Linked List

Introduction

- **List**

It is a term used to refer to a linear collection of data items.

A list can be implemented either by using arrays or linked list.

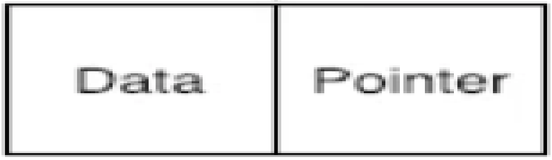
- **Drawbacks of Arrays**

- A large block of memory is occupied by an array which may not be in use and it is difficult to increase the size of an array.
- A contiguous block of memory is required

Introduction to Linked List

☐ Linked List

- Linked lists are a linear collection of data elements that stores a group of values of the same data type.
- They are also known as dynamic data structures because successive elements are not stored at contiguous memory locations.
- **NODE**

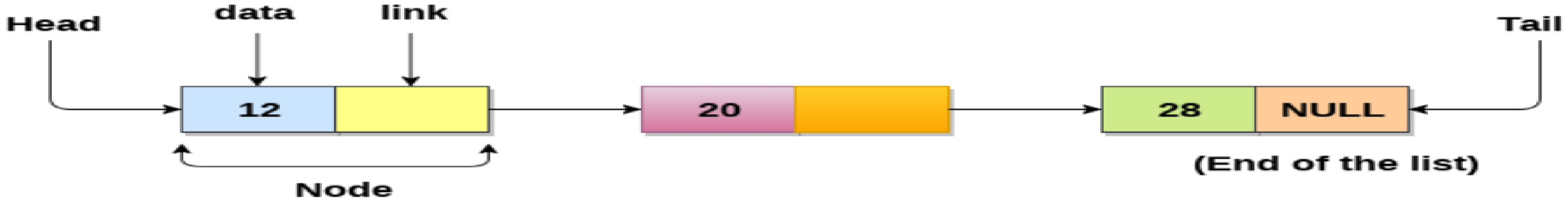


The data elements in a linked list are nodes, each of which contains a pointer field pointing to the next node.

Linked list is a data structure that contains data and link field encapsulated in a node.

Linked List Representation

- Linked list can be visualized as a chain of nodes, where every node points to the next node.



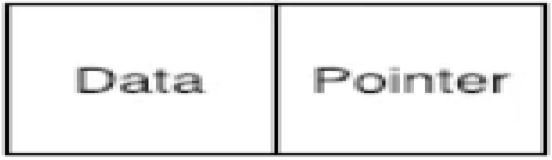
As per the above illustration, following are the important points to be considered.

- A pointer HEAD/START of the list is used to gain access to the list itself.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Linked List

NOTE

- A list that has no nodes is called a null list or empty list and is denoted by the null pointer in the variable START.
- The structure defined for single liked list is implemented as:




```
struct node
{
    Int data;
    Struct node *next;
};
```

Uses of Linked List

- The list is not required to be contiguously present in the memory. The node can reside any where in the memory and linked together to make a list. This achieves optimized utilization of space.
- list size is limited to the memory size and doesn't need to be declared in advance.
- Empty node can not be present in the linked list.
- We can store values of primitive types or objects in the singly linked list.

Why use Linked List over arrays?

Array contains following limitations:

1. The size of array must be known in advance before using it in the program.
2. Increasing size of the array is a time taking process. It is almost impossible to expand the size of the array at run time.
3. All the elements in the array need to be contiguously stored in the memory. Inserting any element in the array needs shifting of all its predecessors.

Linked list is the data structure which can overcome all the limitations of an array. Using linked list is useful because

1. It allocates the memory dynamically. All the nodes of linked list are non-contiguously stored in the memory and linked together with the help of pointers.
2. Sizing is no longer a problem since we do not need to define its size at the time of declaration. List grows as per the program's demand and limited to the available memory space.

Representation of Linked List in Memory

Let LIST→ Linked list

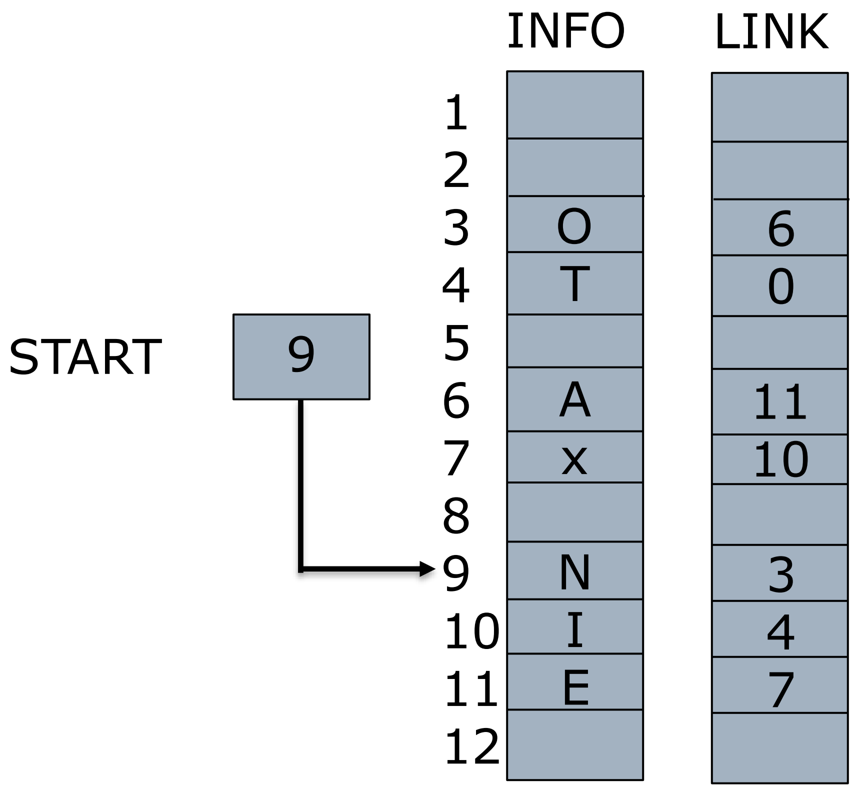
Two linear arrays are required in a LIST

INFO[K] → stores the info part

LINK[K] → stores the address of next node.

START → contains the location of the beginning of the list

NULL → indicates end of List.



Singly Linked List Operations

Basic Linked List Operations

1. Traversing a Linked List

- Processing each node of the list exactly once is known as traversing.
- 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
- PTR is pointer variable which points to the node i.e. currently being processed.
- LINK[PTR] points to the next node to be processed.

Basic Linked List Operations
Traversing a Linked List

Algorithm:

LIST: Linked list in memory

PTR: Pointer to current node

START: Pointer to first node

PROCESS: An operation i.e. to be applied on each node.

