Data Structure

Linked Lists

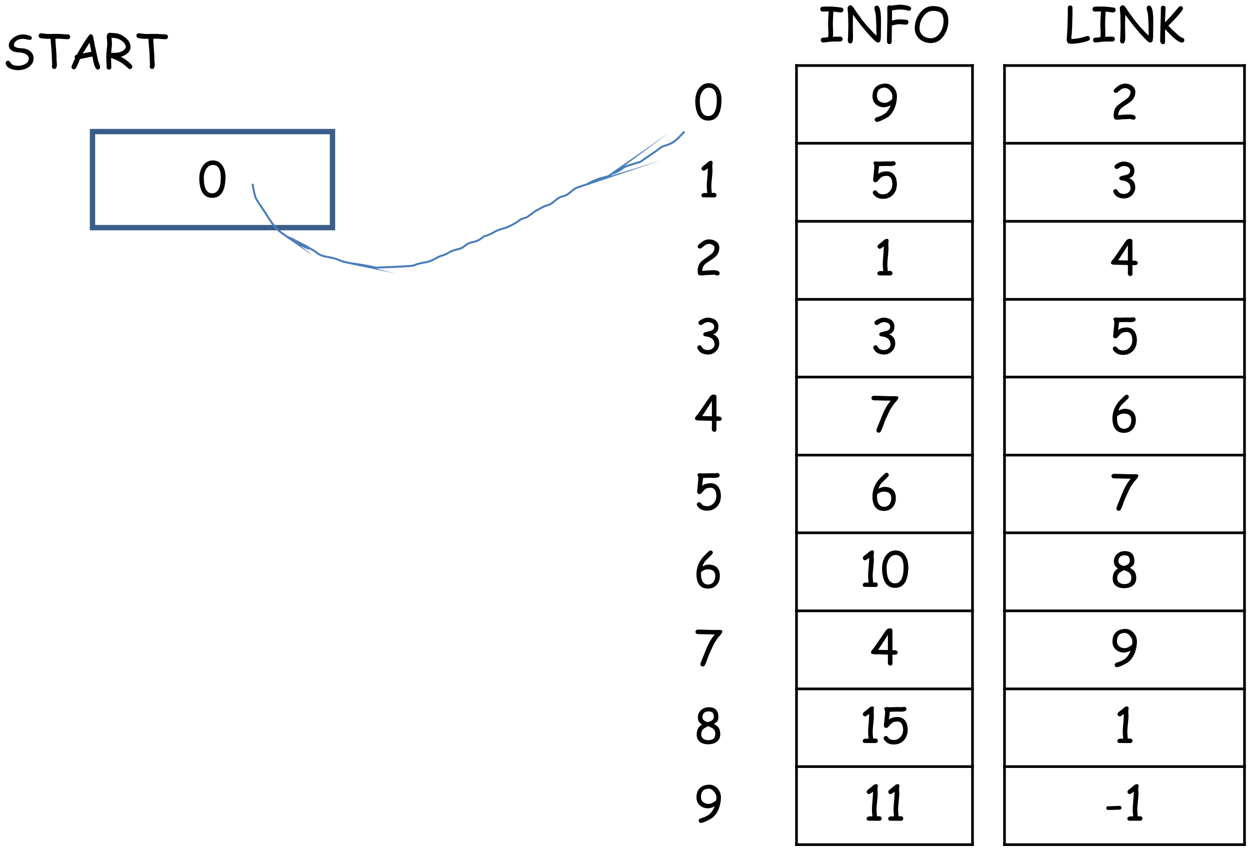
Instructors:
Md Nazrul Islam Mondal &
Rizoan Toufiq
Department of Computer Science & Engineering
Rajshahi University of Engineering &
Technology Rajshahi-6204

Outline

- Introduction
- Linked List
- Representation of Linked Lists in Memory
- **Traversing a Linked List**
- **Searching a Linked List**
- **Memory Allocation; Garbage Collection**
- **Insertion into a Linked List**
- Deletion from a Linked List
- Header Linked List
- Two Way Lists

Traversing a Linked List

Traversing a Linked List



START

0

	INFO	LINK
0	9	2
1	5	3
2	1	4
3	3	5
4	7	6
5	6	7
6	10	8
7	4	9
8	15	1
9	11	-1

Traversing a Linked List

Algorithm 5.1: LIST is a linked list in memory. This algorithm traverses **LIST**, applying an operation **PROCESS** to each element of **LIST**. The variable **PTR** points to a node currently being processed.

```
1.  Set PTR:=START
2.  Repeat steps 3 and 4 while PTR≠NULL
3.      Apply PROCESS to INFO[PTR]
4.      SET PTR := LINK[PTR]
    [End of Repeat 2 loop]
5.  Exit
```

Searching a Linked List

Searching a Linked List

(LIST is unsorted)

START

0

	INFO	LINK
0	9	2
1	5	3
2	1	4
3	3	5
4	7	6
5	6	7
6	10	8
7	4	9
8	15	1
9	11	-1

Searching a Linked List
(LIST is unsorted)

