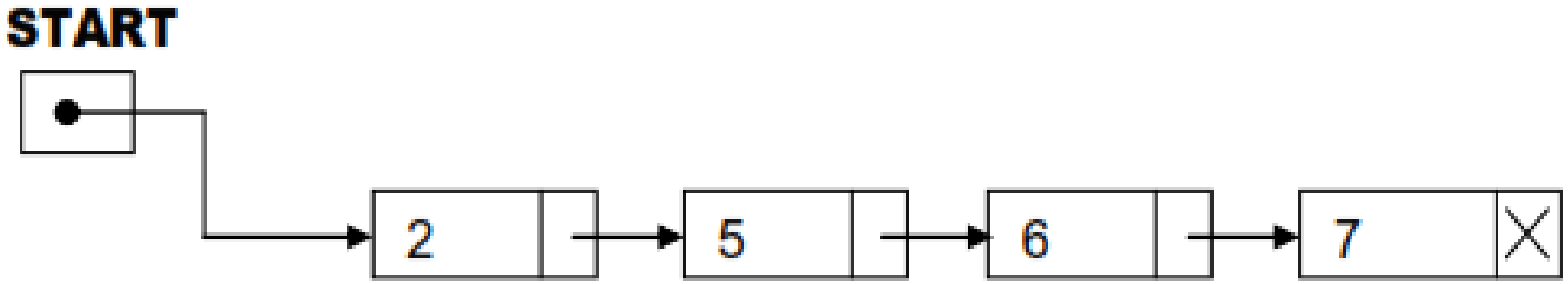
Algorithm 5.2 :LIST is unsorted

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

- Let LIST is a linked list in memory.
- This algorithm finds the location LOC of the node where ITEM first appears in LIST, or sets LOC:=NULL.

```
1.      Set PTR: =START.
2.      Repeat Step 3 While PTR ≠ NULL:
3.              If ITEM = INFO [PTR], then:
                Set LOC: =PTR, and Exit.
        Else:
                Set PTR: = LINK [PTR].
                [End of If structure.]
        [End of Step 2 loop.]
4.      [Search is unsuccessful.]
        Set LOC: = NULL.
5.      Exit.
```

LIST is Sorted



- The data in the LIST are sorted.
- Again we search for ITEM in LIST by traversing the list using a point variable PTR and comparing ITEM with the contents INFO[PTR] of each node, one by one, of LIST.
- Here we can stop once ITEM exceeds INFO[PTR].

Algorithm 5.3: :LIST is sorted deseeding

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

Let LIST is a sorted list in the memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST, or set LOC: = NULL.

Note that the List is sorted deseeding

```
1.      Set PTR: = START.
2.      Repeat Step 3 While PTR ≠ NULL:
3.              If ITEM > INFO [PTR], then:
                Set PTR: = LINK [PTR].     [PTR now points to the next node.]
                Else If ITEM = INFO [PTR], then:
                        Set LOC: = PTR, and Exit.  [Search is successful]
                Else:
        Set LOC: =NULL, and Exit. [ITEM now exceeds INFO [PTR].]
                [End of If structure.]
        [End of Step 2 loop.]
4.      Set LOC: =NULL.
5.      Exit.
```

5.6 Memory allocation; Garbage collection

Memory allocation

- Memory space can be reused if a node is deleted from a list
  - i.e deleted node can be made available for future use.
- Together with the linked list in memory, a special list is maintained which consists of unused memory cells.
- This list, which has its own pointer, is called the **list of available space or the 'Free Storage list' or the 'Free Pool' .**

Example 5.10

- Suppose the list of patients in last Example is stored in the linear arrays BED and LINK (so that the patient in bed K is assigned to BED[K]).
- Then the available space in the linear array BED may be linked as in Fig. 5.9.
- Observe that:
- BED[10] is the first available bed,
- BED[2] is the next available bed, and
- BED[6] is the last available bed.
- Hence BED[6] has the null pointer in its next pointer field; that is, LINK[6] = 0.