STEP 1: [Initialize PTR]
    Set PTR:=START

STEP 2: [Repeat Step 3 and 4]
    while PTR≠ NULL

STEP 3: [Apply operation on node]
    Apply PROCESS to INFO[PTR]

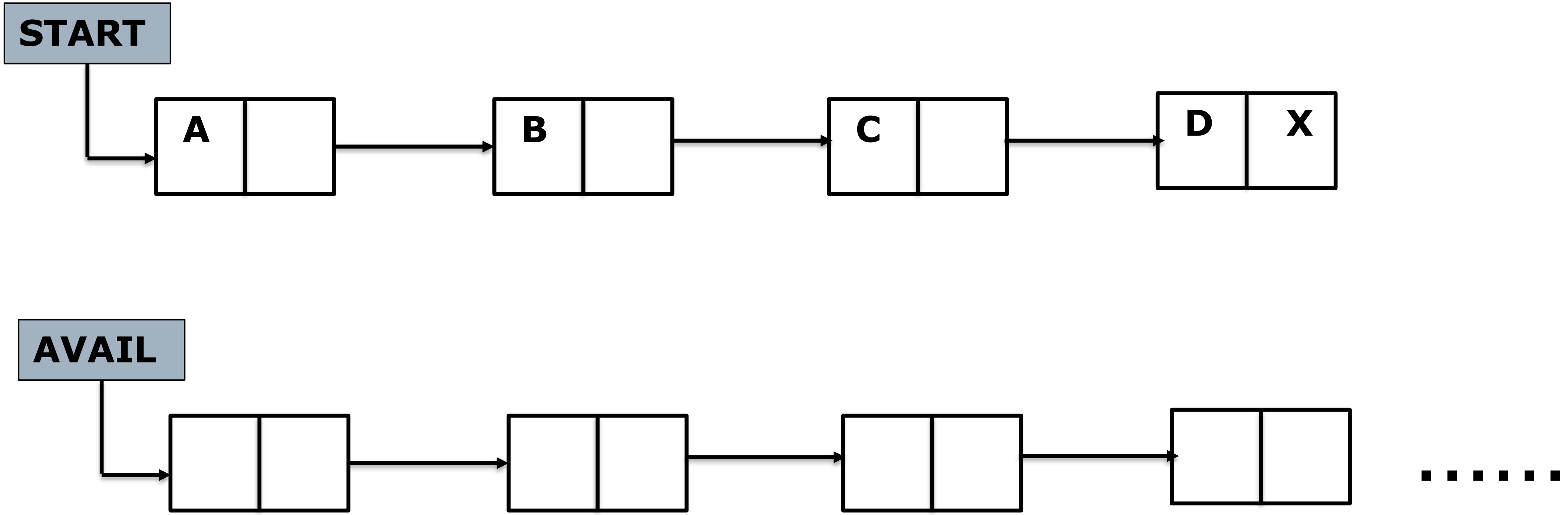
STEP 4: [Set PTR to point to the next node]
    Set PTR:=LINK[PTR]
    [End of STEP 2 loop]

STEP 5: [Finished]
    Exit

Basic Linked List Operations

2. Insertion in a Linked List

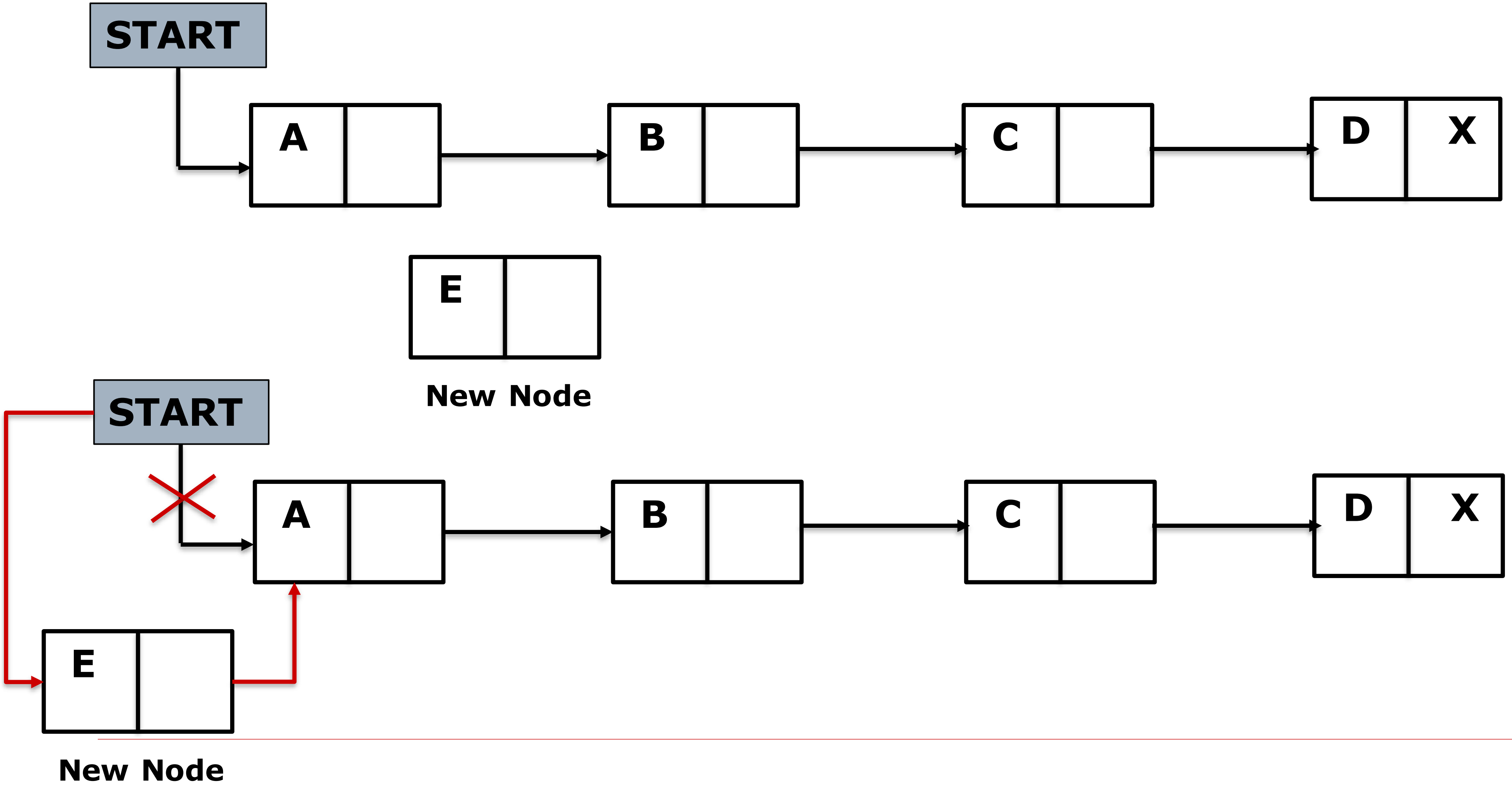
(a)Insertion at the beginning of the linked List



Basic Linked List Operations

2. Insertion in a Linked List

(a)Insertion at the beginning of the linked List



Basic Linked List Operations

2. Insertion in a Linked List

(a)Insertion at the beginning of the linked List

NOTE: Sometimes new data are to be inserted into a Data structure but there is no available space. This situation is called "Overflow" i.e. AVAIL=NULL

The term underflow refers to a situation where one wants to delete a data item from data structure that is empty i.e. START=NULL

Basic Linked List Operations

2. Insertion in a Linked List

(a)Insertion at the beginning of the linked List

Algorithm: INSFIRST(INFO, LINK, START, AVAIL, ITEM)

STEP 1: [Checking for Overflow condition]