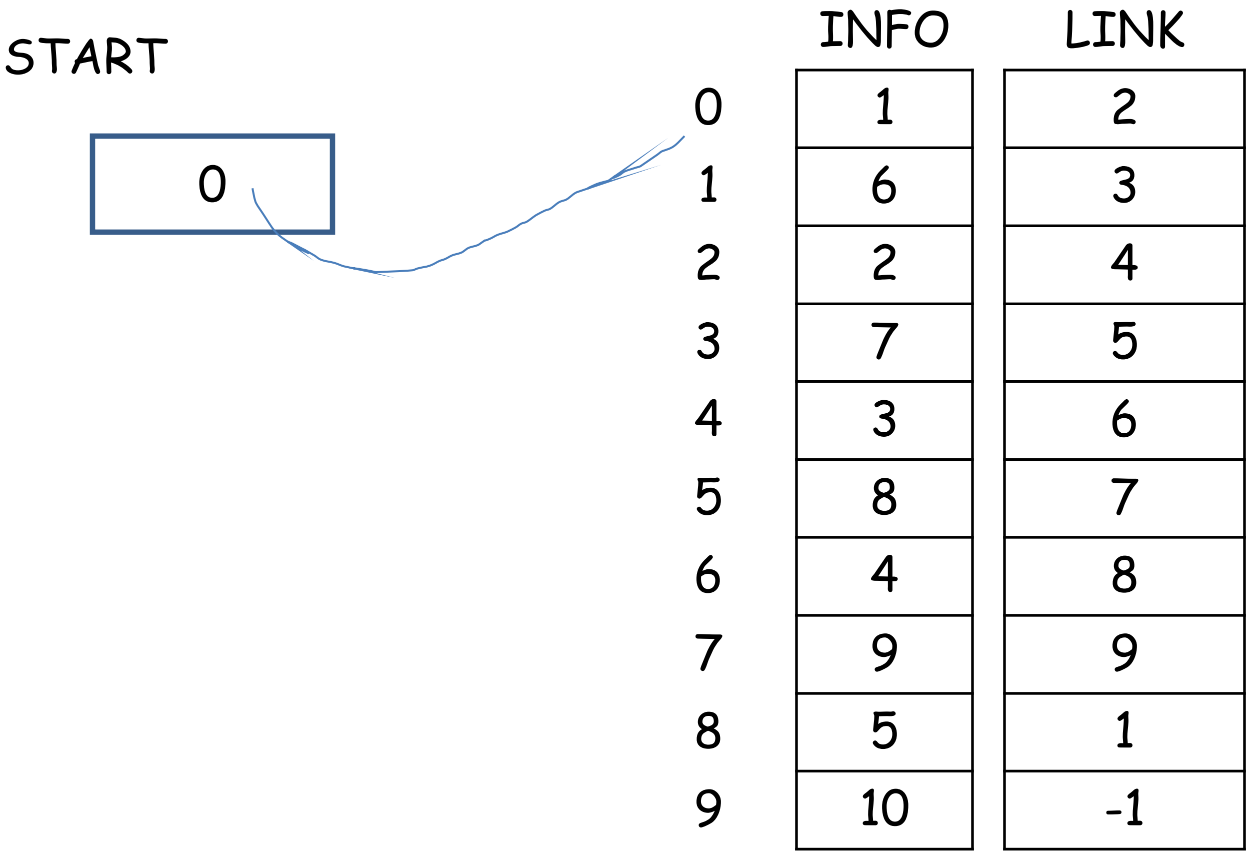
Algorithm 5.2: SEARCH (INFO, LINK, START, ITEM, LOC)

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 steps 3 and 4 while PTR≠NULL
3.      If ITEM = INFO[PTR] then:
             Set LOC:=PTR and Exit.
        Else:
             SET PTR := LINK[PTR]
        [End of If statement]
   [End of Repeat 2 loop]
4. [Search is unsuccessful] Set LOC:=NULL
5. Exit
```

Searching a Linked List

(LIST is sorted)



START

0

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

Searching a Linked List
(LIST is sorted)

Algorithm 5.3: SRCHSL (INFO, LINK, START, ITEM, LOC)

LIST is a sorted 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 steps 3 and 4 while PTR≠NULL
3.      If ITEM<INFO [PTR], then:
                    Set PTR:= LINK[PTR]
        Else if ITEM = INFO [PTR] then:
                    Set LOC:=PTR and Exit.
        Else:
                    SET LOC:= NULL, and Exit
                [End of If statement]
            [End of Repeat 2 loop]
4. [Search is unsuccessful] Set LOC:=NULL
5. Exit
```

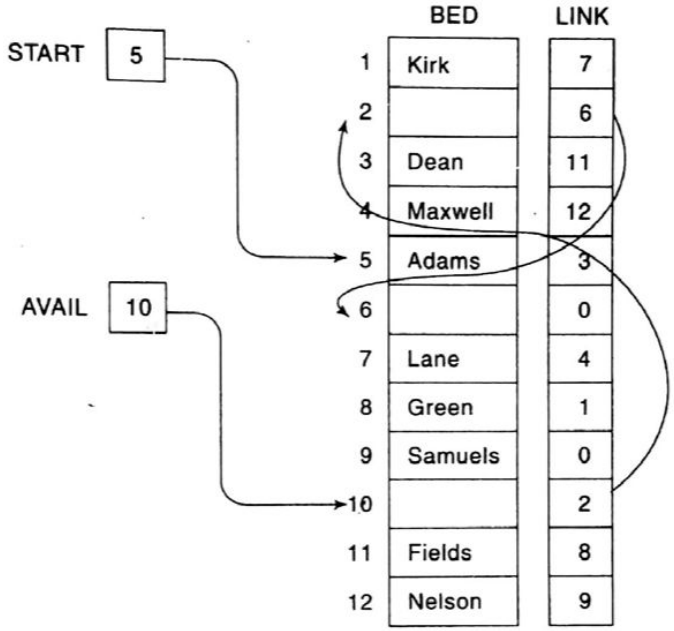
Searching a Linked List

Source Code

Memory Allocation; Garbage Collection

Memory Allocation; Garbage Collection

- A list is maintained which consists of unused memory cells. This list is called the **list of available space** or the **free storage list** or the **free pool**.



