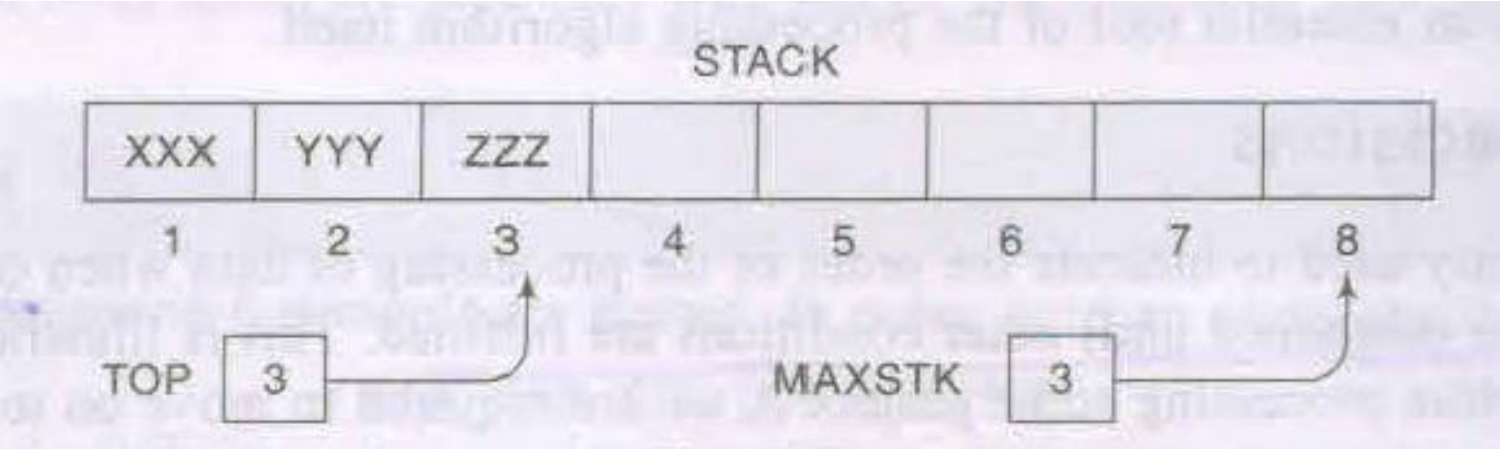
Stacks and queues

Array Representation of Stack



- Stack Overflow
- Stack Underflow

PUSH and POP Operations

PUSH(STACK, TOP, MAXSTK, ITEM)
This procedure pushes an ITEM onto a stack.

1. [Stack already filled?]
   If TOP = MAXSTK, then: Print: OVERFLOW, and Return.
2. Set TOP := TOP + 1. [Increases TOP by 1.]
3. Set STACK[TOP] := ITEM. [Inserts ITEM in new TOP position.]
4. Return.