EXAMPLE 5.10

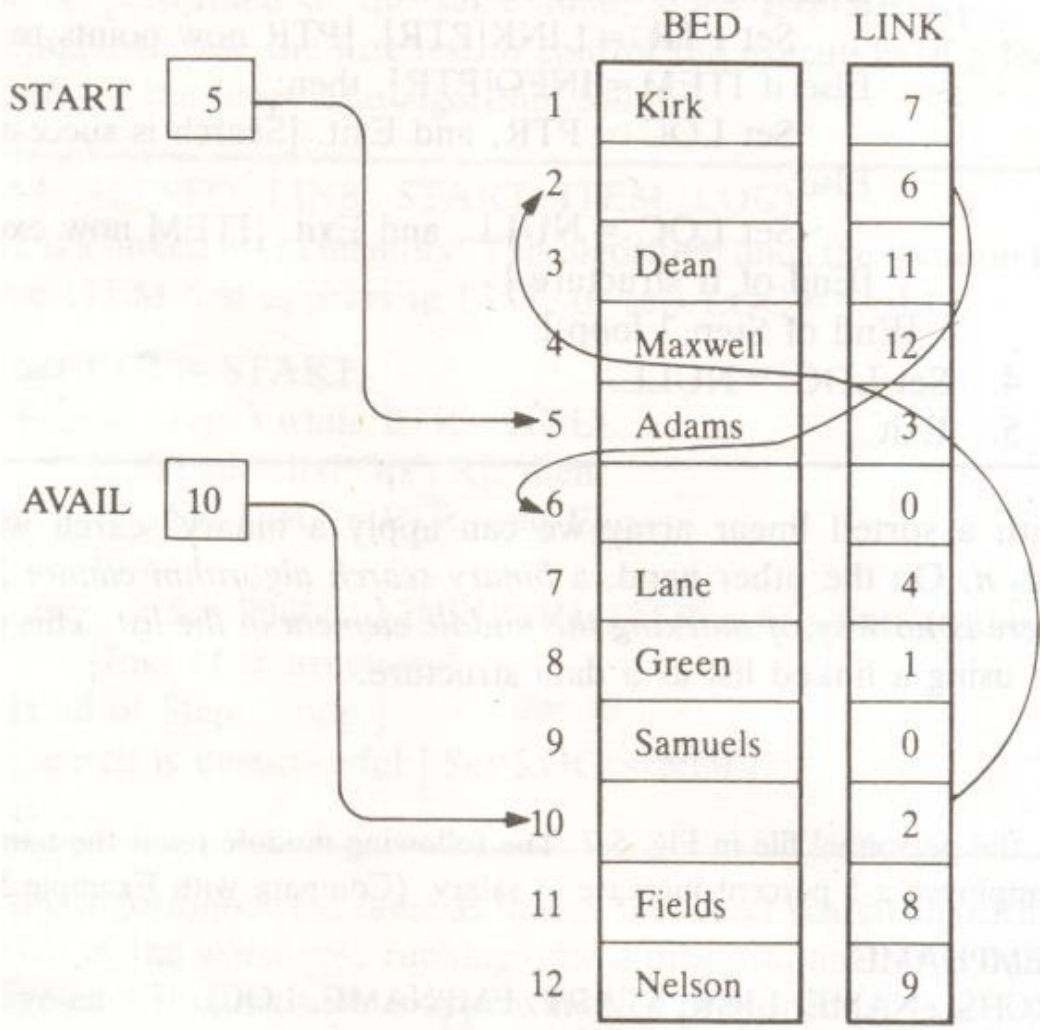


Fig. 5-9

EXAMPLE 5.11 (a)

(a) The available space in the linear array TEST in Fig. may be linked as in Fig. Observe that each of the lists ALG and GEOM may use the AVAIL list. Note that AVAIL = 9, so TEST[9] is the first free node in the AVAIL list. Since LINK[AVAIL] = LINK[9] = 10, TEST[l0] is the second free node in the AVAIL list. And so on.

