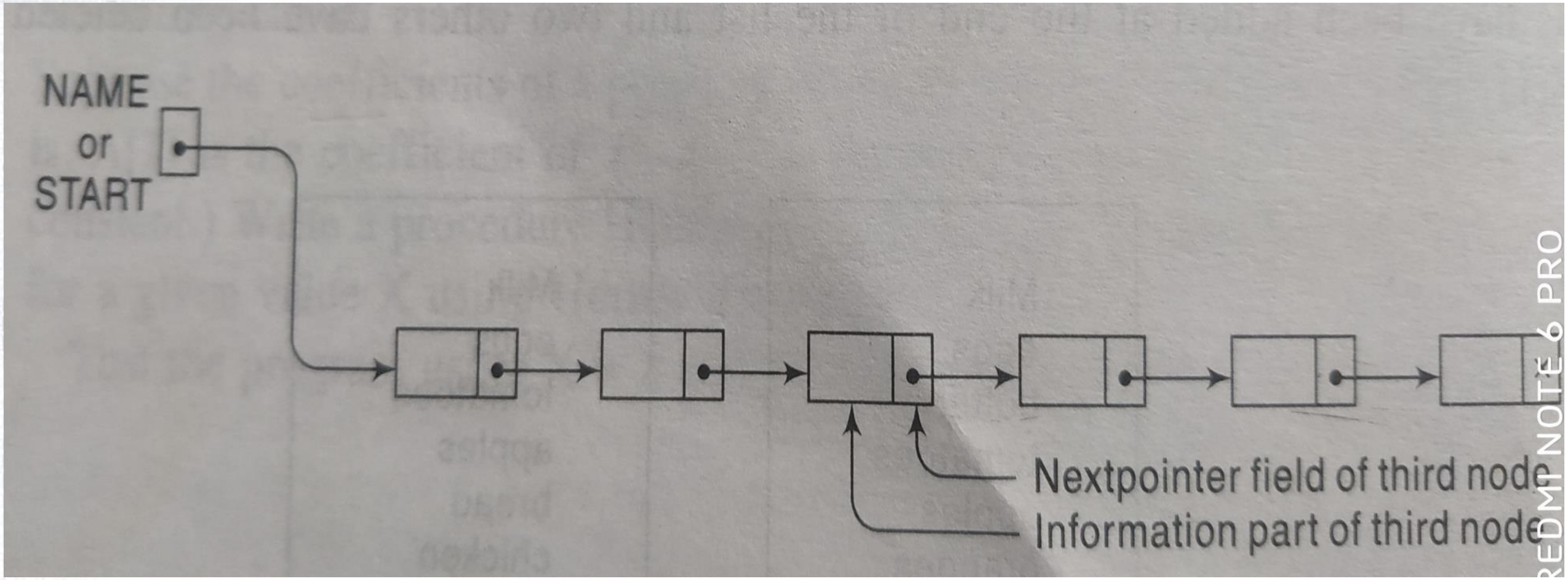
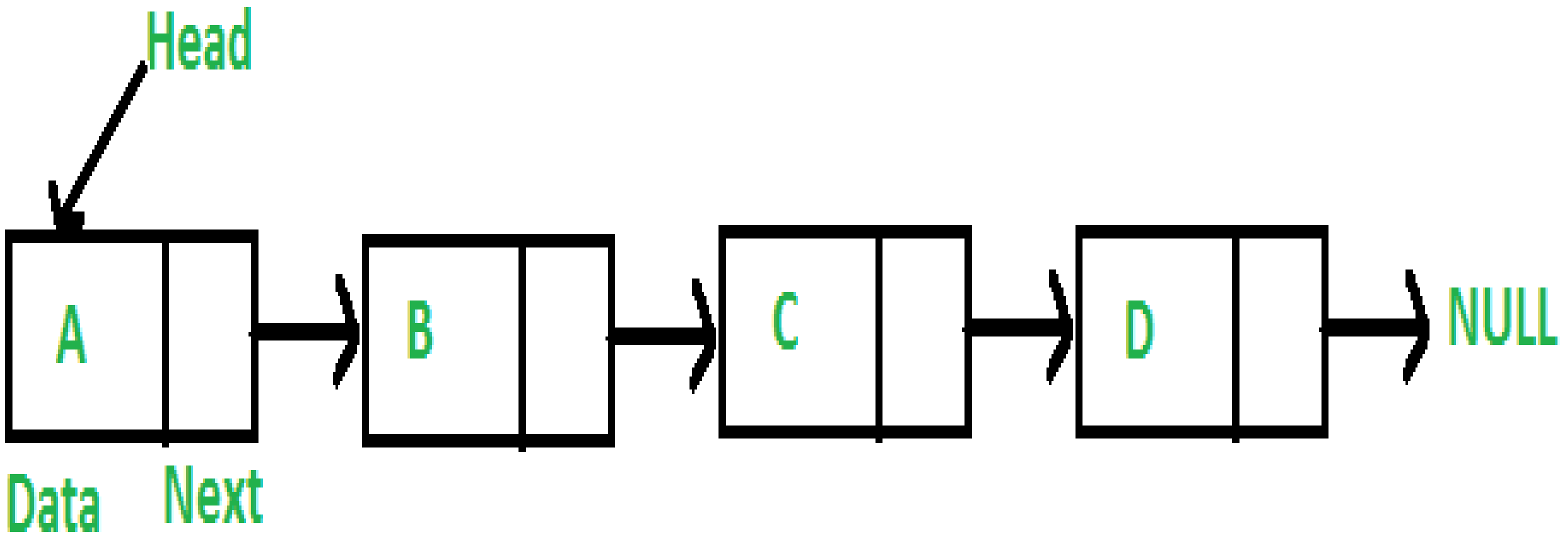
Linked List

Linked List



Linked List with 6 nodes

Linked List



Linked List with 4 nodes

Representation of LL in memory

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

In other words, NO EXIT is the character string.