	BED	LINK
1	Kirk	7
2		6
3	Dean	11
4	Maxwell	12
5	Adams	3
6		0
7	Lane	4
8	Green	1
9	Samuels	0
10		2
11	Fields	8
12	Nelson	9

Fig. 5.9

Memory Allocation; Garbage Collection

- **Garbage Collection**:
  - (traditional process) It is time-consuming for the operation system to reinsert the deleted node from a list into the free storage list.
  - **Definition of Garbage collection** - The operating system of a computer may periodically collect all the deleted space into the free-storage list.
  - **Two steps** technique:
    - The computer run through all lists, tagging those cells which are currently in use.
    - The computer runs through the memory, collecting all untagged space into the free-storage list.
  - **Take place** –
    - Minimum amount of space or no space at all left in the free-storage list
    - CPU is idle and has time to do the collection.
    - Invisible to the programmer.

Memory Allocation; Garbage Collection

- **Overflow**
  - New data are to be inserted into a data structure but there is no available space i.e. **the free-storage list is empty**.
  - Occur:
    - AVAIL = NULL
- **Underflow**
  - The situation where one want to delete data from a data structure that is empty.
  - Occur:
    - START = NULL

Insertion into a Linked List

Insertion into a Linked List

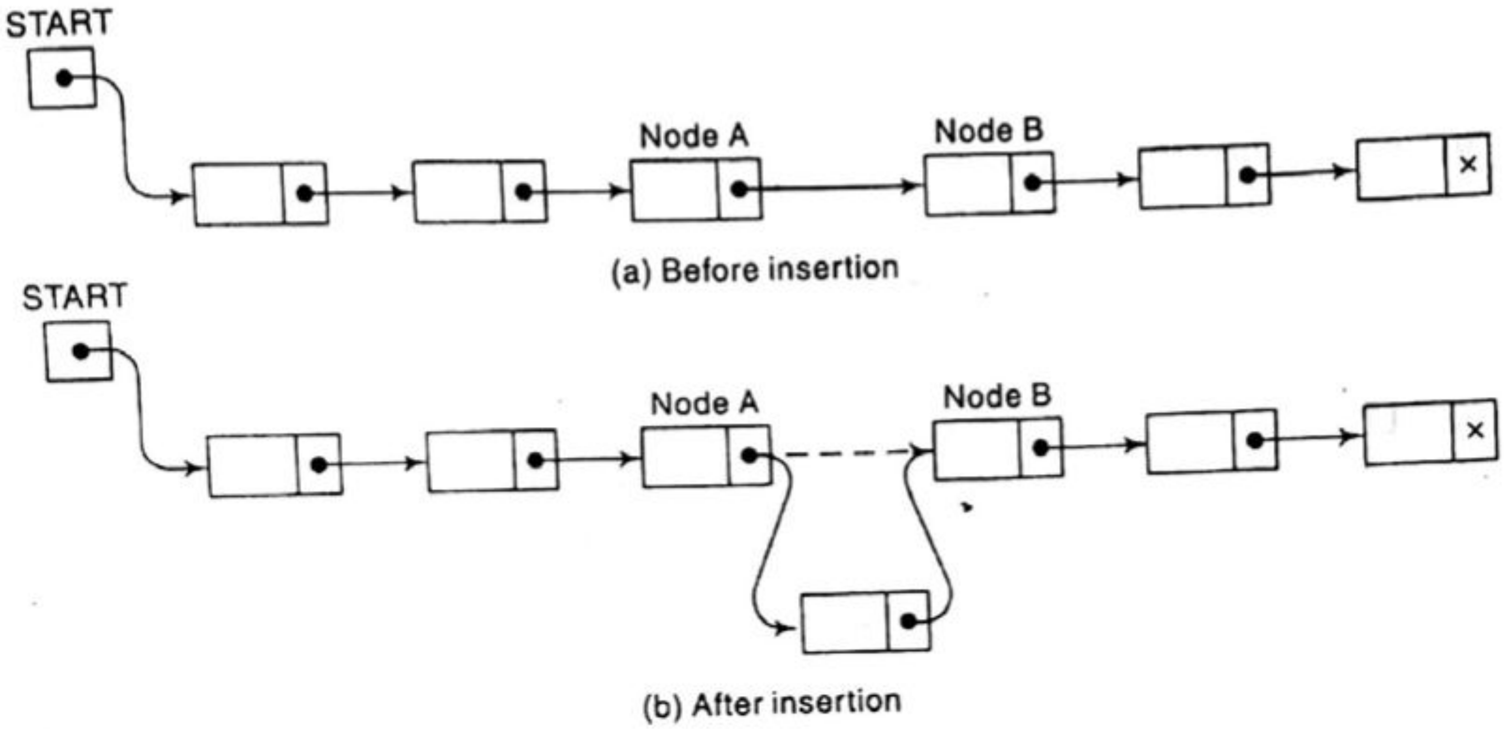
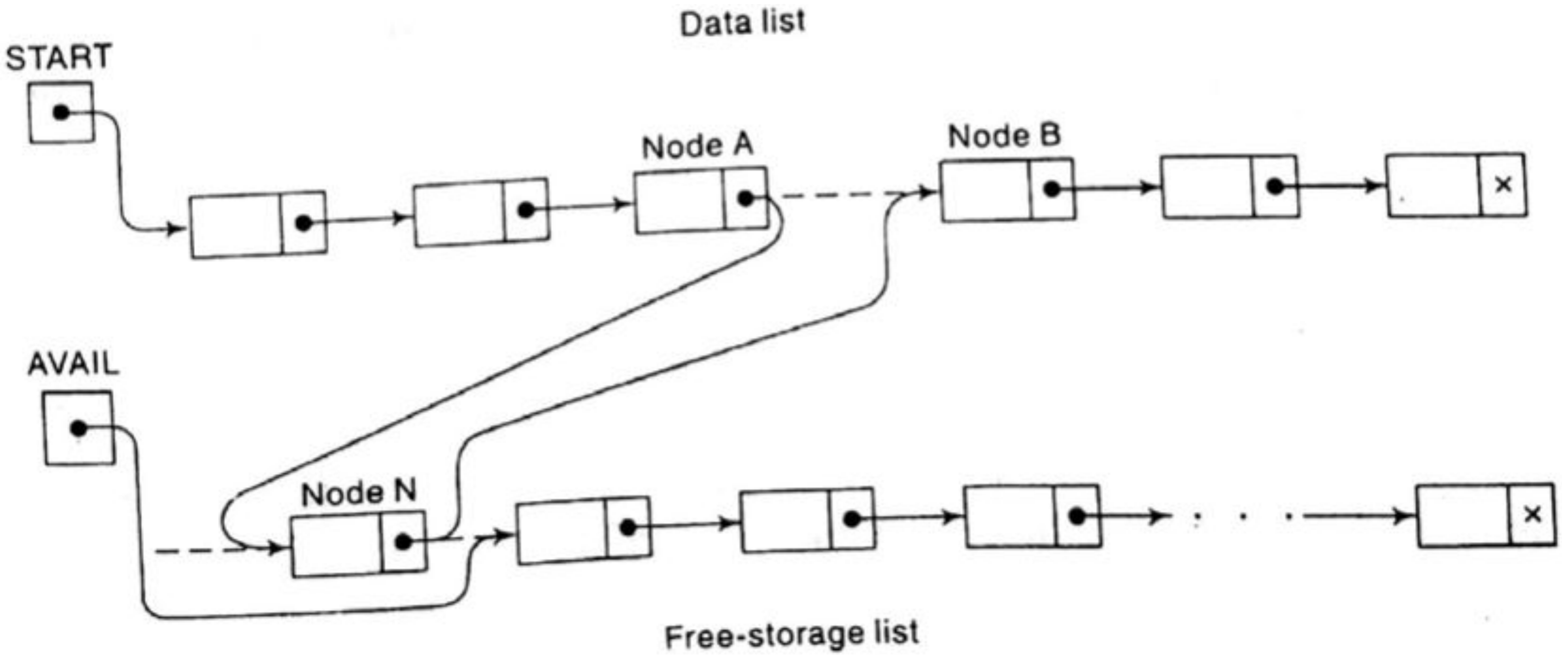