EXAMPLE 5.11 (a)

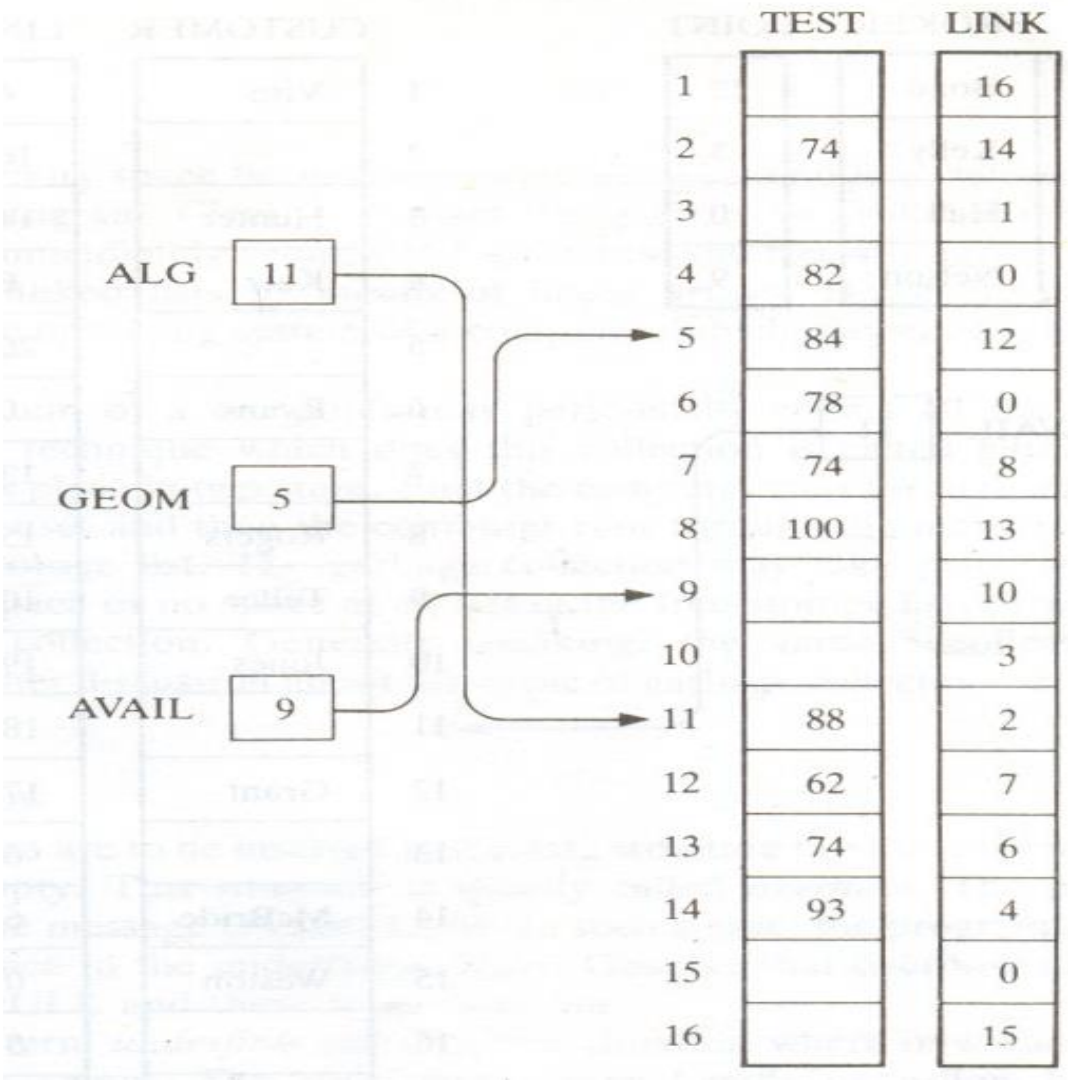


Fig. 5-10

EXAMPLE 5.11 (b)

(b) Consider the personnel file in Fig. 5-7. The available space in the linear array NAME may be linked as in Fig. 5-11. Observe that the free-storage list in NAME consists of NAME[8], NAME[ll], NAME[13], NAME[5] and NAME[1]. Moreover, observe that the values in LINK simultaneously list the free-storage space for the linear arrays SSN, SEX and SALARY.