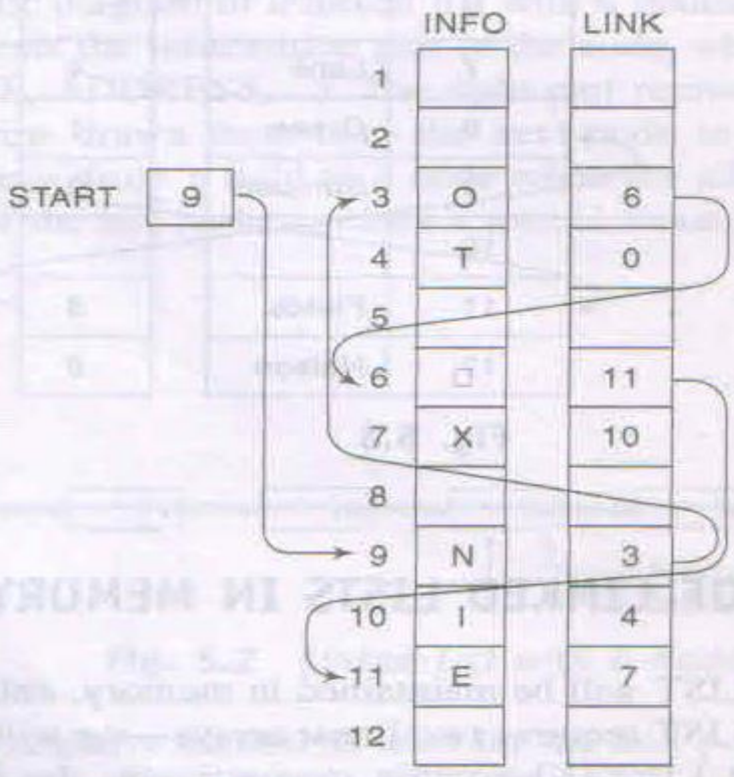
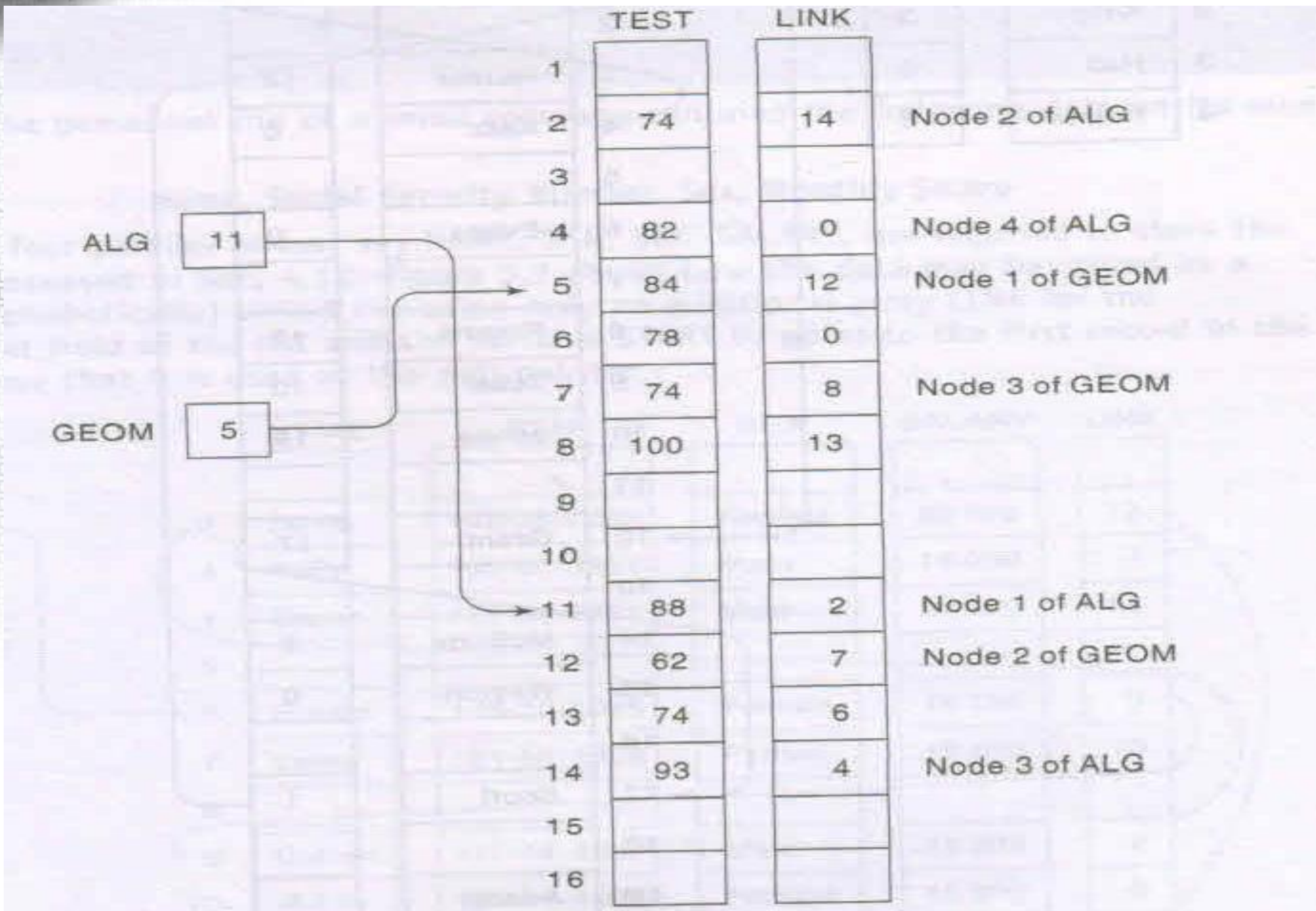
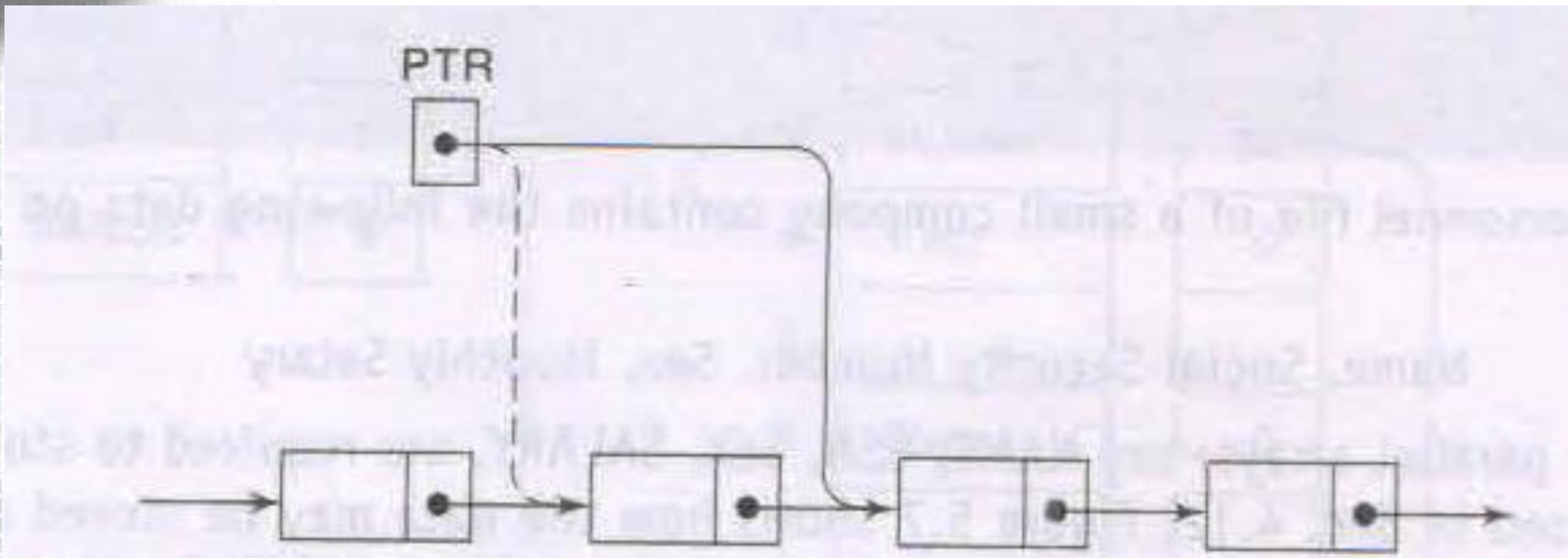


Fig. 5.4

Representation of LL in memory



Traversing a LL



Process each node exactly once

Traversing a LL (Algorithm)

(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.