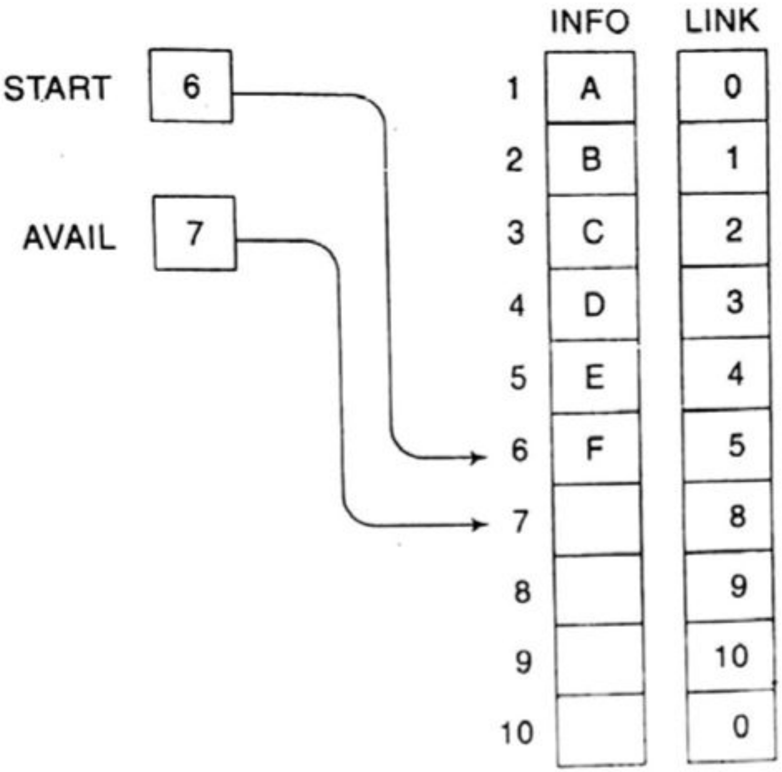
Fig. 5.14

Insertion into a Linked List

- Three Pointer fields are changed



Insertion into a Linked List



START 6

AVAIL 7

	INFO	LINK
1	A	0
2	B	1
3	C	2
4	D	3
5	E	4
6	F	5
7		8
8		9
9		10
10		0

Insertion into a Linked List

```
Algorithm: (Create a new Node) This algorithm check available memory.
1.  [OVERFLOW] If AVAIL = NULL, then: Write: OVERFLOW, and Exit
2.  [Remove first node from AVAIL.]
        Set NEW:=AVAIL and AVAIL := LINK[AVAIL]
3.  Exit
```

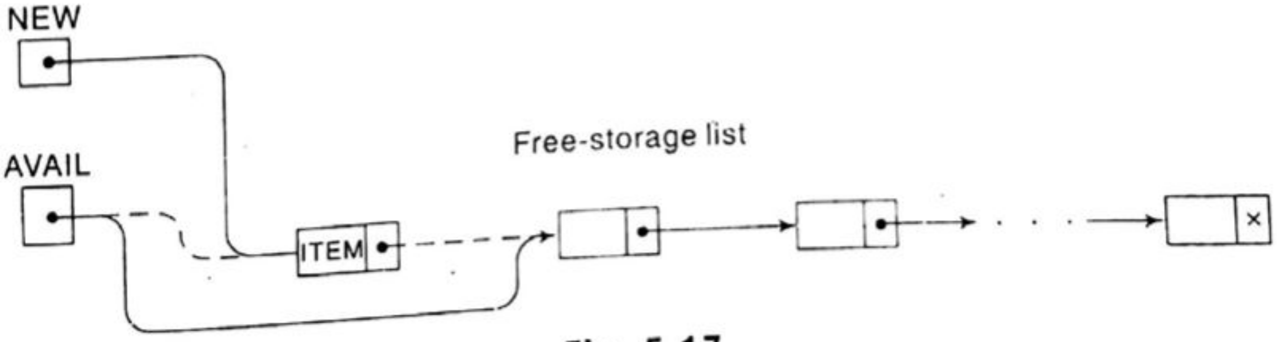


Fig. 5.17

Insertion into a Linked List (At the beginning of a List)