EXAMPLE 5.11 (b)

START
6

AVAIL
8

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

Fig. 5-11

EXAMPLE 5.11 (c)

(c)The available space in the array CUSTOMER in the Fig. may be linked as in Fig. given. We emphasize that each of the four lists may use the AVAIL list for a new customer.

EXAMPLE 5.11 (c)

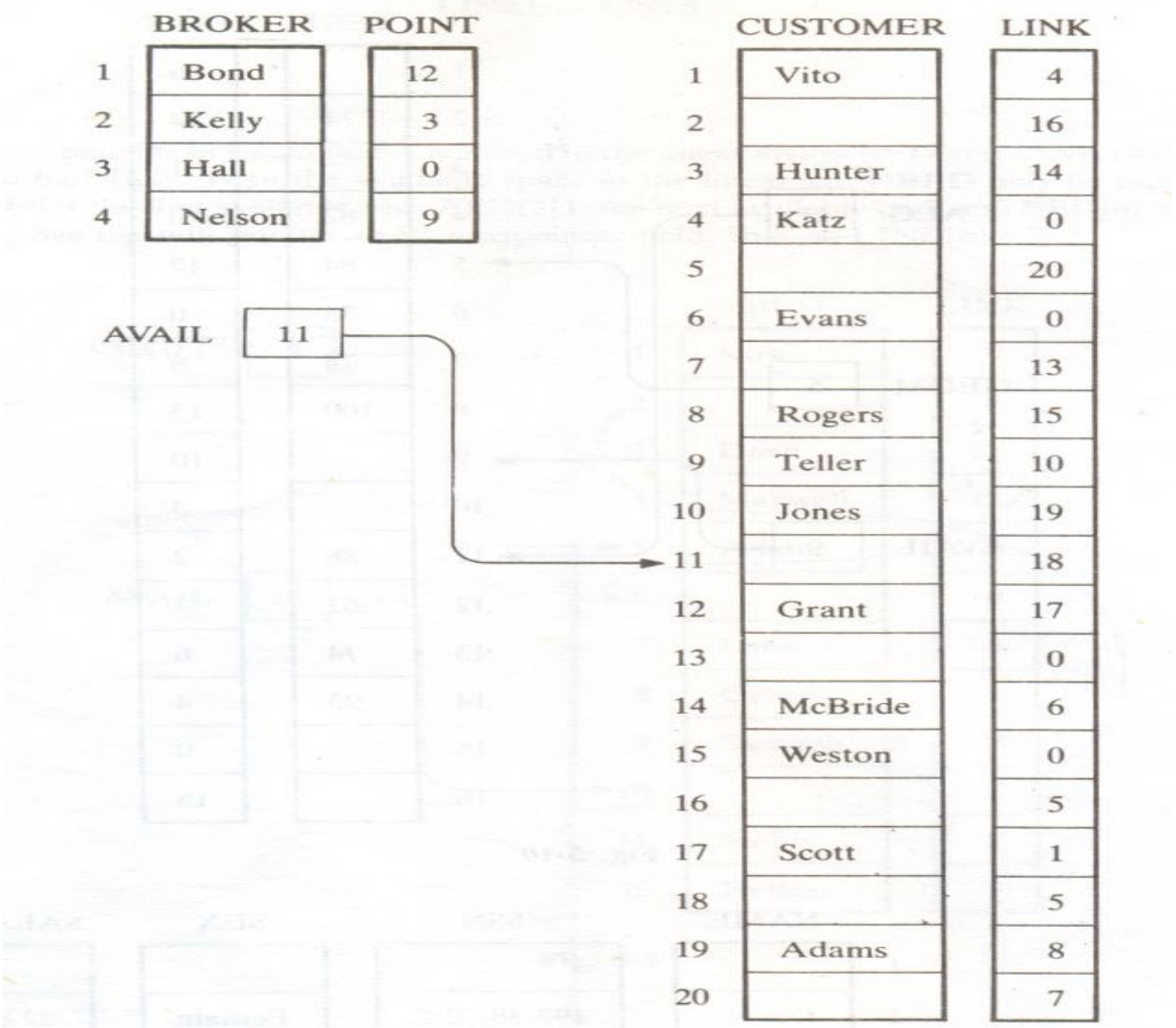


Fig. 5-12

EXAMPLE 5.12

- Suppose LIST(INFO, LINK, START, AVAIL) has memory space for $n = 10$ nodes. Furthermore, suppose LIST is initially empty. Figure shows the values of LINK so that the AVAIL list consists of the sequence