Traversing a LL (Algorithm)

PRINT(INFO, LINK, START)

This procedure 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]. [Updates pointer.]

   [End of Step 2 loop.]

5. Return.

Searching a LL (Algorithm)

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 Step 3 while PTR ≠ NULL:
3. If ITEM = INFO[PTR], then:
   Set LOC := PTR, and Exit.
   Else:
   Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of If structure.]
   [End of Step 2 loop.]
4. [Search is unsuccessful.] Set LOC := NULL.
5. Exit.

- Worst-case running time is proportional to the n.
- Average-case running time is approximately proportional to n/2 (Condition that ITEM appears once in LIST but with equal probability)

Searching a LL - Sorted (Algorithm)

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

LIST is a sorted 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 PTR := LINK[PTR]. [PTR now points to 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.

Running Time ?

Searching a LL - Sorted (Algorithm)

- **Step I:** Set PTR := START. [Initializes pointer PTR.]

  Step II: Repeat steps 3 while PTR != NULL:

  Step III: If ITEM < INFO[PTR], then:

  Set LOC := NULL, and Exit. [ITEM not in LIST.]

  Else if ITEM = INFO[PTR], then:

  Set LOC := PTR, and Exit. [Search is successful.]

  Else:

  Set PTR := LINK[PTR].

  [PTR now points to the next node.]

  [End of If structure.]

  [End of step 2 loop.]

  Step IV: [Search is unsuccessful.]

  Set LOC := NULL.

  Step V: Exit.

Garbage Collection, Overflow and Underflow

Insertion Algorithm

- Insert a node at the beginning
- Insert a node after a node with a given location
- Insert a node into a sorted LIST

Insertion Algorithm - Steps

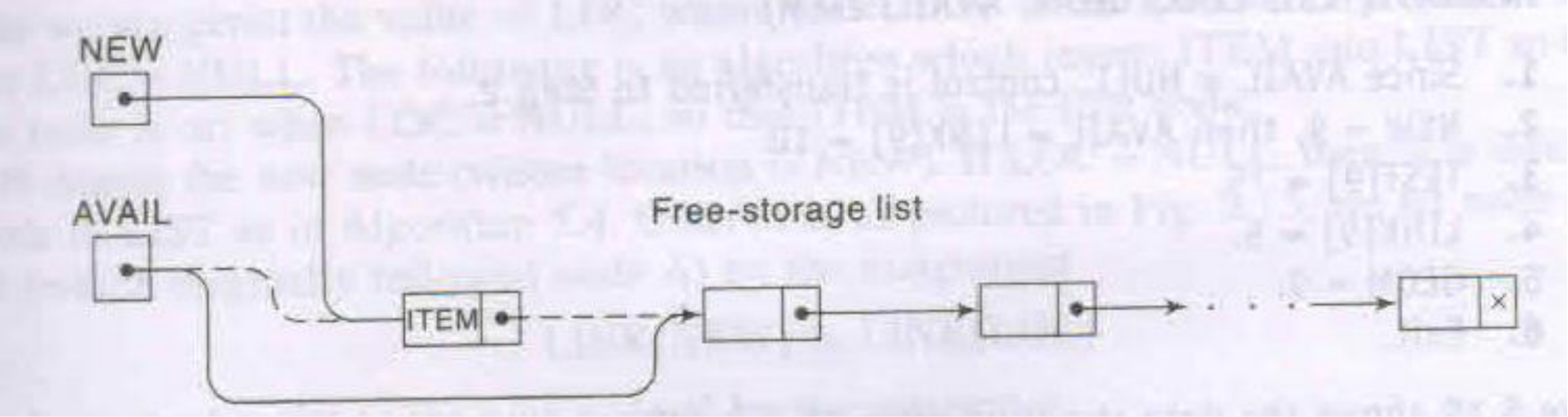
(a) Checking to see if space is available in the AVAIL list. If not, that is, if AVAIL = NULL then the algorithm will print the message OVERFLOW.

(b) Removing the first node from the AVAIL list. Using the variable NEW to keep track of the location of the new node, this step can be implemented by the pair of assignments (in this order)

NEW := AVAIL,   AVAIL := LINK[AVAIL]

(c) Copying new information into the new node. In other words,

INFO[NEW] := ITEM

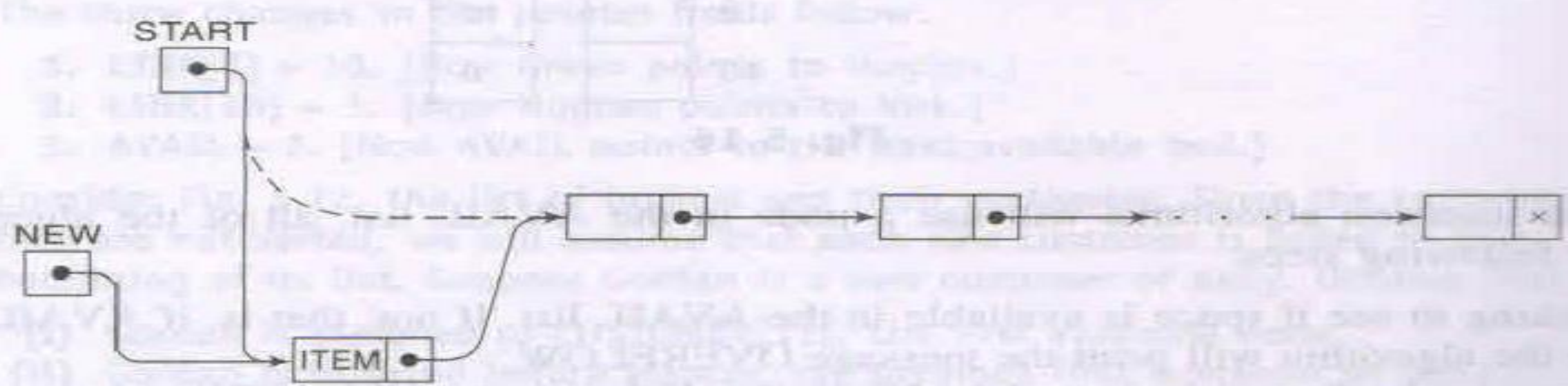


Insertion Algorithm – at the beginning

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.



Insertion Algorithm – After given node

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.