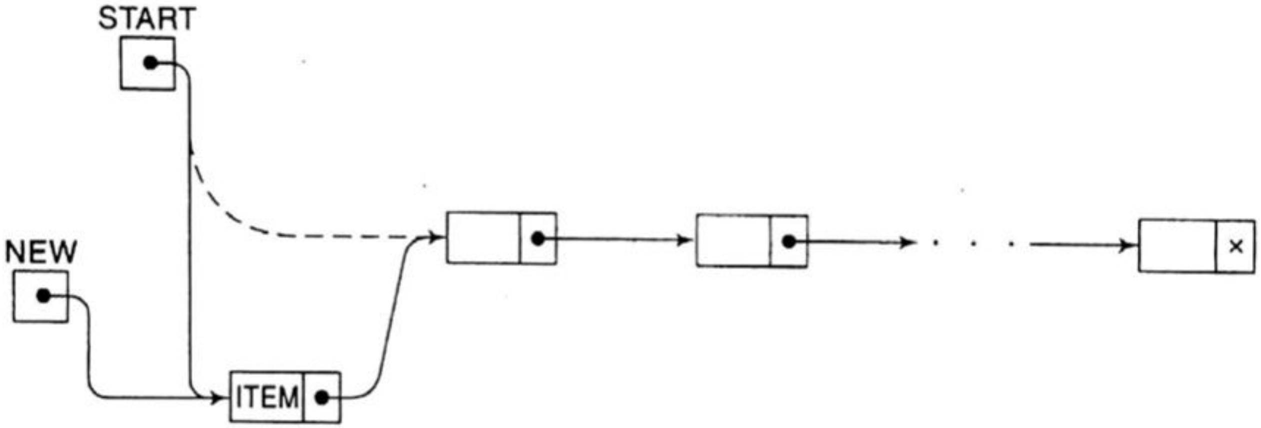
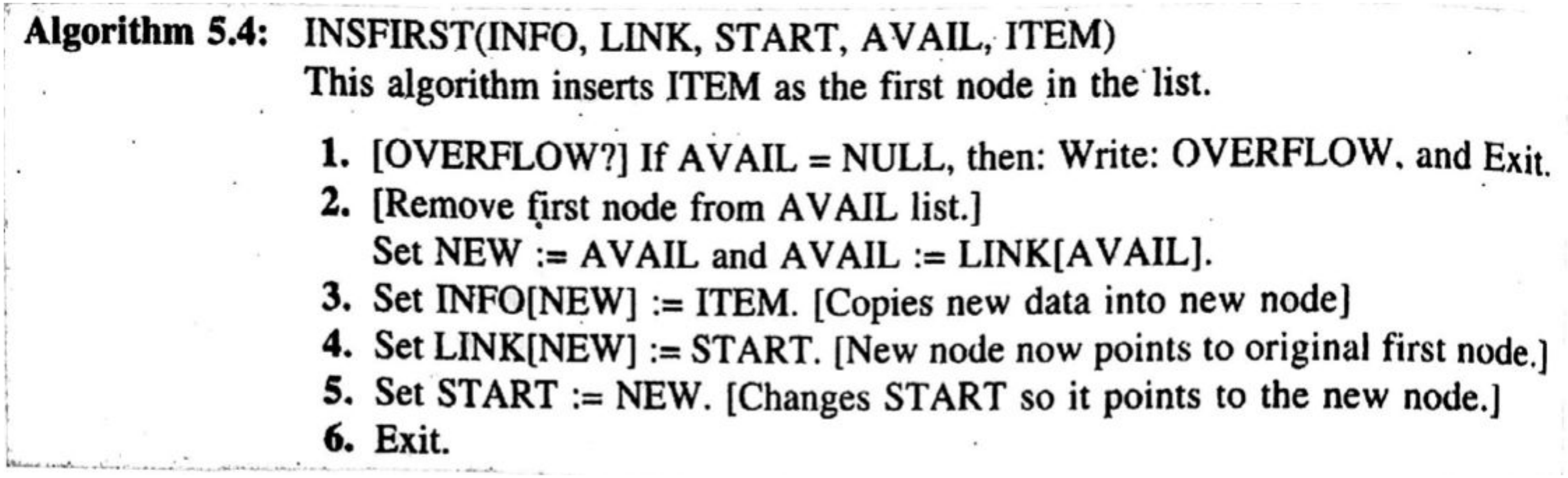


Fig. 5.18 *Insertion at the Beginning of a List*

```
LINK[NEW]←START
START ←NEW
```