  INFO[l],INFO[2],……,INFO[10]

  that is, so that the AVAIL list consists of the elements of INFO in the usual order. Observe that START = NULL, since the list is empty.

EXAMPLE 5.12

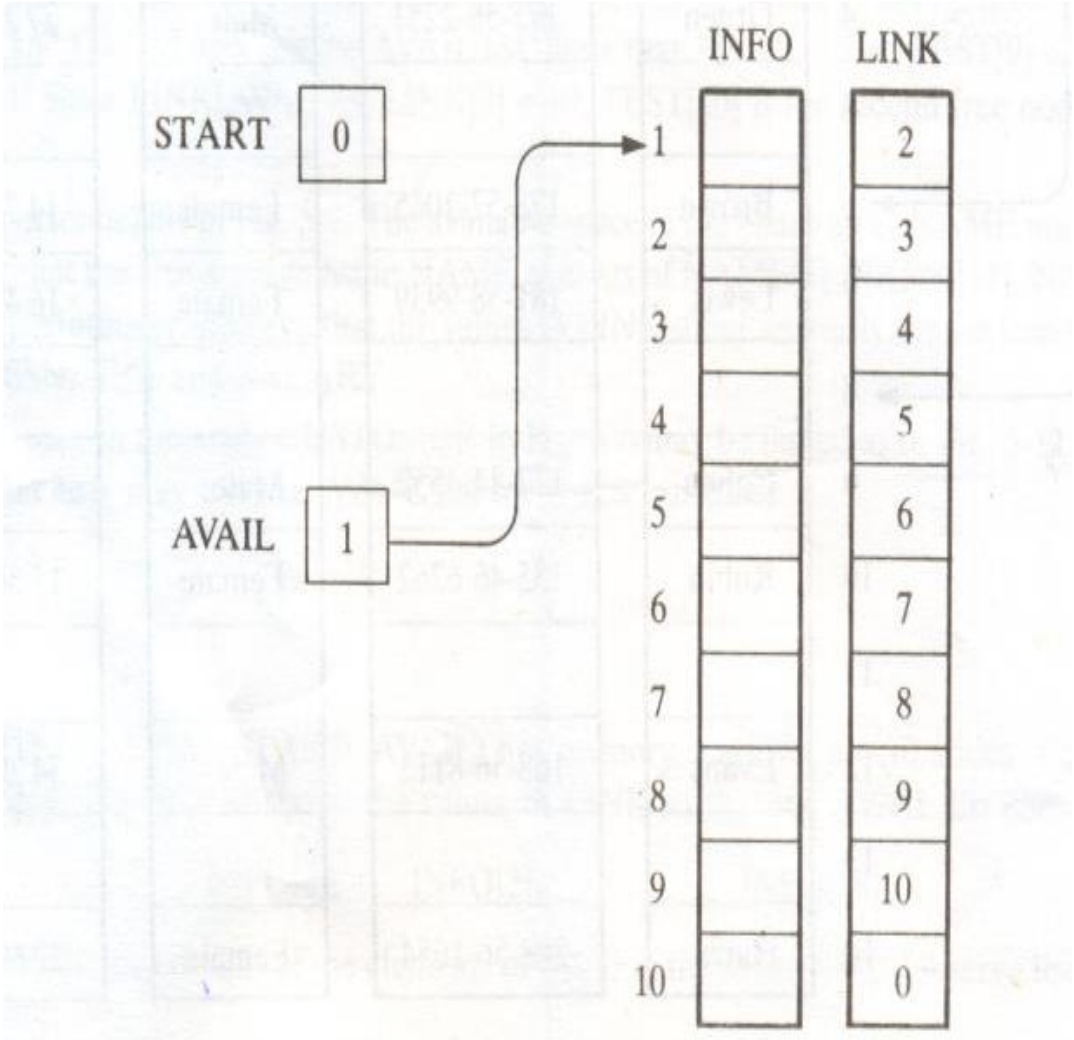


Fig. 5-13

5.6 Garbage collection (page 127)