If AVAIL=NULL then Write "Overflow" EXIT

STEP 2: [Remove first node from AVAIL List]

Set NEW:=AVAIL

AVAIL:=LINK[AVAIL]

STEP 3: [Copies new data into new node]

Set INFO[NEW]:=ITEM

STEP 4: [New Node now points to the original first node]

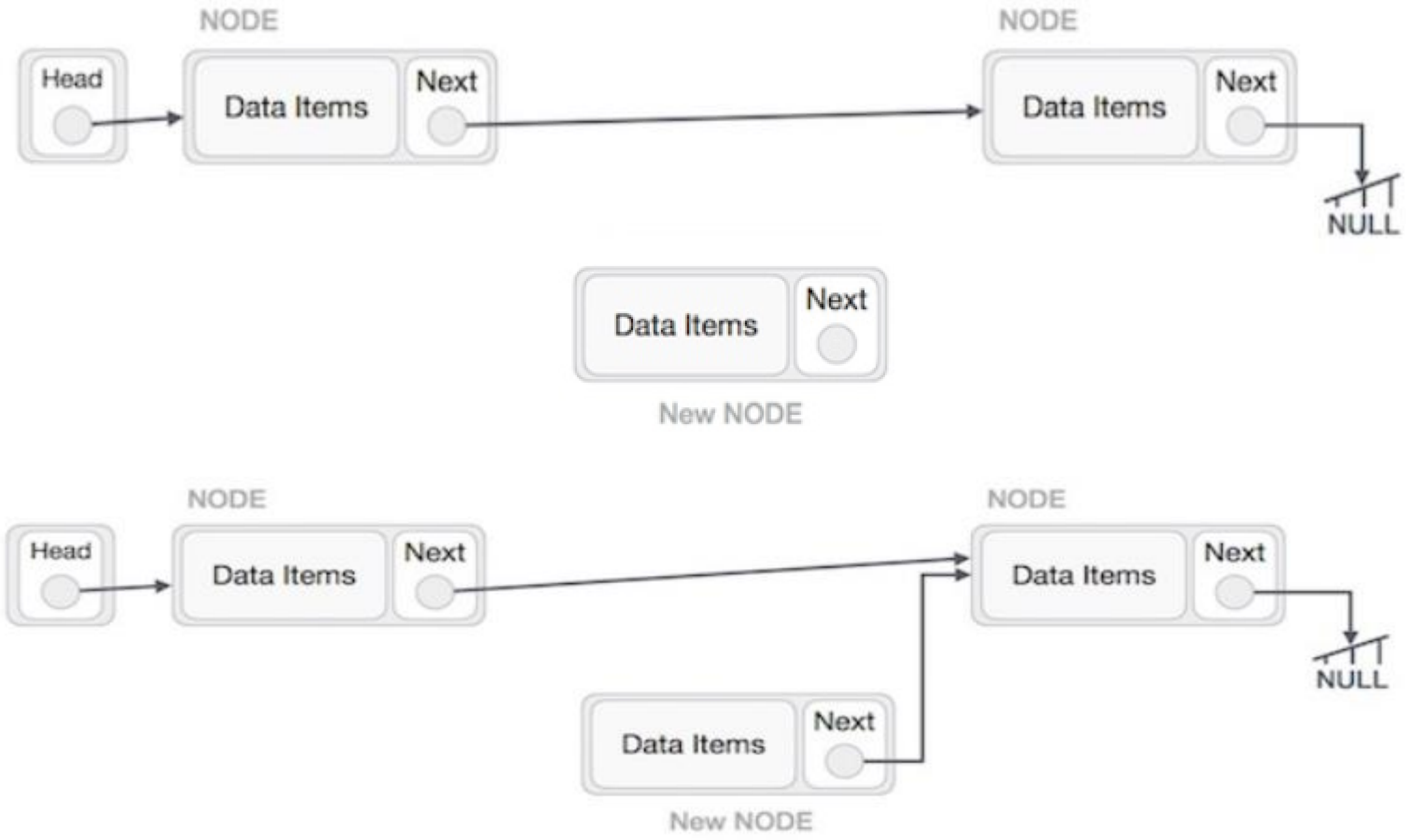
Set LINK[NEW]:=START

STEP 5: [Change START so it points to the new node]

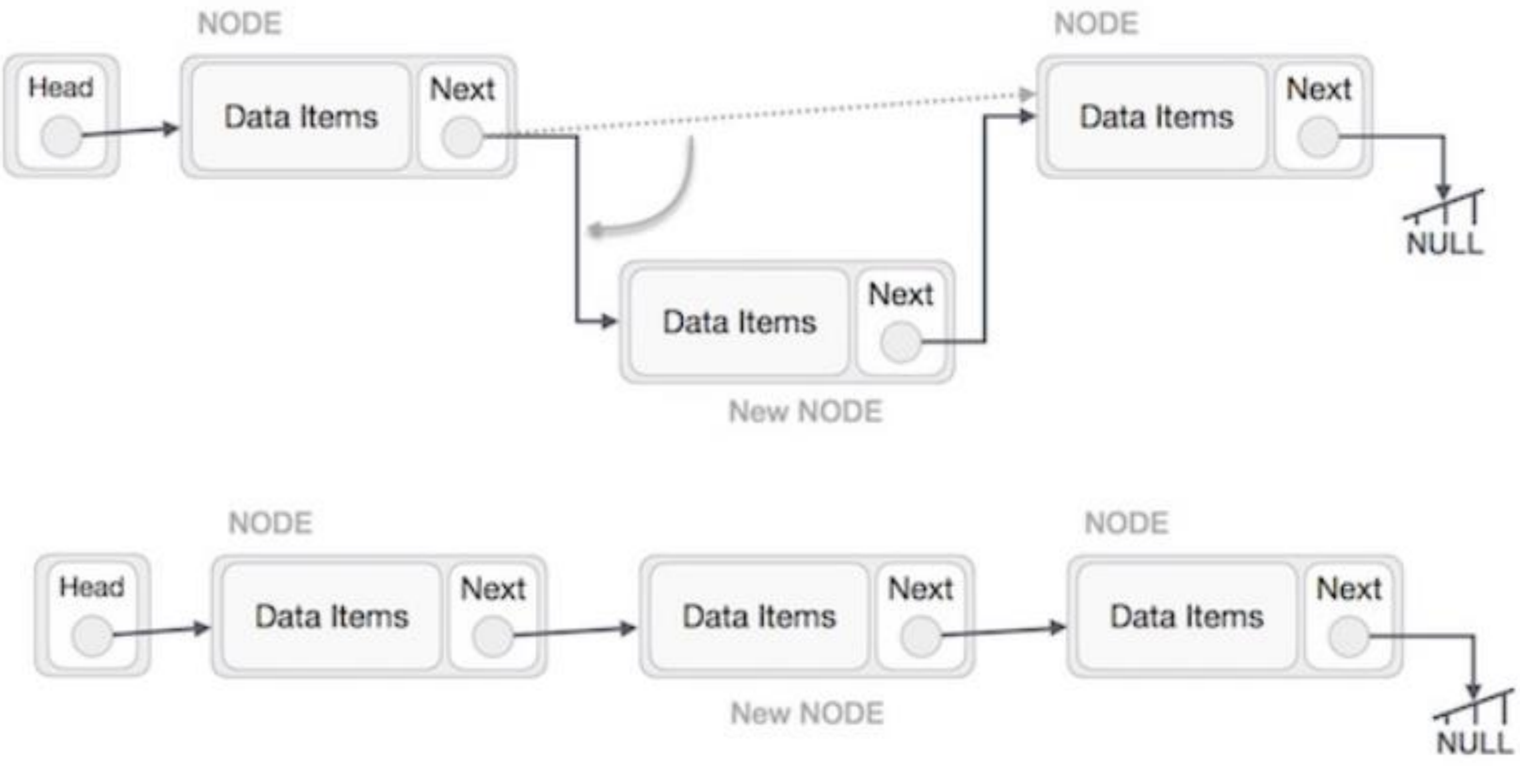
START:=NEW

STEP 6: EXIT

Basic Linked List Operations
2. Insertion in a Linked List
(b)Insertion after a given node