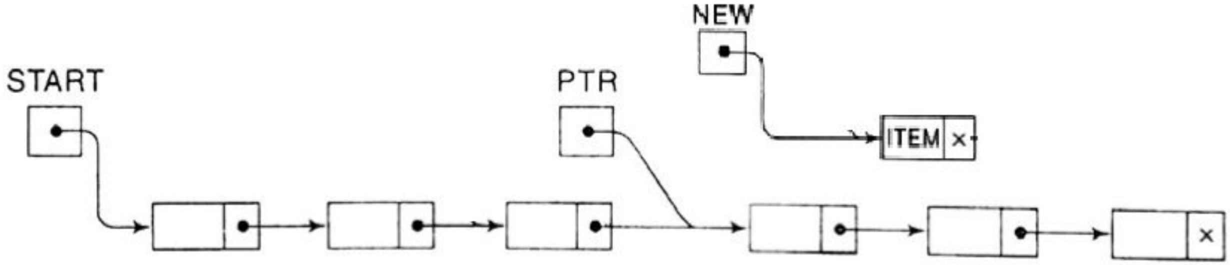
Insertion into a Linked List (At the beginning of a List)



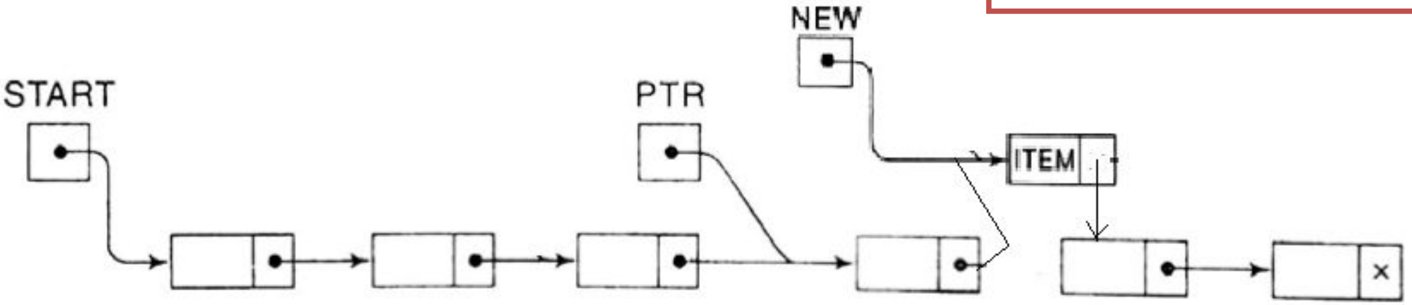
```
Algorithm 5.4: INSFIRST(INFO, LINK, START, AVAIL, ITEM)
               This algorithm inserts ITEM as the first node in the list.

                1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
                2. [Remove first node from AVAIL list.]
                   Set NEW := AVAIL and AVAIL := LINK[AVAIL].
                3. Set INFO[NEW] := ITEM. [Copies new data into new node]
                4. Set LINK[NEW] := START. [New node now points to original first node.]
                5. Set START := NEW. [Changes START so it points to the new node.]
                6. Exit.
```

LINK[NEW]←START
START ←NEW

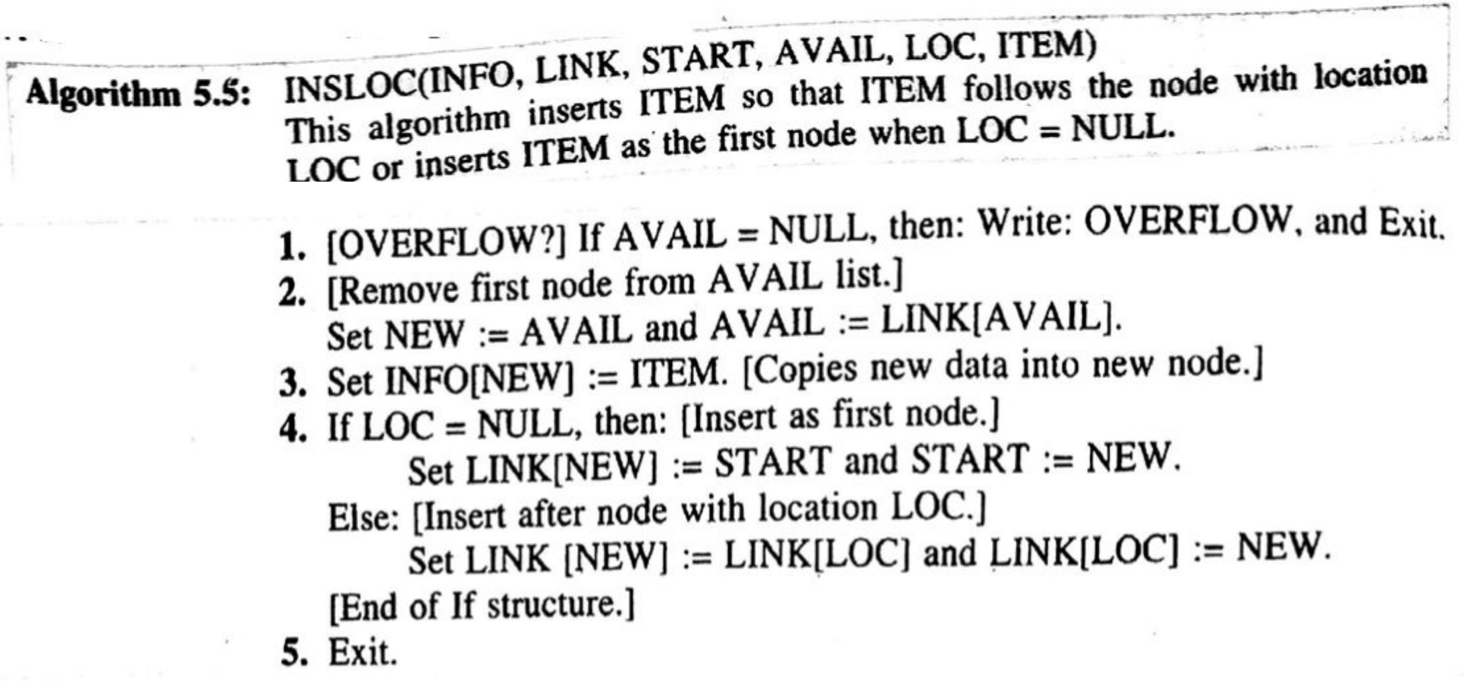
Insertion into a Linked List (After a Given Node)



```
LINK[NEW]←LINK[PTR]
LINK[PTR] ←NEW
```