Garbage Collection

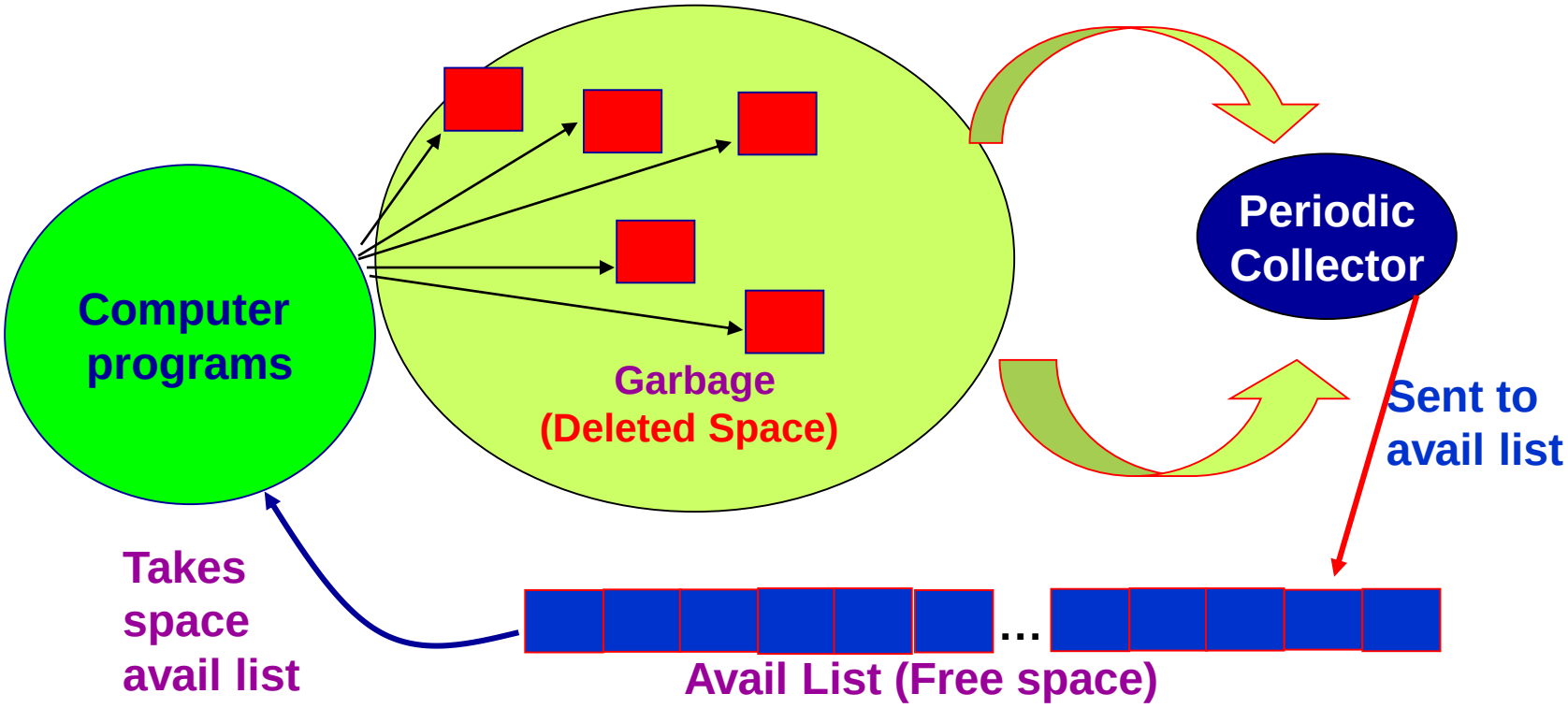
- The operating system of a computer may periodically collect all the deleted space onto the free storage list.
- Any technique which does this collection is called garbage collection.