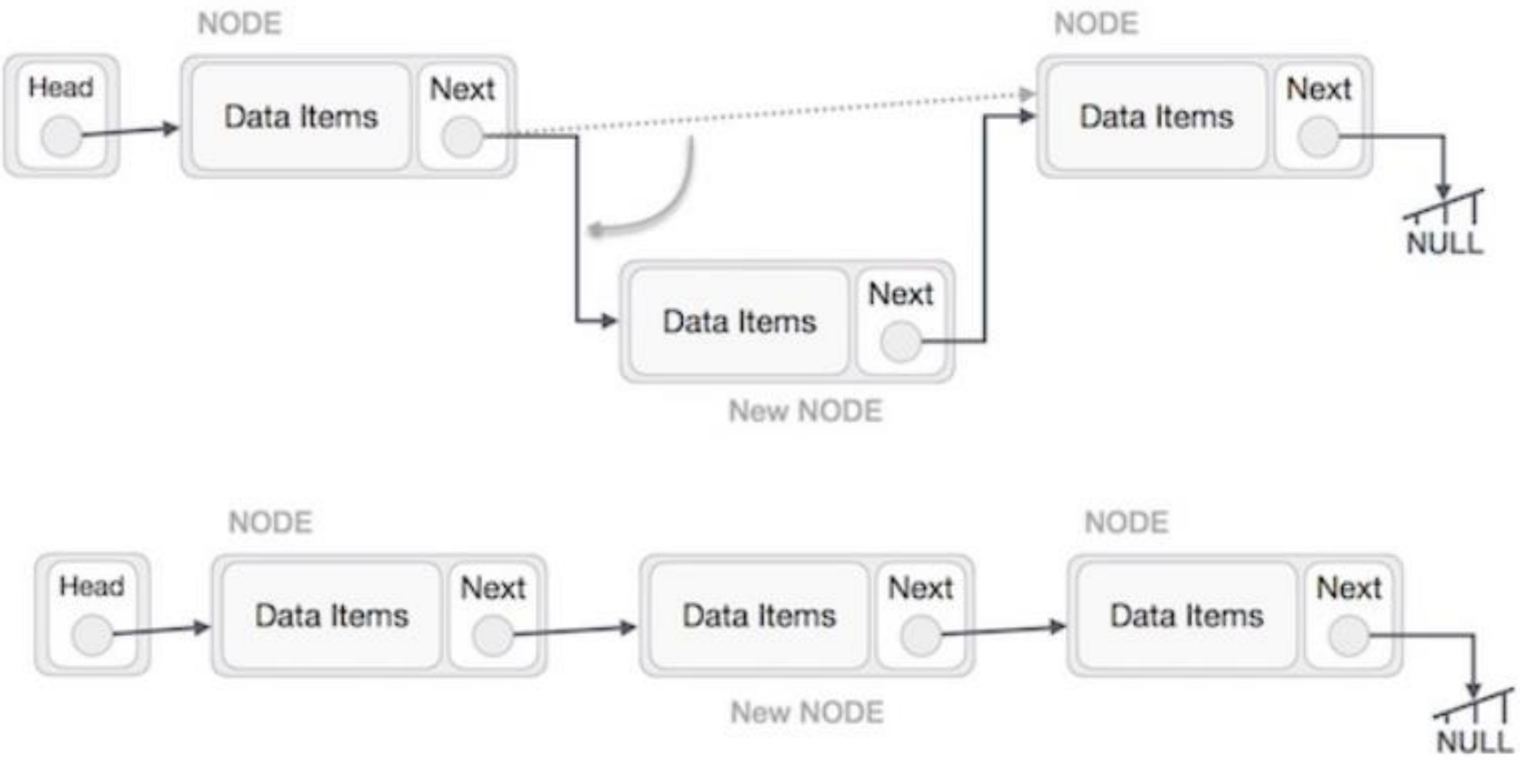
Basic Linked List Operations
2. Insertion in a Linked List
(b)Insertion after a given node



Basic Linked List Operations

2. Insertion in a Linked List

(b)Insertion after a given node



Basic Linked List Operations

2. Insertion in a Linked List

(b)Insertion after a given node

Algorithm: INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

NOTE: This algorithm inserts ITEM so that ITEM follows the node with the location LOC or inserts ITEM as the first node when LOC =NULL

STEP 1: [Checking for Overflow condition]

If AVAIL=NULL then Write “Overflow” EXIT

STEP 2: [Remove first node from AVAIL List]

Set NEW:=AVAIL

AVAIL:=LINK[AVAIL]

STEP 3: [Copies new data into new node]

Set INFO[NEW]:=ITEM

STEP 4: If LOC=NULL then

[Insert as first node]

Set LINK[NEW]:=START

START:=NEW

else

[Insert after new node with location LOC]

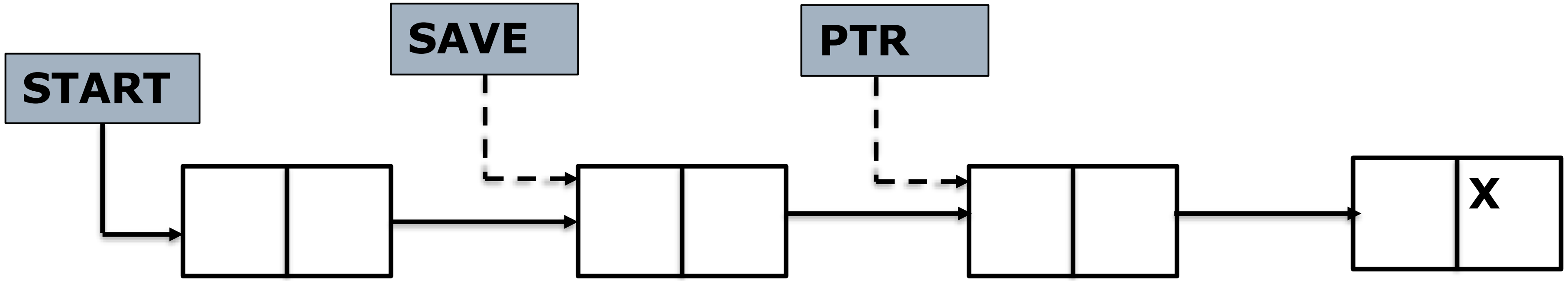
Set LINK[NEW]:=LINK[LOC]

LINK[LOC]:=NEW

[End of if structure]

STEP 5: EXIT

Basic Linked List Operations
2. Insertion in a Linked List
(c)Inserting into a sorted linked List



Basic Linked List Operations
2. Insertion in a Linked List
(c)Inserting into a sorted linked List

ITEM is to be inserted between node A and B i.e. INFO[A]<ITEM<=INFO[B]

NOTE: First find the LOC of node A.