Insertion into a Linked List (After a Given Node)



Algorithm 5.5: INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)
This algorithm inserts ITEM so that ITEM follows the node with location LOC or inserts ITEM as the first node when LOC = NULL.

```
1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
2. [Remove first node from AVAIL list.]
   Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM. [Copies new data into new node.]
4. If LOC = NULL, then: [Insert as first node.]
         Set LINK[NEW] := START and START := NEW.
   Else: [Insert after node with location LOC.]
         Set LINK [NEW] := LINK[LOC] and LINK[LOC] := NEW.
   [End of If structure.]
5. Exit.
```

LINK[NEW]←LINK[LOC]
LINK[LOC] ←NEW

Insertion into a Linked List (Inserting into a Sorted Linked List)

- Suppose ITEM is to be inserted into a sorted linked list LIST. Then ITEM must be inserted between node A and node B

  INFO(A)<ITEM≤INFO(B)

- Find the location LOC of node A.
- PTR pointer: Use for traversing
- It is necessary to keep track the location of the preceding node
- SAVE pointer: Use to track preceding Node

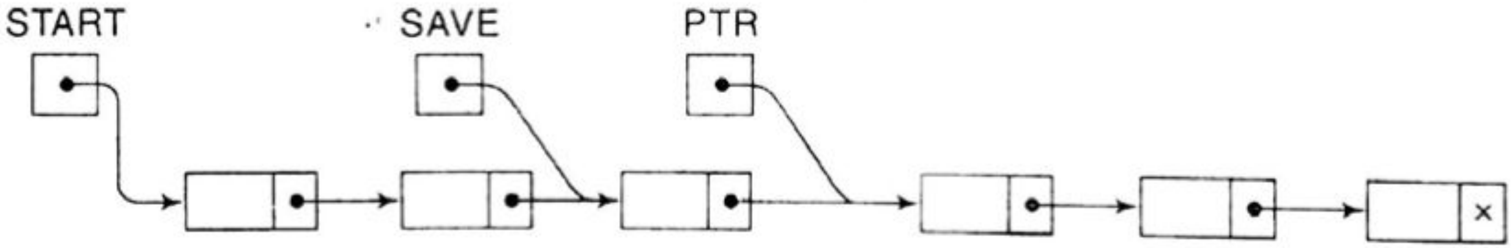
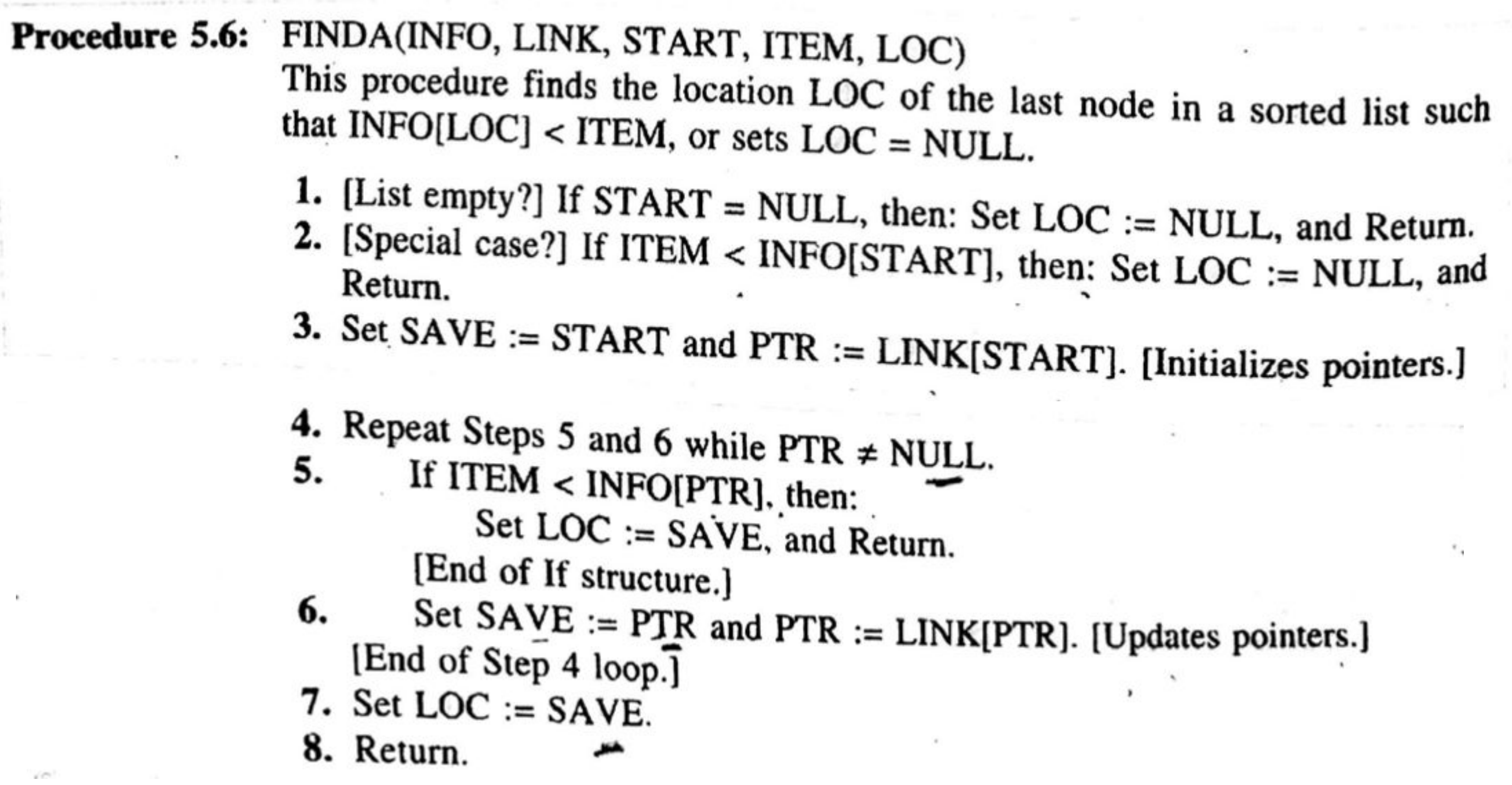