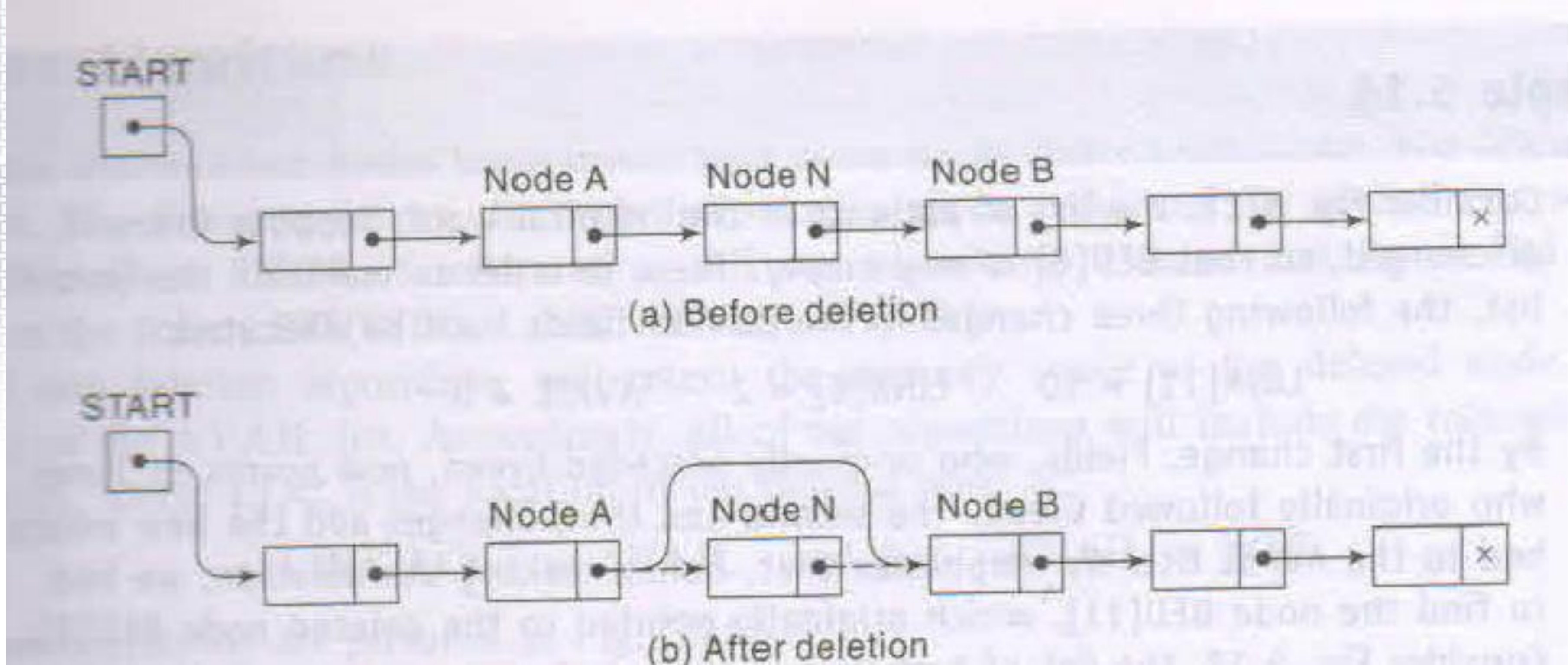
Assignment:

1. Algorithm to Insert a node in sorted Linked List
2. Algorithm to Insert a node at the end of Linked List

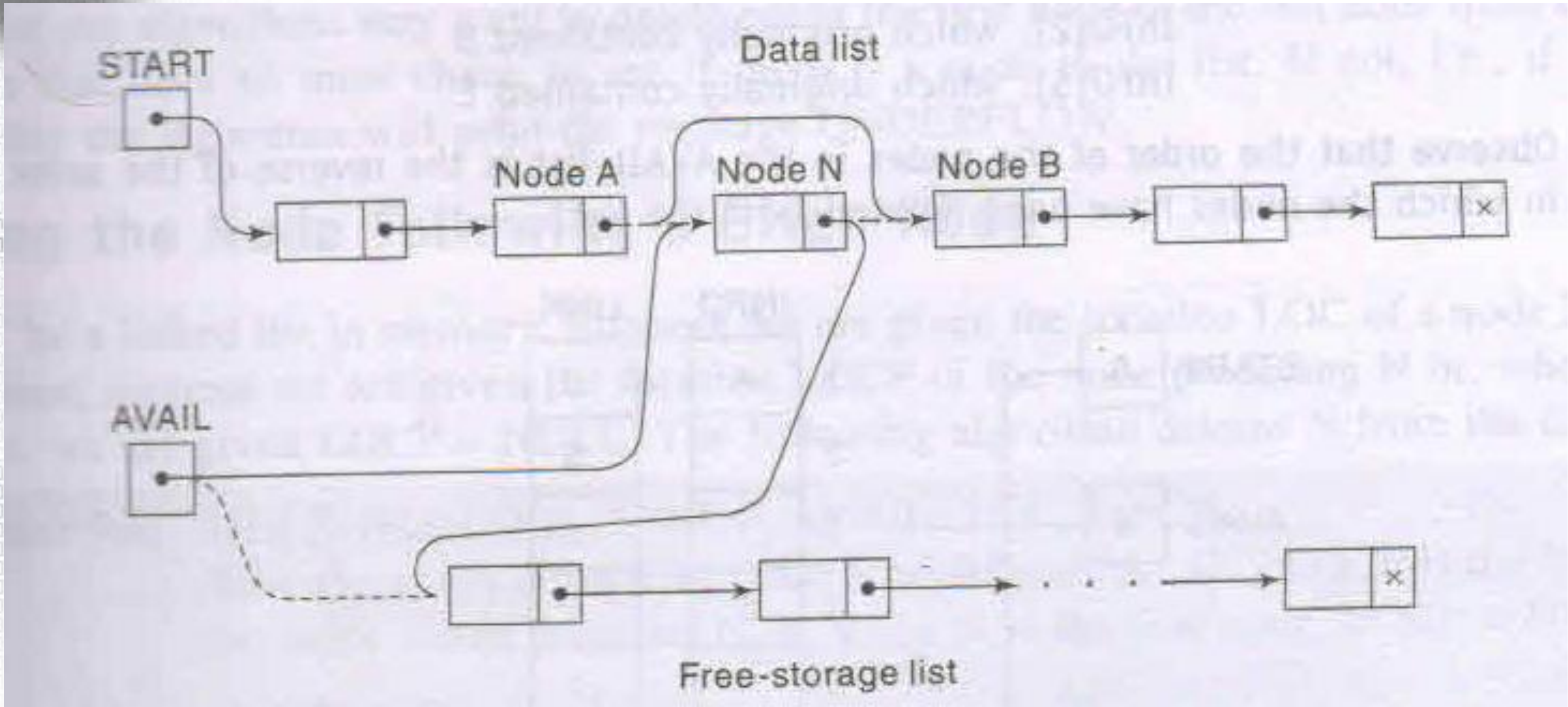
Deletion Algorithm

Two Situations

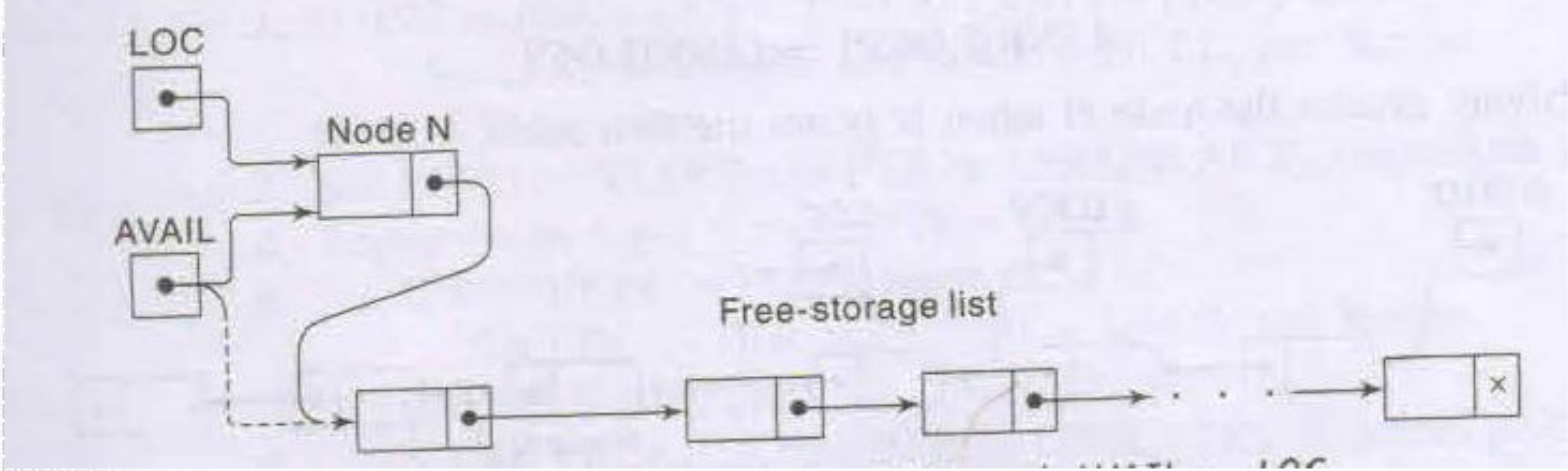
1. Deleting a node following a given node
2. Deleting the node with given ITEM



Deletion Algorithm

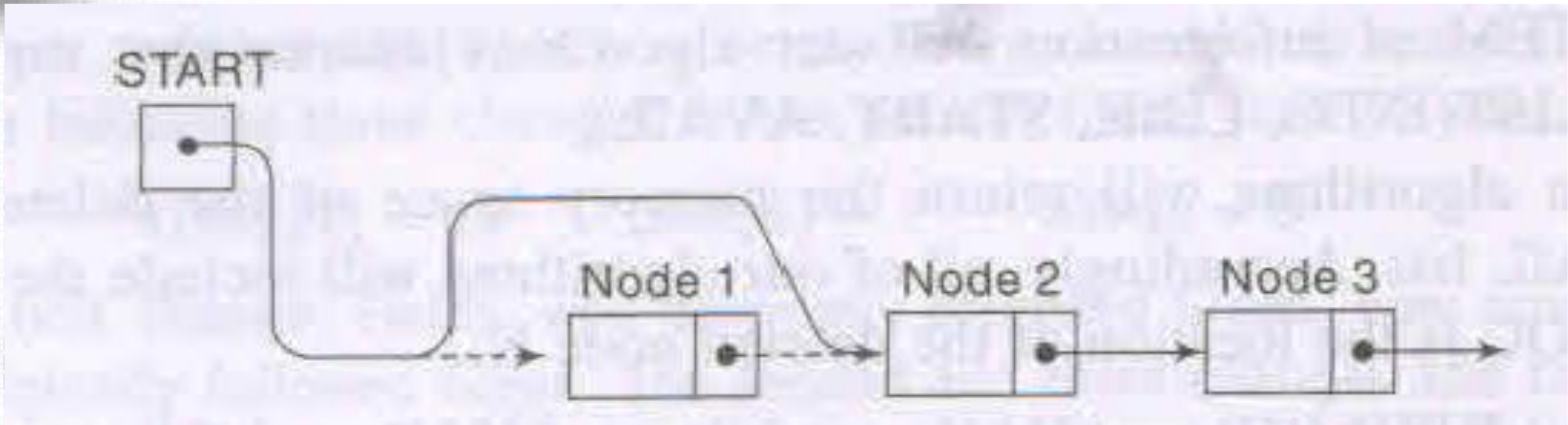


Deletion Algorithm

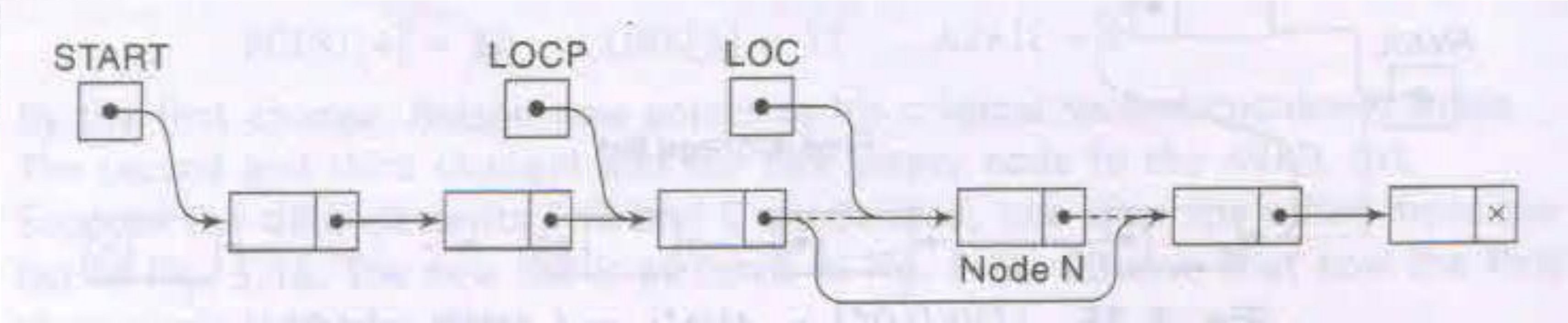


LINK[LOC] := AVAIL and then AVAIL := LOC

Deletion Algorithm – Following a given node



Deleting First Node



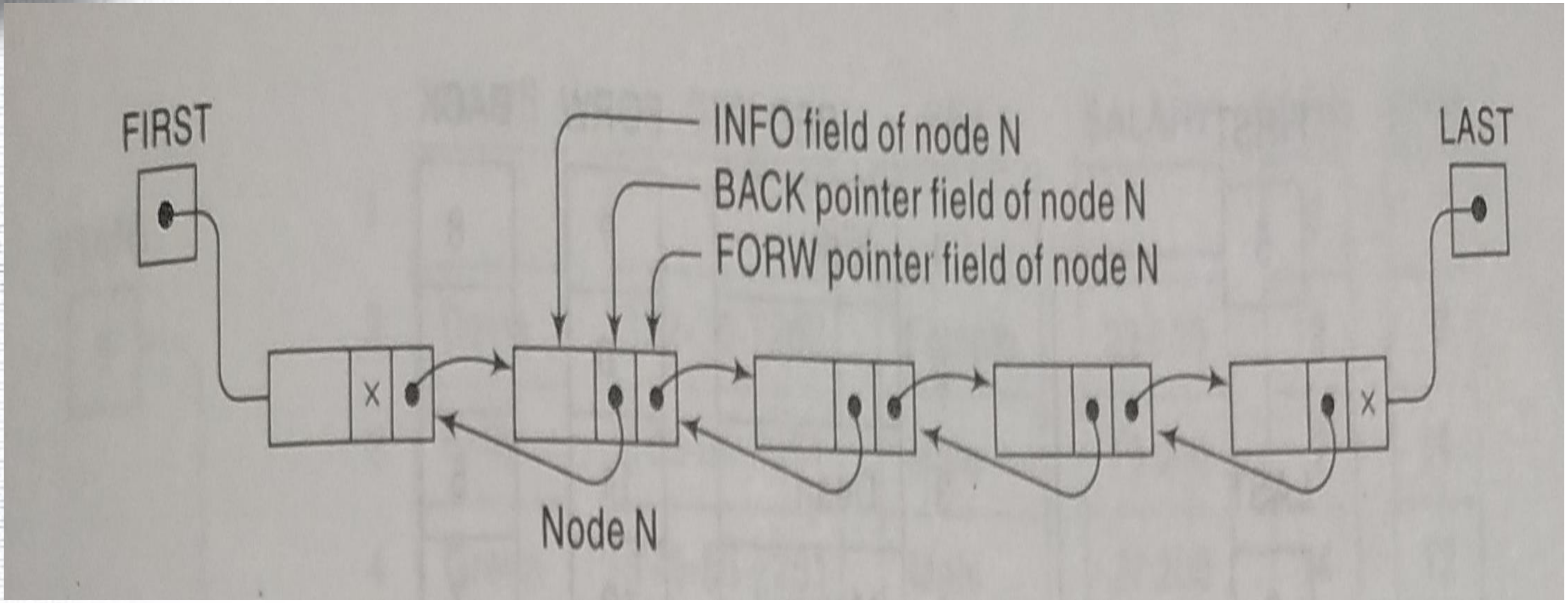
Deleting Node N when N is not the first node