Traverse the list using a pointer variable PTR and comparing INFO[PTR] with ITEM at each node. While traversing, keep track of the location of the preceding node by using a pointer variable SAVE i.e. SAVE:=PTR

And PTR:=LINK[PTR]

When list is empty where ITEM<=INFO[START] so LOC=NULL

Basic Linked List Operations
2. Insertion in a Linked List
(c)Inserting into a sorted linked List

Procedure for finding the location of node A

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

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

Basic Linked List Operations

2. Insertion in a Linked List

(c)Inserting into a sorted linked List

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

STEP 1: [List is Empty?]
If START=NULL then Set LOC:=NULL and Return
STEP 2: [Special Case?]
If ITEM<INFO[START] then Set LOC:=NULL and Return
STEP 3: [Initialize Pointer]
Set SAVE:=START
PTR:=LINK[PTR]
STEP 4: Repeat STEP 5 and 6 while PTR≠NULL
STEP 5: If ITEM<INFO[PTR] then
Set LOC:=SAVE and Return
STEP 6: Set SAVE:=PTR
PTR:=LINK[PTR]
[End of STEP 4 Loop]
STEP 7: Set LOC:=SAVE
STEP 8: Return

Basic Linked List Operations

2. Insertion in a Linked List

(c)Inserting into a sorted linked List

Algorithm: INSERT(INFO, LINK, START, AVAIL,ITEM)

This algorithm inserts the ITEM into a sorted linked list

STEP 1: [Use Procedure to find the location of node preceding ITEM]

Call FINDA(INFO, LINK, START, ITEM, LOC)

STEP 2: [Use Algorithm to insert ITEM after node with location LOC]

Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM)

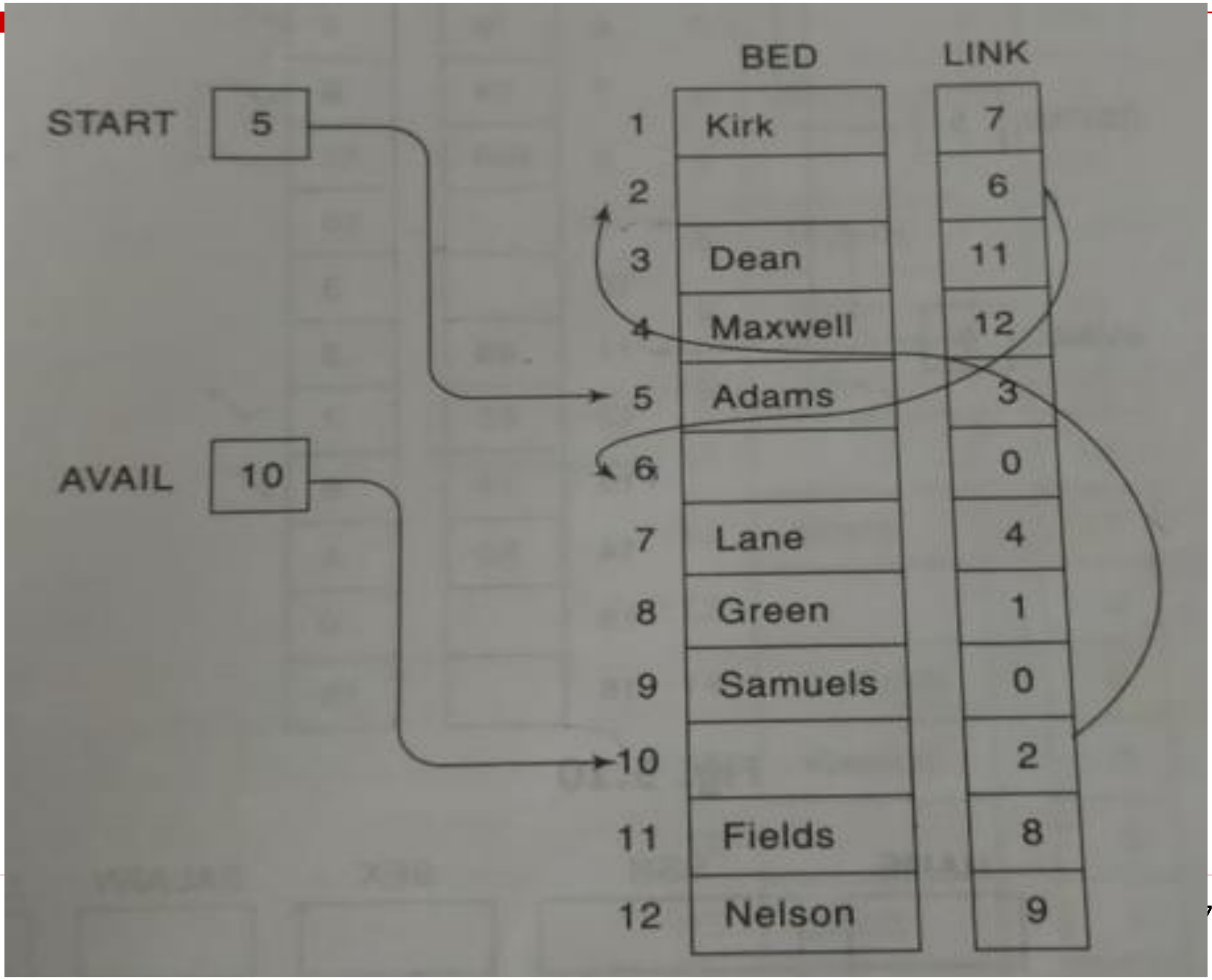
STEP 3: EXIT

Example

- Consider the alphabetized list of patients in the given Figure. Determine the changes if "Jones" is added to the list of patients.
- Solution:

  Given ITEM="Jones"

  INFO=BED



Example Solution

(A)FINDA(BED, LINK, START, ITEM, LOC)

Step 1: Since START≠NULL, Control shall be transferred to Step 2
Step 2: Since BED[5]=“Adams” Therefore BED[5]<ITEM Control shall be transferred to Step 3
Step 3: SAVE=5 and PTR=LINK[5]=3
Step 4: Step 5 and Step 6 are repeated as follows

- **(a) BED[3]=“Dean”<“Jones” so SAVE=3 and PTR= LINK[3]=11**
- **(b) BED[11]=“Fields”<“Jones” SAVE=11 and PTR=LINK[11]=8**
- **(c) BED[8]=“Green”<“Jones” SAVE=8 and PTR=LINK[8]=1**
- **(d) Since BED[1]=“Kirk”>“Jones” Therefore LOC=SAVE=8 and Return**

(B)INSLOC(BED, LINK, START, AVAIL, LOC,ITEM)