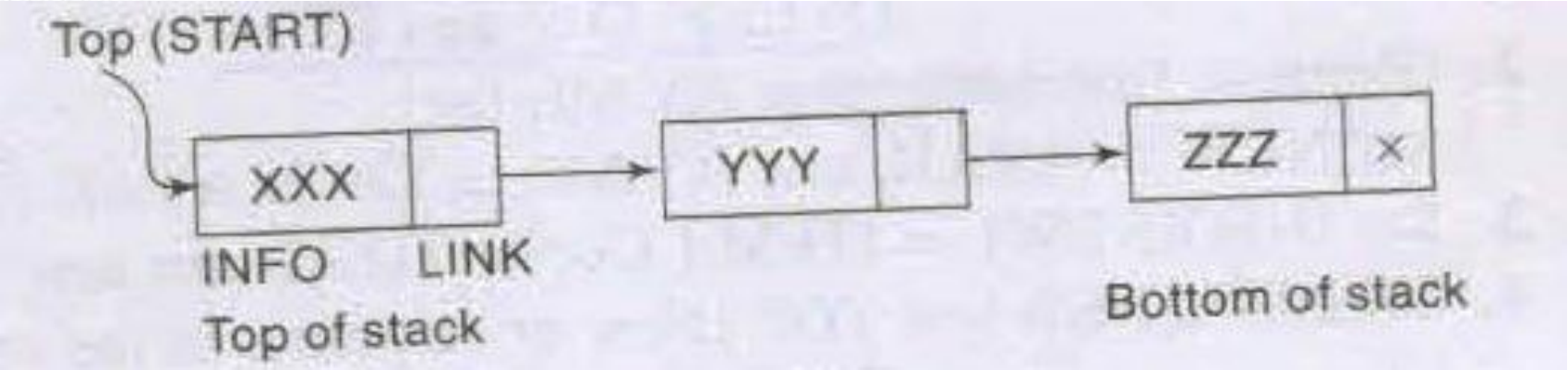
PUSH and POP Operations

POP(STACK, TOP, ITEM)
This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. [Stack has an item to be removed?]
   If TOP = 0, then: Print: UNDERFLOW, and Return.
2. Set ITEM := STACK[TOP]. [Assigns TOP element to ITEM.]
3. Set TOP := TOP – 1. [Decreases TOP by 1.]
4. Return.

Note: The value of TOP is changed before the insertion in PUSH but the value of TOP is changed after deletion in POP