Deletion Algorithm – Following a given node

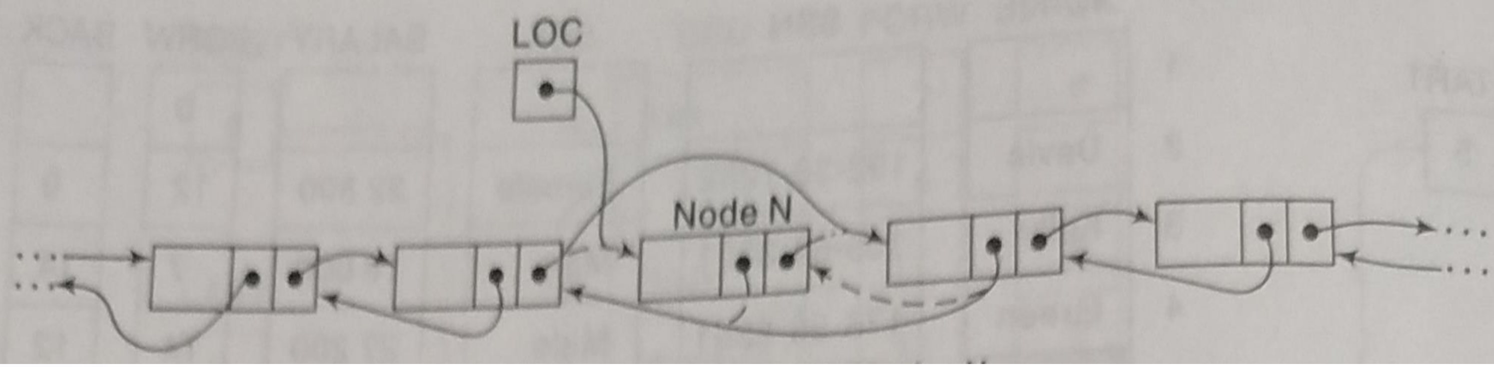
DEL(INFO, LINK, START, AVAIL, LOC, LOCP)
This algorithm deletes the node N with location LOC. LOCP is the location of the node which precedes N or, when N is the first node, LOCP = NULL.

1. If LOCP = NULL, then:
   Set START := LINK[START]. [Deletes first node.]
   Else:
   Set LINK[LOCP] := LINK[LOC]. [Deletes node N.]
   [End of If structure.]
2. [Return deleted node to the AVAIL list.]
   Set LINK[LOC] := AVAIL and AVAIL := LOC.
3. Exit.

Two way List


FIRST
LAST
INFO field of node N
BACK pointer field of node N
FORW pointer field of node N
Node N

Deletion Algorithm - Two way List



FORW[BACK[LOC]] := FORW[LOC]   and   BACK[FORW[LOC]] := BACK[LOC]

The deleted node N is then returned to the AVAIL list by the assignments:

FORW[LOC] := AVAIL   and   AVAIL := LOC

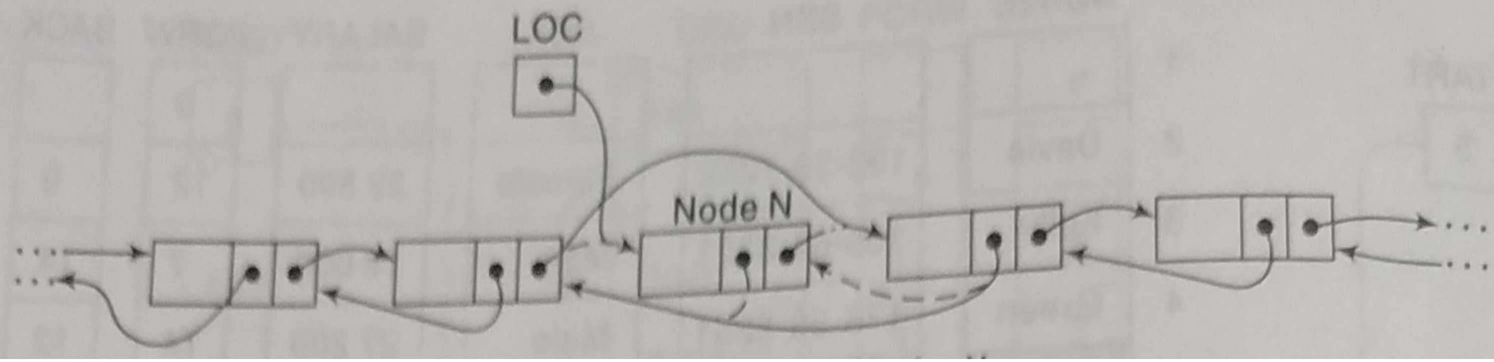
The formal statement of the algorithm follows.

Deletion Algorithm - Two way List

DELTWL(INFO, FORW, BACK, START, AVAIL, LOC)

1. [Delete node.]
   Set FORW[BACK[LOC]] := FORW[LOC] and
   BACK[FORW[LOC]] := BACK[LOC].
2. [Return node to AVAIL list.]
   Set FORW[LOC] := AVAIL and AVAIL := LOC.
3. Exit.