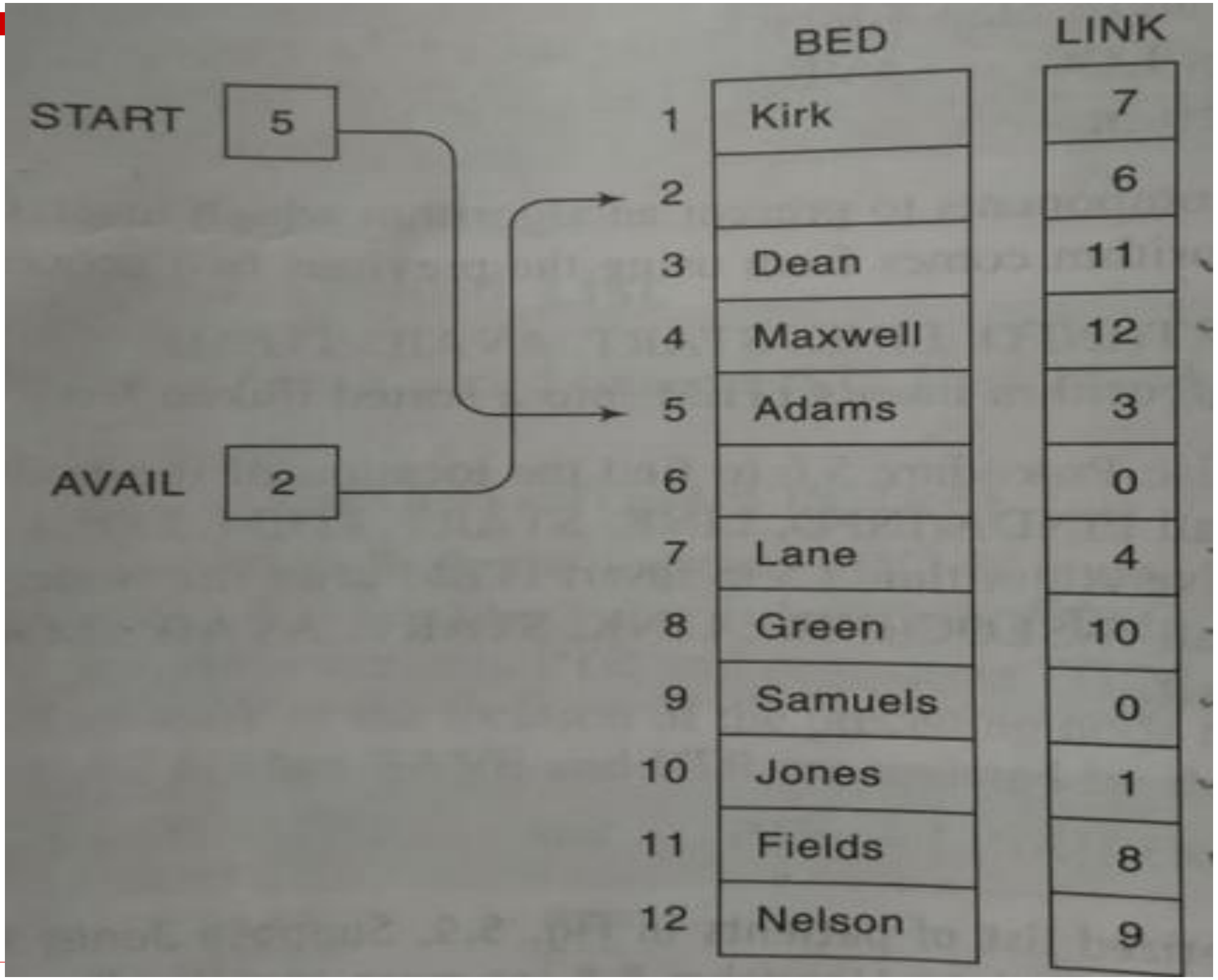
Step 1: Since AVAIL≠NULL Control is transferred to Step 2
Step 2: NEW=10 and AVAIL=LINK[10]=2
Step 3: BED[10]=“Jones”
Step 4: LOC≠NULL We have

LINK[10]=LINK[8]=1 and LINK[8]=NEW=10

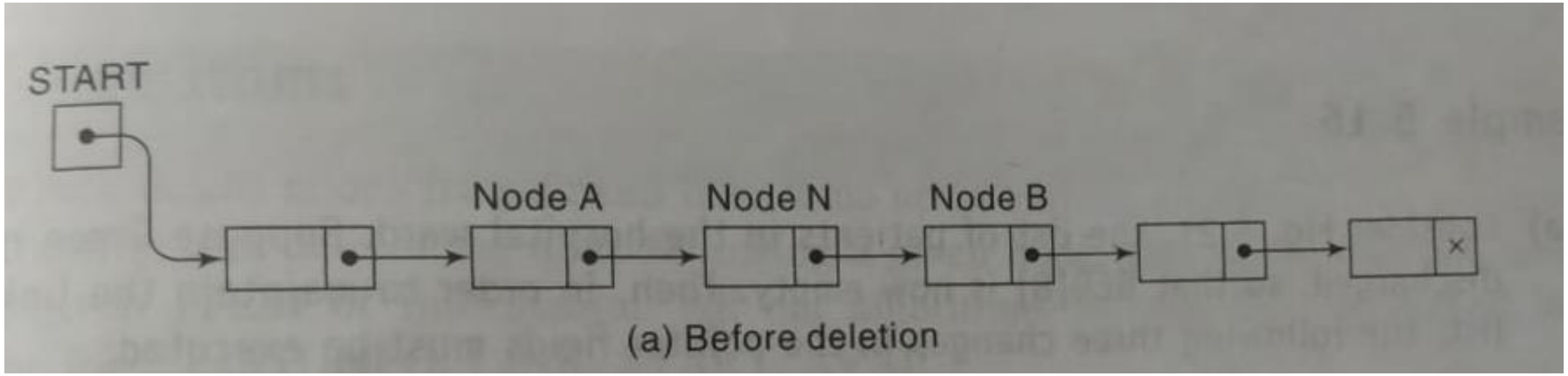
Step 5: Exit

Example Solution

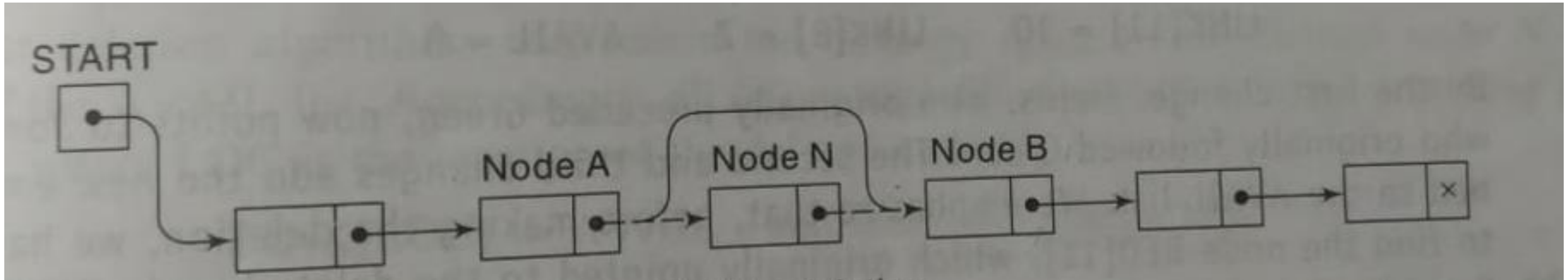
- Updated list after insertion of "Jones"