Linked representation of STACK



Linked representation of STACK

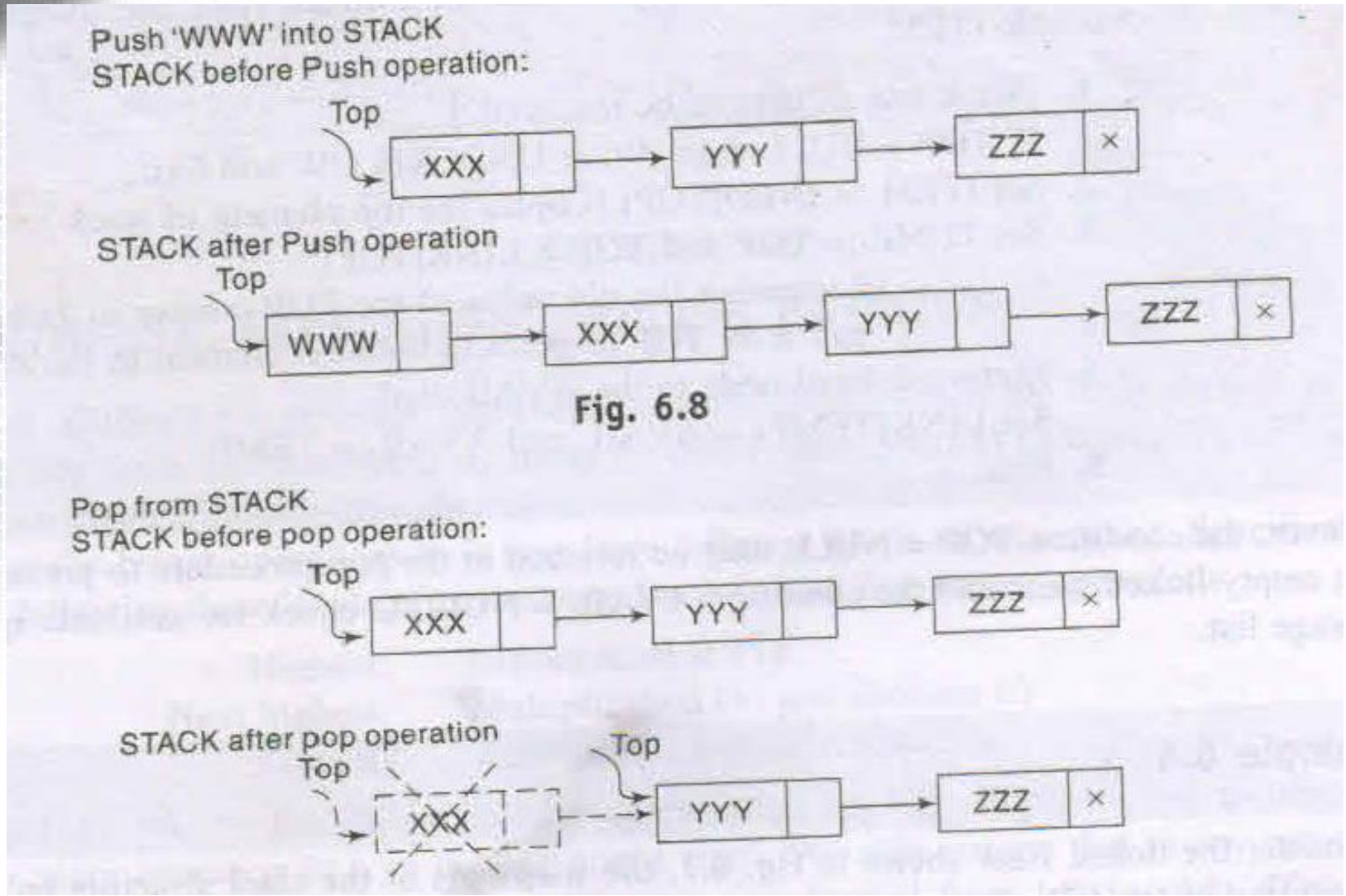


Fig. 6.8

Linked representation of STACK

PUSH_LINKSTACK(INFO, LINK, TOP, AVAIL, ITEM)
This procedure pushes an ITEM into a linked stack

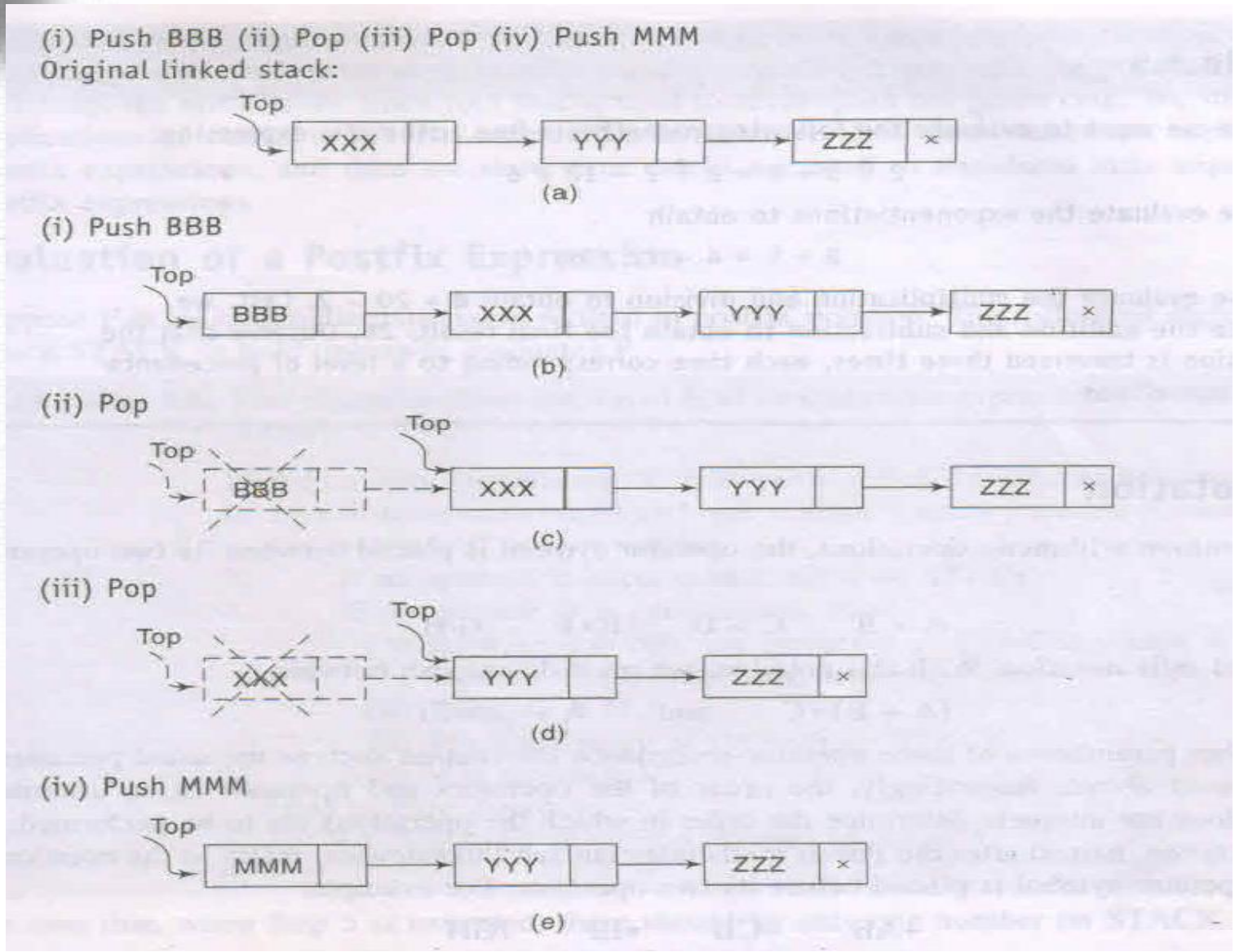
1. [Available space?] If AVAIL = NULL, then Write OVERFLOW and Exit
2. [Remove first node from AVAIL list]
   Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM [Copies ITEM into new node]
4. Set LINK[NEW] := TOP [New node points to the original top node in the stack]
5. Set TOP = NEW [Reset TOP to point to the new node at the top of the stack]
6. Exit.