Fig. 5.20

Insertion into a Linked List (Inserting into a Sorted Linked List)



Procedure 5.6: FINDA(INFO, LINK, START, ITEM, LOC)

This procedure finds the location LOC of the last node in a sorted list such that INFO[LOC] < ITEM, or sets LOC = NULL.

```
1. [List empty?] If START = NULL, then: Set LOC := NULL, and Return.
2. [Special case?] If ITEM < INFO[START], then: Set LOC := NULL, and
   Return.
3. Set SAVE := START and PTR := LINK[START]. [Initializes pointers.]
4. Repeat Steps 5 and 6 while PTR ≠ NULL.
5.     If ITEM < INFO[PTR], then:
           Set LOC := SAVE, and Return.
       [End of If structure.]
6.     Set SAVE := PTR and PTR := LINK[PTR]. [Updates pointers.]
   [End of Step 4 loop.]
7. Set LOC := SAVE.
8. Return.
```

Insertion into a Linked List (Inserting into a Sorted Linked List)

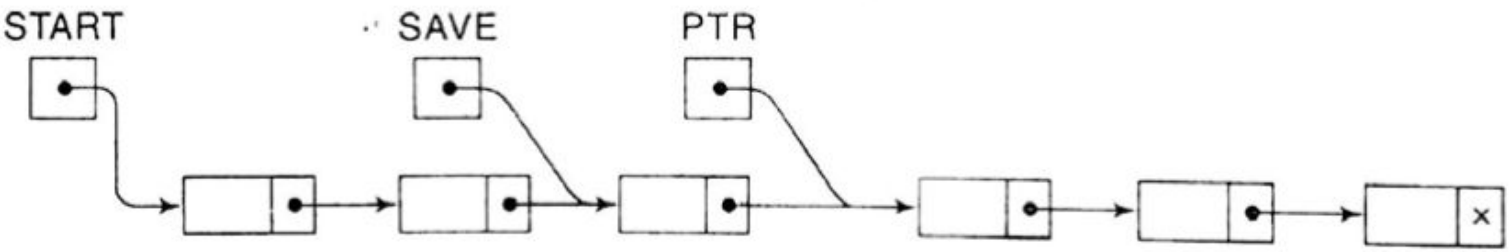
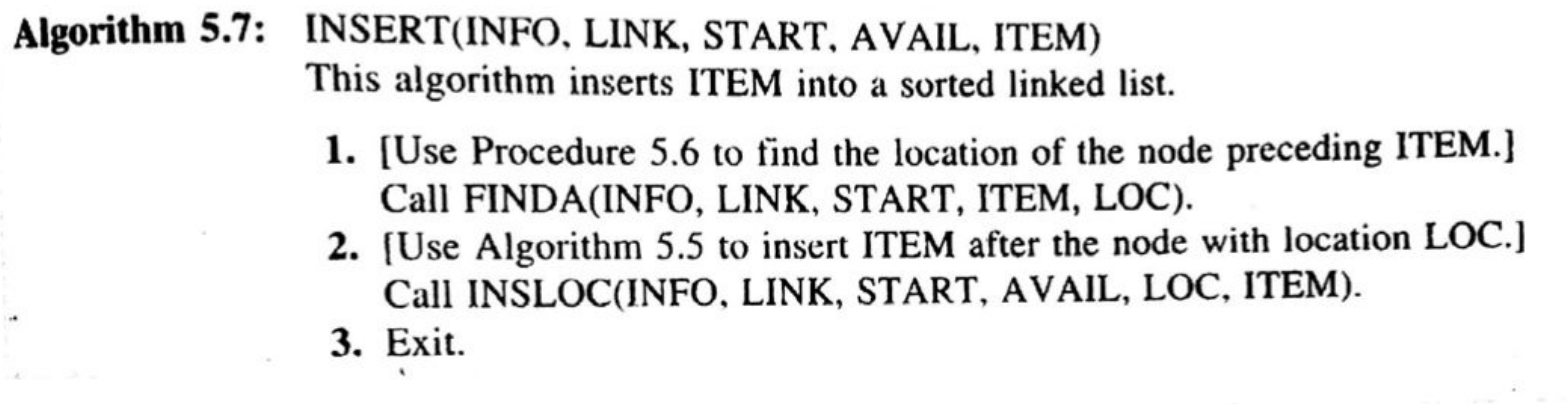


Fig. 5.20

LOC ←SAVE

- Insert ITEM after a given Node i.e. INSLOC()

Insertion into a Linked List (Inserting into a Sorted Linked List)



Algorithm 5.7: INSERT(INFO, LINK, START, AVAIL, ITEM)
This algorithm inserts ITEM into a sorted linked list.

```
1. [Use Procedure 5.6 to find the location of the node preceding ITEM.]
   Call FINDA(INFO, LINK, START, ITEM, LOC).
2. [Use Algorithm 5.5 to insert ITEM after the node with location LOC.]
   Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM).
3. Exit.
```

Source Code

END