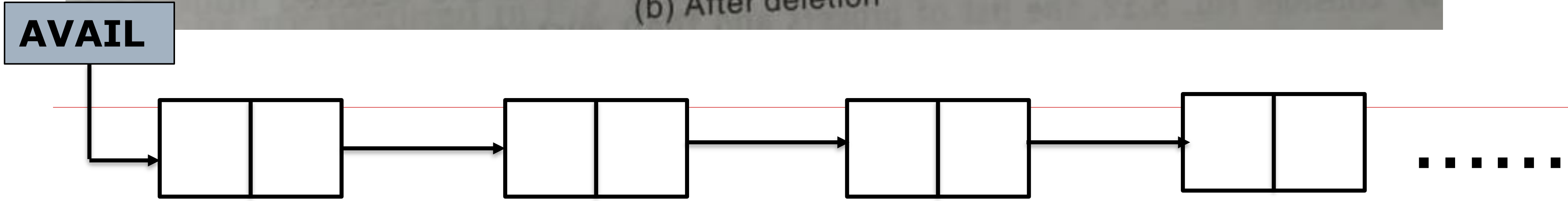
Deletion from a Linked List



(a) Before deletion



(b) After deletion



Deletion from a Linked List

Two Special Cases:

1. **If the deleted Node N is the first node in the list then START will point to Node B**
2. **If the deleted node N is the last node in the list then Node A will contain NULL pointer in the LINK part.**

(a)Deletion a Node after a Given Node

Let LIST be a list in memory.

Suppose LOC be the location of Node N in the LIST.

Let LOCP be the location of the Node preceding Node N.

When Node N is the first node then LOCP=NULL

Algorithm: DEL(INFO, LINK, START, AVAIL, LOC, LOCP)

Step 1: if LOCP=NULL Then

Set START:=Link[START] [Deletes First Node]

else

Set LINK[LOCP]:=LINK[LOC] [Deletes Node N]

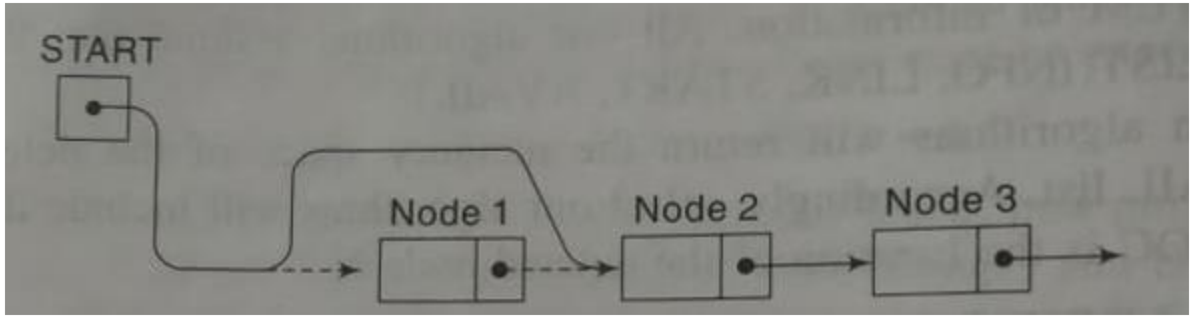
Step 2: [Return deleted node to the AVAIL list]

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

Step 3: [Finished]

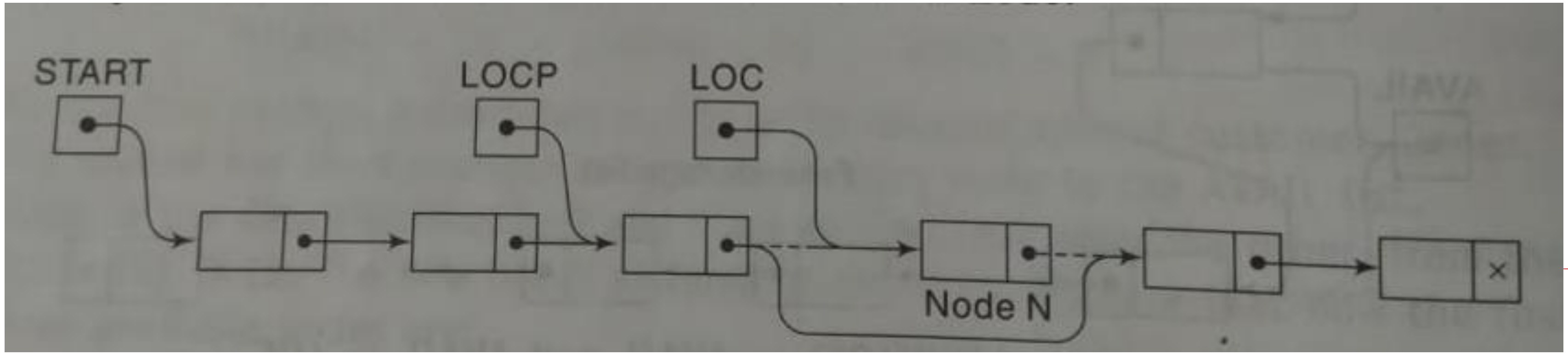
Exit

To Delete the First Node



START := LINK[START]

To Delete the Node N when N is not the First node



LINK[LOCP] := LINK[LOC]

(b)Deletion a Node with a Given ITEM of Information

NOTE: Find the LOC of Node N containing ITEM and LOCP of the node preceding Node N.

Procedure: FINDB(INFO, LINK, START, ITEM, LOC, LOCP)

Step 1: [Empty List?]
    if START=NULL Then
        Set LOC:=NULL and LOCP:=NULL and Return
    [End of If Structure]
Step 2: [ITEM is first Node?]
    if INFO[START] =ITEM then
        Set LOC:=START and LOCP:=NULL and Return
    [End of If structure]
Step 3: [Initialize Pointer]
    Set SAVE:=START and PTR:=LINK[START]

Step 4: Repeat Step 5 and Step 6 while PTR≠NULL
Step 5: If INFO[PTR]=ITEM then
        Set LOC:=PTR and LOCP:=SAVE and Return
    [End of if structure]
Step 6: [Update Pointers]
    Set SAVE:=PTR and PTR:=LINK[PTR]
    [End of Step 4 Loop]
Step 7: Set LOC:=NULL [Search Unsuccessful]
Step 8: Return

(b)Deletion a Node with a Given ITEM of Information(Contd..)