Linked representation of STACK

POP_LINKSTACK(INFO, LINK, TOP, AVAIL, ITEM)
This procedure deletes the top element of a linked stack and assigns it to the variable ITEM

1. [Stack has an item to be removed?]
   IF TOP = NULL then Write: UNDERFLOW and Exit.
2. Set ITEM := INFO[TOP] [Copies the top element of stack into ITEM]
3. Set TEMP := TOP and TOP = LINK[TOP]
   [Remember the old value of the TOP pointer in TEMP
   and reset TOP to point to the next element in the stack]
4. [Return deleted node to the AVAIL list]
   Set LINK[TEMP] = AVAIL and AVAIL = TEMP.
5. Exit.

Linked representation of STACK



QUICKSORT, an application of STACK

Suppose A is the following list of 12 numbers:

(44) 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, (66)

(22) 33, 11, 55, 77, 90, 40, 60, 99, (44) 88, 66

22, 33, 11, (44) 77, 90, 40, 60, 99, (55) 88, 66

22, 33, 11, (40) 77, 90, (44) 60, 99, 55, 88, 66

22, 33, 11, (40) 44, 90, (77) 60, 99, 55, 88, 66

22, 33, 11, 40, (44) 90, 77, 60, 99, 55, 88, 66

First sublist Second sublist

Polish Notation: Evaluation of Postfix expression

: This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis ")" at the end of P. [This acts as a sentinel.]
2. Scan P from left to right and repeat Steps 3 and 4 for each element of P until the sentinel ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator $\otimes$ is encountered, then:
   (a) Remove the two top elements of STACK, where A is the top element and B is the next-to-top element.
   (b) Evaluate B $\otimes$ A.
   (c) Place the result of (b) back on STACK.
   [End of If structure.]
   [End of Step 2 loop.]
5. Set VALUE equal to the top element on STACK.
6. Exit.

Polish Notation: Evaluation of Postfix expression

Symbol Scanned		STACK
(1)	5	5
(2)	6	5, 6
(3)	2	5, 6, 2
(4)	+	5, 8
(5)	*	40
(6)	12	40, 12
(7)	4	40, 12, 4
(8)	/	40, 3
(9)	–	37
(10)	)	

5 * (6 + 2) - 12 / 4

P: 5, 6, 2, +, *, 12, 4, /, -

Polish Notation: Transforming Infix into Postfix

POLISH(Q, P)
Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto STACK, and add ")" to the end of Q.
2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty:
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it onto STACK.
5. If an operator $\otimes$ is encountered, then:

Polish Notation: Transforming Infix into Postfix

(a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than ⊗.

(b) Add ⊗ to STACK.

[End of If structure.]

6. If a right parenthesis is encountered, then:

(a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.

(b) Remove the left parenthesis. [Do not add the left parenthesis to P.]

[End of If structure.]

[End of Step 2 loop.]