Garbage Collection

Garbage collection usually takes place in two steps.

1.The computer runs through all lists, tagging those cells which are currently in use.

2.The computer runs through all list collecting all list, collecting all untagged space onto the free storage list.



Garbage Collection

The garbage collection may take place

- when there is only some minimum amount of space or no space at all left in the free – storage list, or
- when the CPU is idle & has time to do the collection.

[NOTE:- Garbage collection is invisible to the programmer .]

Overflow and Underflow

Condition : **AVAIL= NULL**

- **Overflow**:
  - Sometimes data are inserted into a data structure but there is no available space.
  - This situation is called overflow
  - Example: In linked list overflow occurs when
    - AVAIL= NULL and
    - There is an insertion operation

Overflow and Underflow

Condition : **START= NULL**

- **Underflow**:
  - Situation:
    - Want to delete data from data structure that is empty.
  - Example: In linked list **Underflow** occurs when
    - START = NULL and
    - There is an deletion operation

Programming problems

Problems 5.35 to 5.40 refer to the data structure in Fig. 5-51, which consists of four alphabetized lists of clients and their respective lawyers.

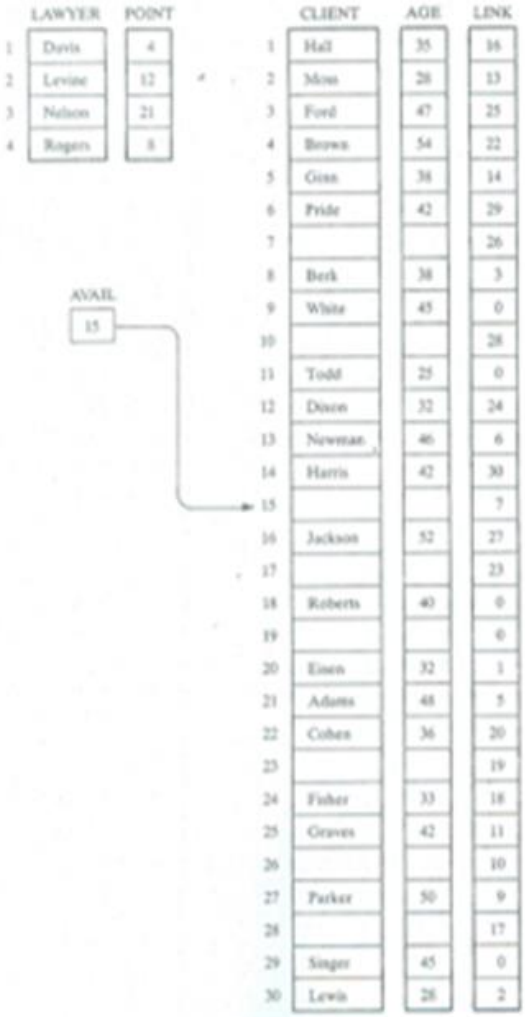


Fig. 5-51

Programming problems

5.35 Write a program which reads an integer K and prints the list of clients of lawyer K. Test the program for each K.

5.36 Write a program which prints the name and lawyer of each client whose age is L or higher. Test the program using (*a*) L = 41 and (*b*) L = 48.

5.37 Write a program which reads the name LLL of a lawyer and prints the lawyer's list of clients. Test the program using (*a*) Rogers, (*b*) Baker and (*c*) Levine.

5.38 Write a program which reads the NAME of a client and prints the client's name, age and lawyer. Test the program using (*a*) Newman, (*b*) Ford, (*c*) Rivers and (*d*) Hall.

5.39 Write a program which reads the NAME of the client and deletes the client's record from the structure. Test the program using (*a*) Lewis, (*b*) Klein and (*c*) Parker.

5.40 Write a program which reads the record of a new client, consisting of the client's name, age and lawyer, and inserts the record into the structure. Test the program using (*a*) Jones, 36, Levine; and (*b*) Olsen, 44, Nelson.

تم الإنتهاء من المحاضرة