Deletion Algorithm - Two way List



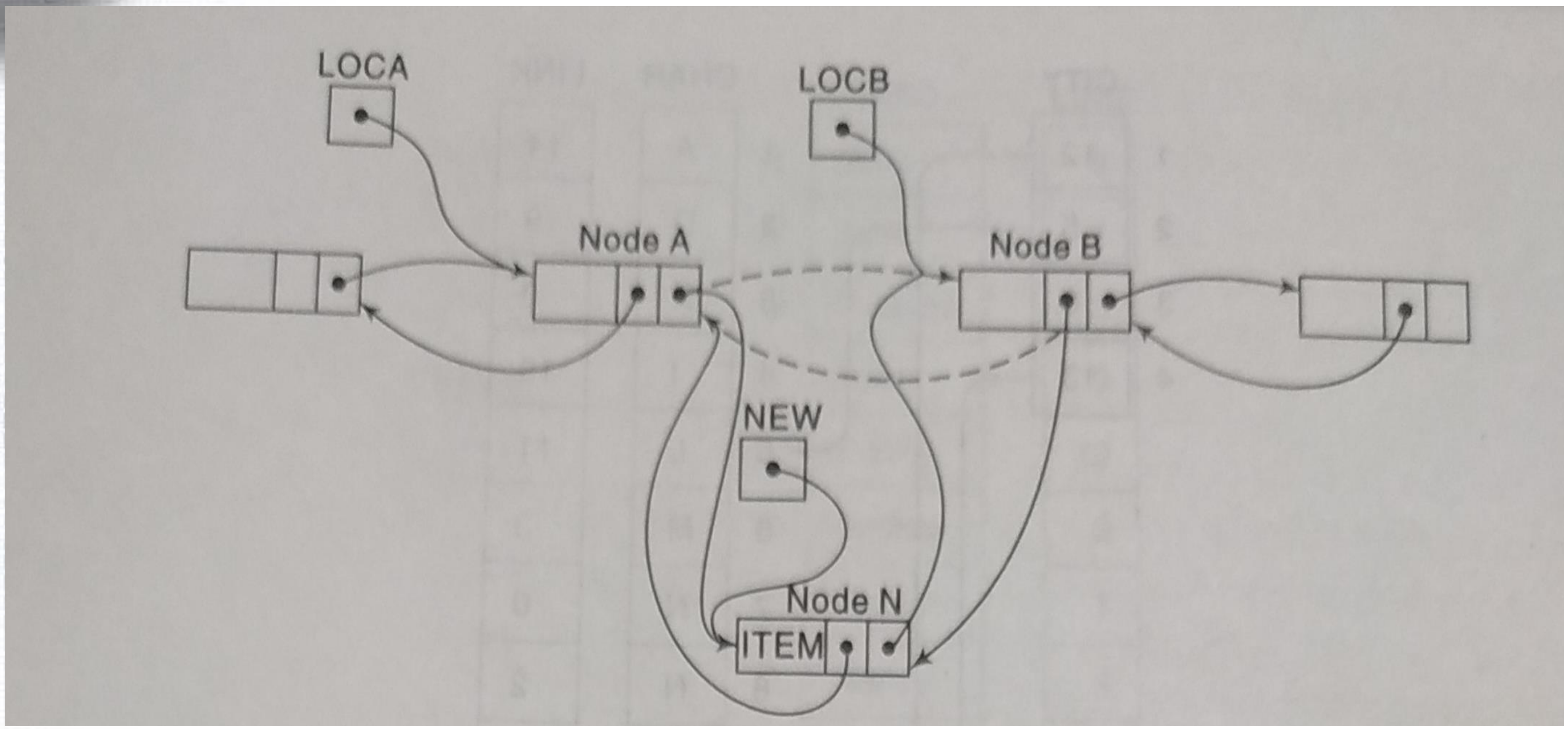
FORW[BACK[LOC]] := FORW[LOC] and BACK[FORW[LOC]] := BACK[LOC]

The deleted node N is then returned to the AVAIL list by the assignments:

FORW[LOC] := AVAIL and AVAIL := LOC

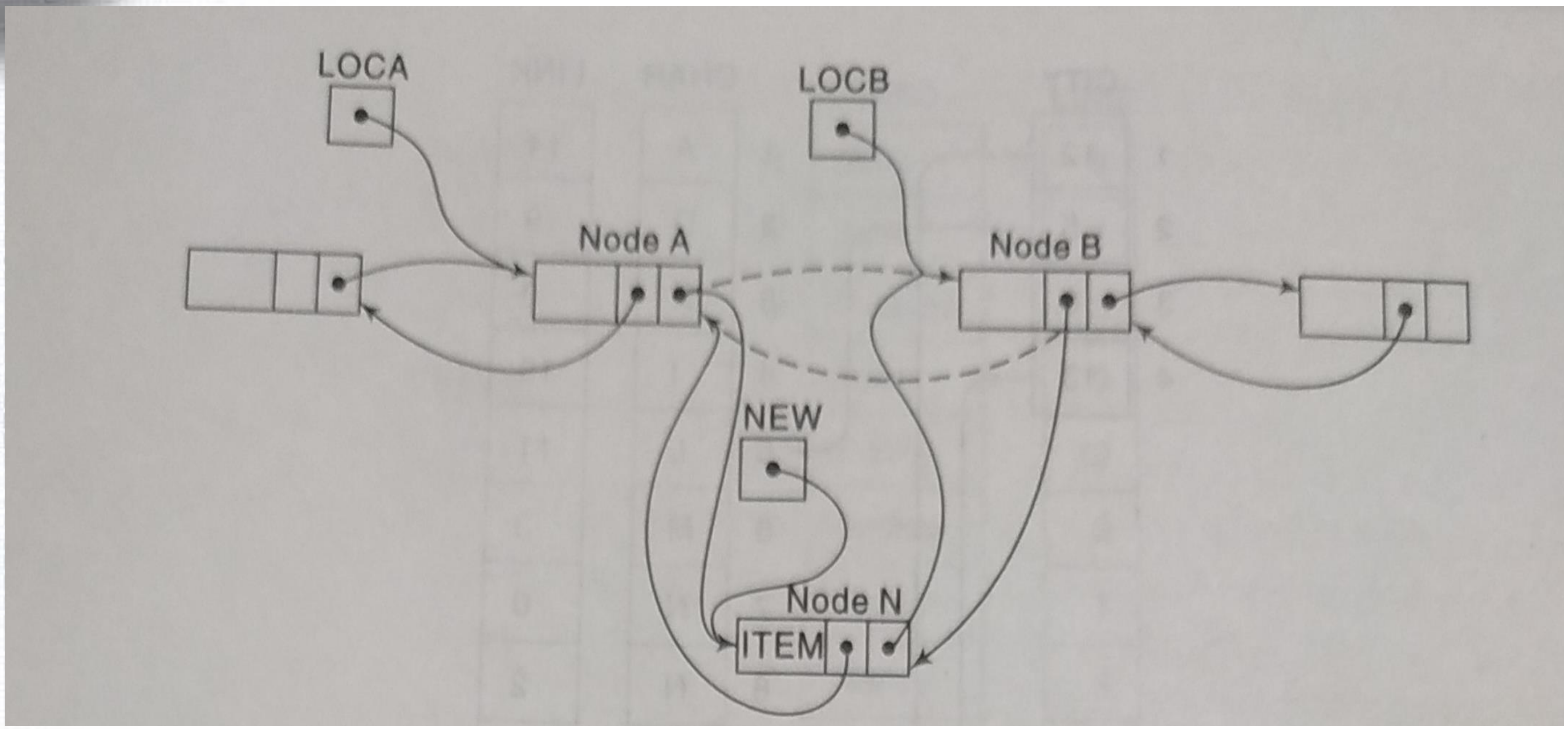
The formal statement of the algorithm follows.