7. Exit.

Polish Notation: Transforming Infix into Postfix

Symbol Scanned		STACK	Expression P
(1)	A	(	A
(2)	+	(+	A
(3)	(	(+ (	A
(4)	B	(+ (	A B
(5)	*	(+ (*	A B
(6)	C	(+ (*	A B C
(7)	−	(+ (−	A B C *
(8)	(	(+ (− (	A B C *
(9)	D	(+ (− (	A B C * D
(10)	/	(+ (− (/	A B C * D
(11)	E	(+ (− (/	A B C * D E
(12)	↑	(+ (− (/ ↑	A B C * D E
(13)	F	(+ (− (/ ↑	A B C * D E F
(14)	)	(+ (−	A B C * D E F ↑ /
(15)	*	(+ (− *	A B C * D E F ↑ /
(16)	G	(+ (− *	A B C * D E F ↑ / G
(17)	)	(+	A B C * D E F ↑ / G * −
(18)	*	(+ *	A B C * D E F ↑ / G * −
(19)	H	(+ *	A B C * D E F ↑ / G * − H
(20)	)		A B C * D E F ↑ / G * − H * +

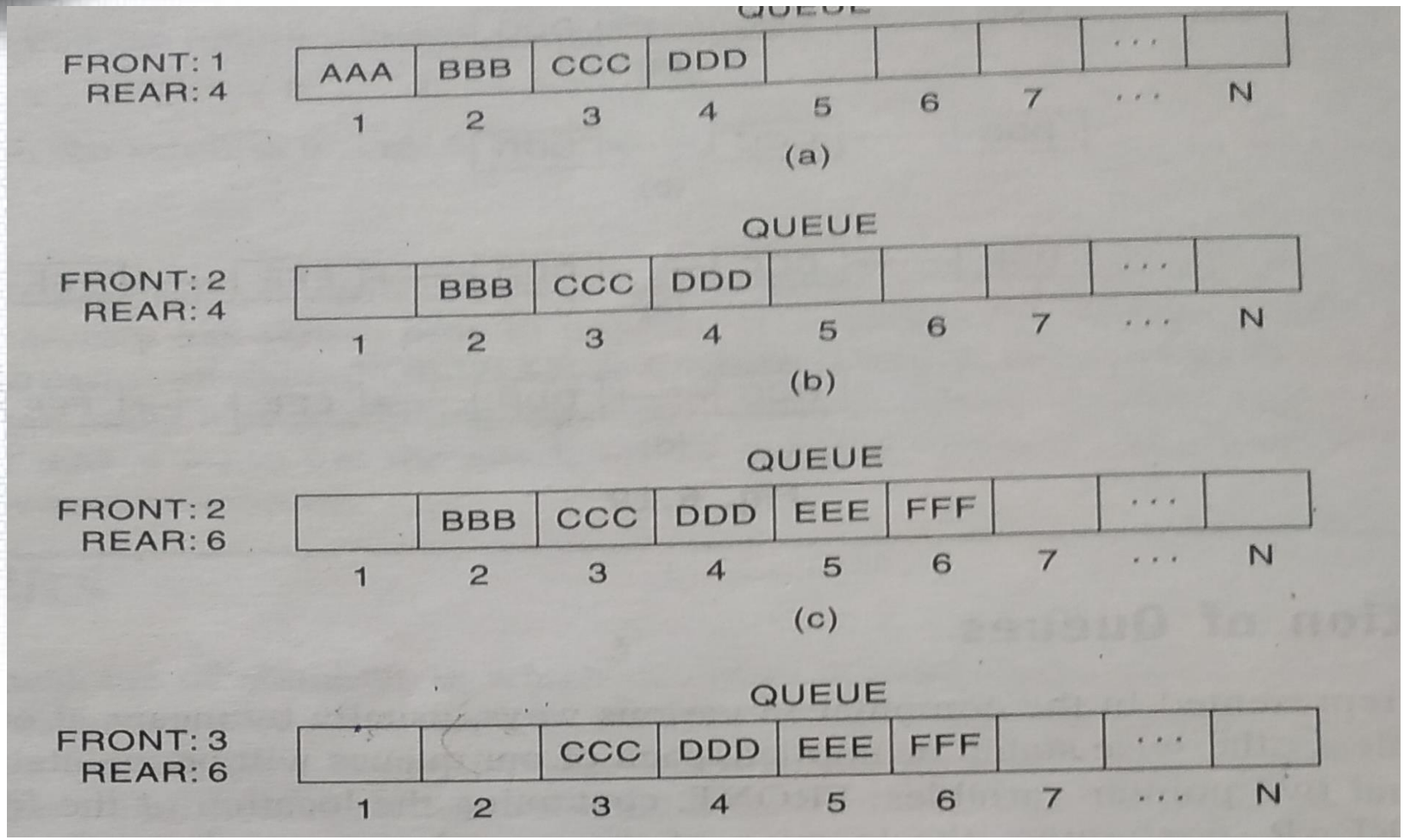
A + (B * C − (D / E ↑ F) * G) * H

A B C * D E F ↑ / G * − H * +

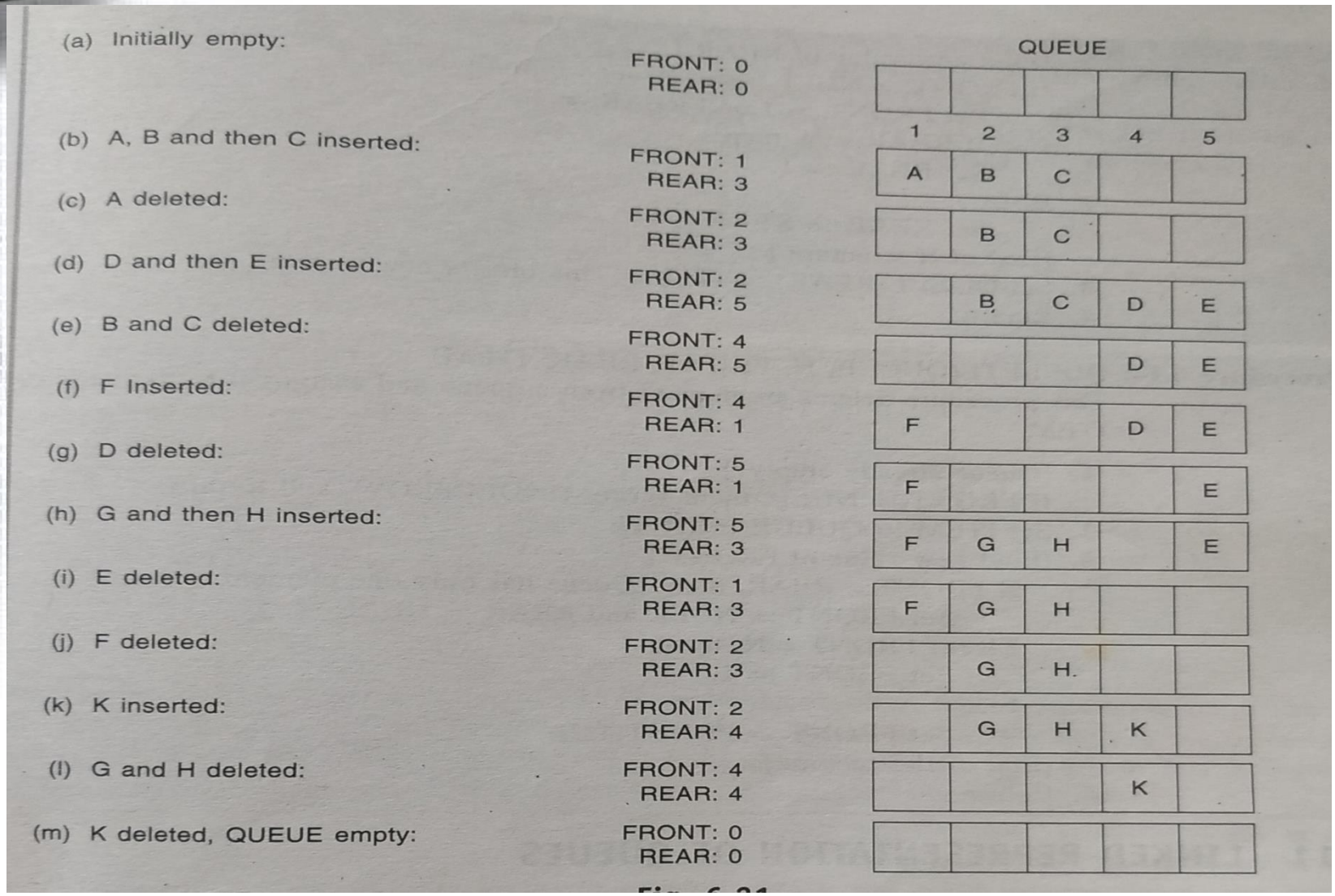