Algorithm: DELETE(INFO, LINK, START, AVAIL, ITEM)

Step 1: [Use Procedure FINDB() to find the location of N and its preceding node]
  Call FINDB(INFO, LINK, START, ITEM, LOC, LOCP)

Step 2: if LOC=NULL then
  Write "Item not in list"   Exit

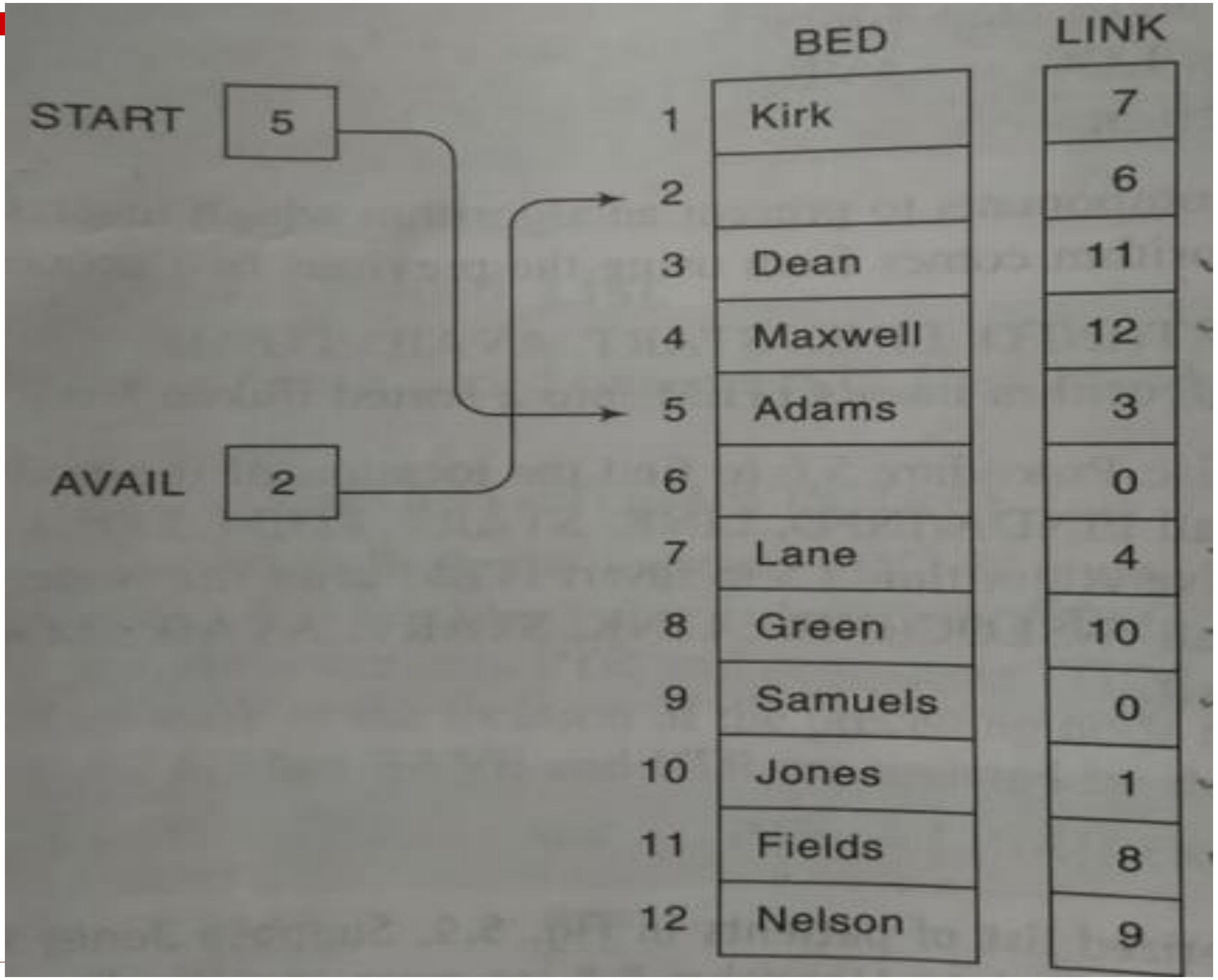
Step 3: [Delete Node]
  if LOCP=NULL then
    Set START:=LINK[START]   [Deletes first node]
  else
    Set LINK[LOCP]:=LINK[LOC]
  [End of If structure]

Step 4: [Return deleted node to the AVAIL list]
  Set LINK[LOC]:=AVAIL AND AVAIL:=LOC

Step 5: [Finished]
  Exit

Example

- Consider the list of patients in the given Figure.
- Delete the patient “Green” when he is discharged.



Example Solution

- Here ITEM="Green", INFO=BED, START=5, AVAIL=2
- **FINDB(BED, LINK, START, ITEM, LOC, LOCP)**

1. Since START≠NULL Control is transferred to Step 2
2. Since BED[5]="Adams" ≠ "Green" Control is transferred to Step 3
3. Set SAVE=5 and PTR=LINK[5]=3
4. Step 5 and 6 are repeated as follows:

    (a) BED[3]="Dean" ≠ "Green" Therefore

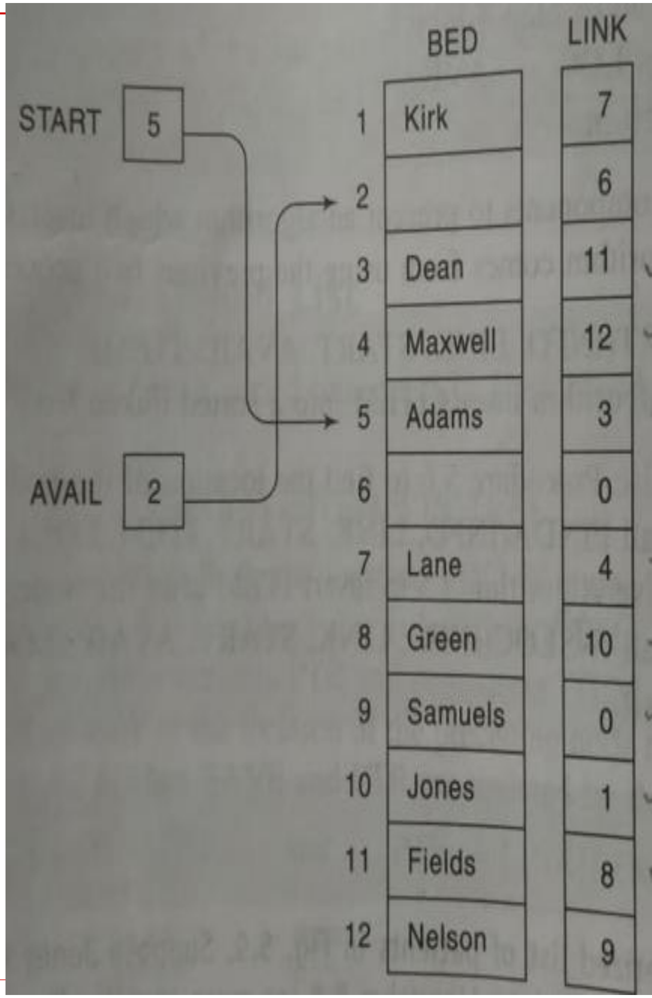
    SAVE=3 PTR=LINK[3]=11

    (b) BED[11]="Fields" ≠ "Green" Therefore

    SAVE=11 PTR=LINK[11]=8

    (c) BED[8]="Green" = "Green" Therefore

    LOC=PTR=8, LOCP=SAVE=11 and Return



Example Solution(Contd..)

- **DELETE(BED, LINK, START, AVAIL, ITEM)**
  1. Call FINDB(BED, LINK, START, ITEM, LOC, LOCP)
     Hence LOC=8 and LOCP=11
  2. Since LOC≠NULL control is transferred to Step 3
  3. Since LOCP≠NULL then
     LINK[11]=LINK[8]=10
  4. LINK[8]=2 AND AVAIL=8
  5. Exit

NOTE: The updated list has been shown in Figure