Insert - Two way List



NEW := AVAIL, AVAIL := FORW[AVAIL], INFO[NEW] := ITEM

Insert - Two way List



FORW[LOCA] := NEW, FORW[NEW] := LOCB
BACK[LOCB] := NEW, BACK[NEW] := LOCA

ent of our algorithm

Insert - Two way List

INSTWL(INFO, FORW, BACK, START, AVAIL, LOCA, LOCB, ITEM)

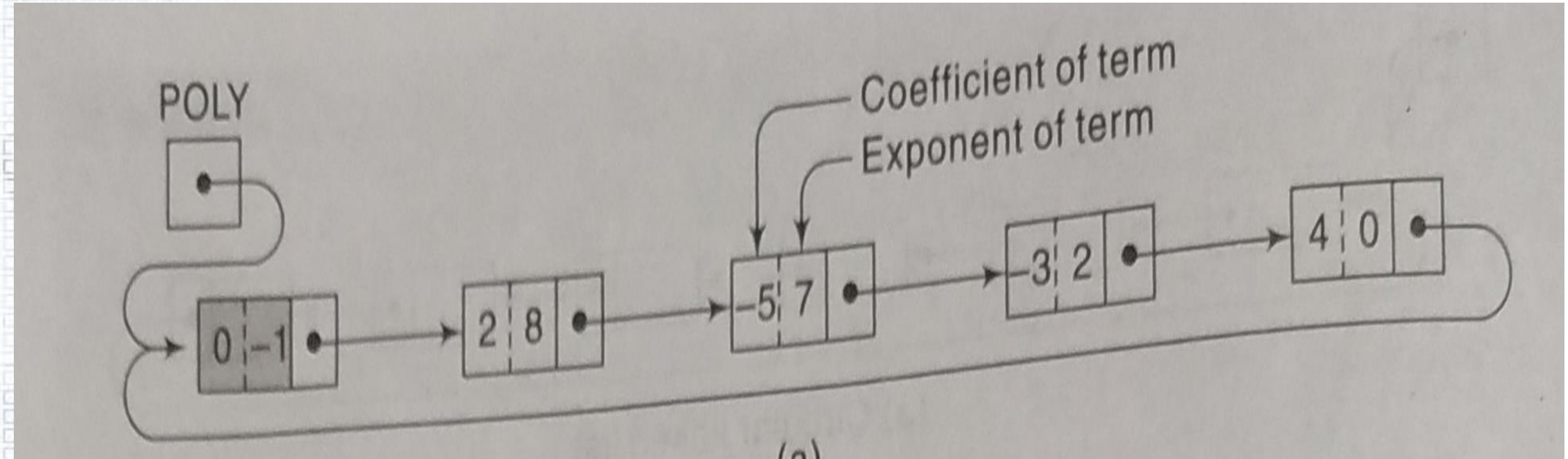
1. [OVERFLOW?] If AVAIL = NULL, then: Write: OVERFLOW, and Exit.
2. [Remove node from AVAIL list and copy new data into node.]
   Set NEW := AVAIL, AVAIL := FORW[AVAIL], INFO[NEW] := ITEM.
3. [Insert node into list.]
   Set FORW[LOCA] := NEW, FORW[NEW] := LOCB,
   BACK[LOCB] := NEW, BACK[NEW] := LOCA.
4. Exit.

Polynomials using Linked List

Let $p(x)$ denote the following polynomial in one variable (containing four nonzero terms):

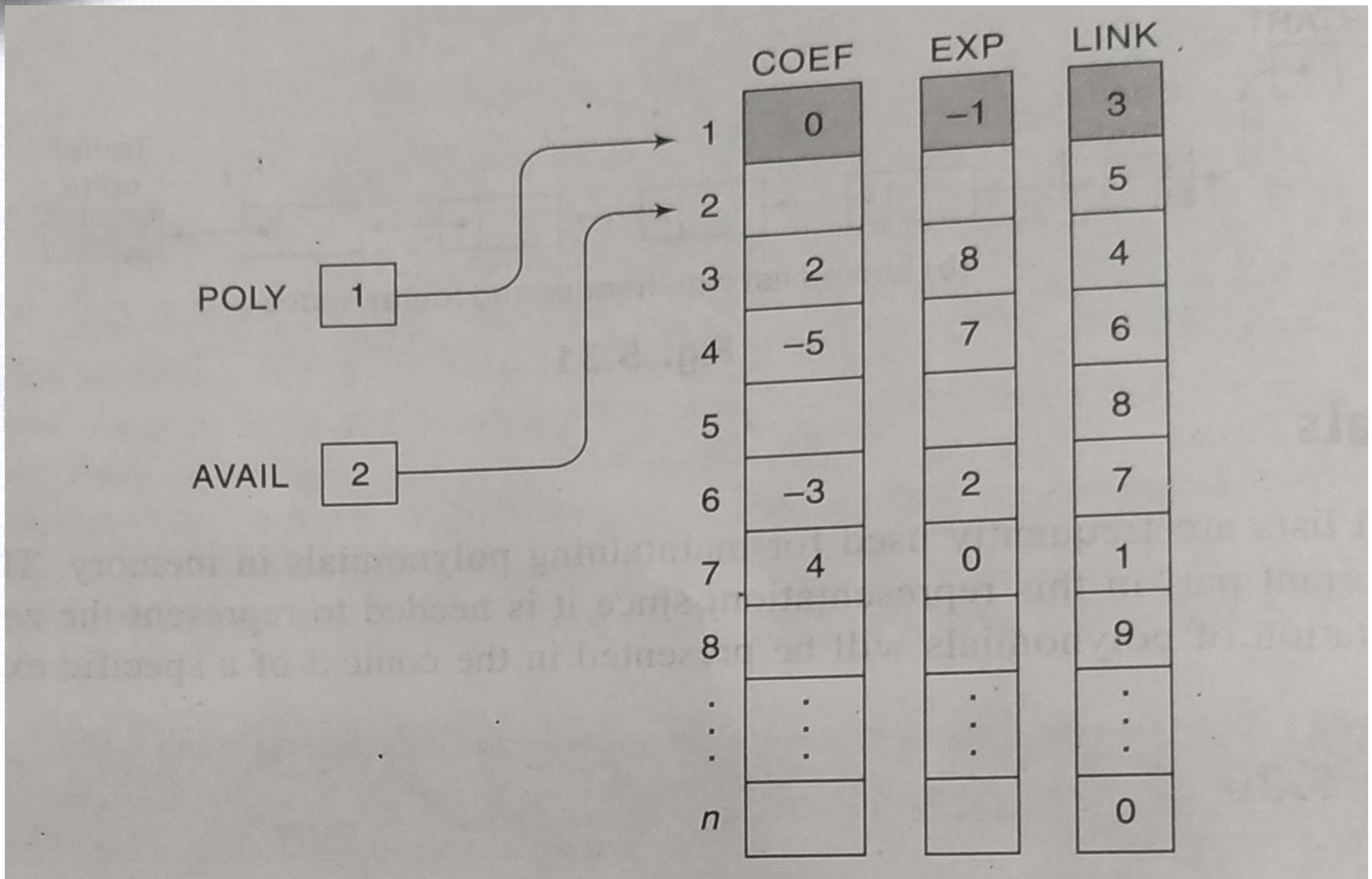
$$p(x) = 2x^8 - 5x^7 - 3x^2 + 4$$

Polynomials using Linked List



(a)

Polynomials using Linked List – Memory representation



$$p(x) = 2x^8 - 5x^7 - 3x^2 + 4$$