Queues

- Linear DS
- FIFO
- Can be implemented using Arrays or LL
- Front Pointer
- Rear Pointer

Queues : Array Representation



Queues : Circular Array Representation



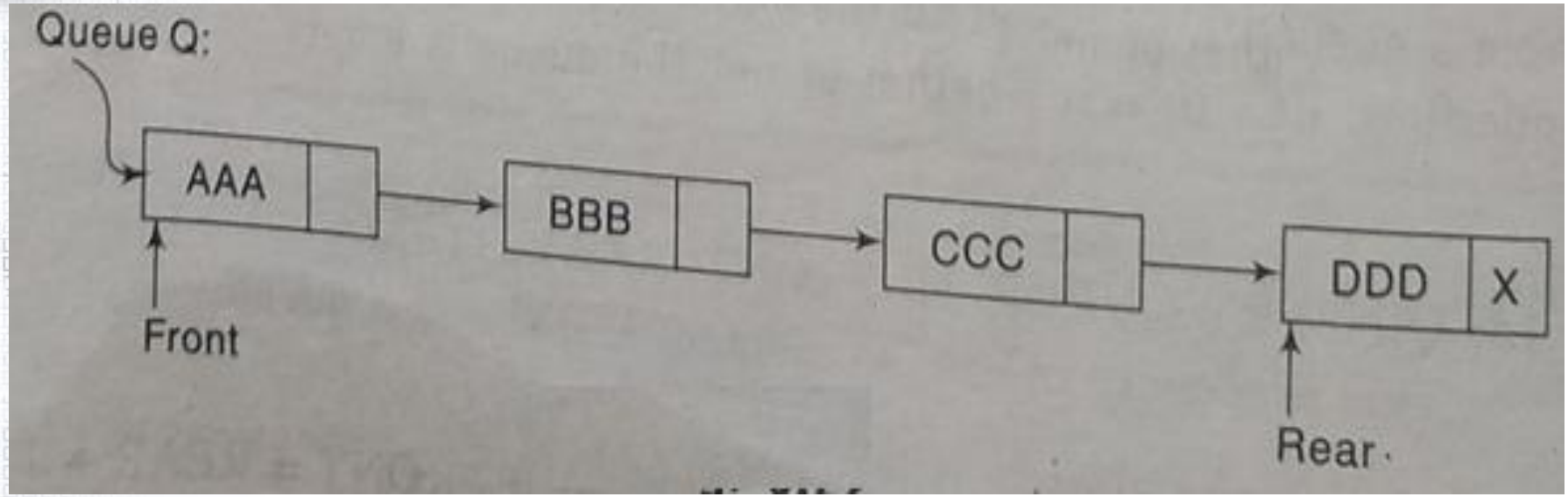
Queues : Insert

```
QINSERT(QUEUE, N, FRONT, REAR, ITEM)
This procedure inserts an element ITEM into a queue.

1. [Queue already filled?]
   If FRONT = 1 and REAR = N, or if FRONT = REAR + 1, then:
      Write: OVERFLOW, and Return.
2. [Find new value of REAR.]
   If FRONT := NULL, then: [Queue initially empty.]
      Set FRONT := 1 and REAR := 1.
   Else if REAR = N, then:
      Set REAR := 1.
   Else:
      Set REAR := REAR + 1.
   [End of If structure.]
3. Set QUEUE[REAR] := ITEM. [This inserts new element.]
4. Return.
```

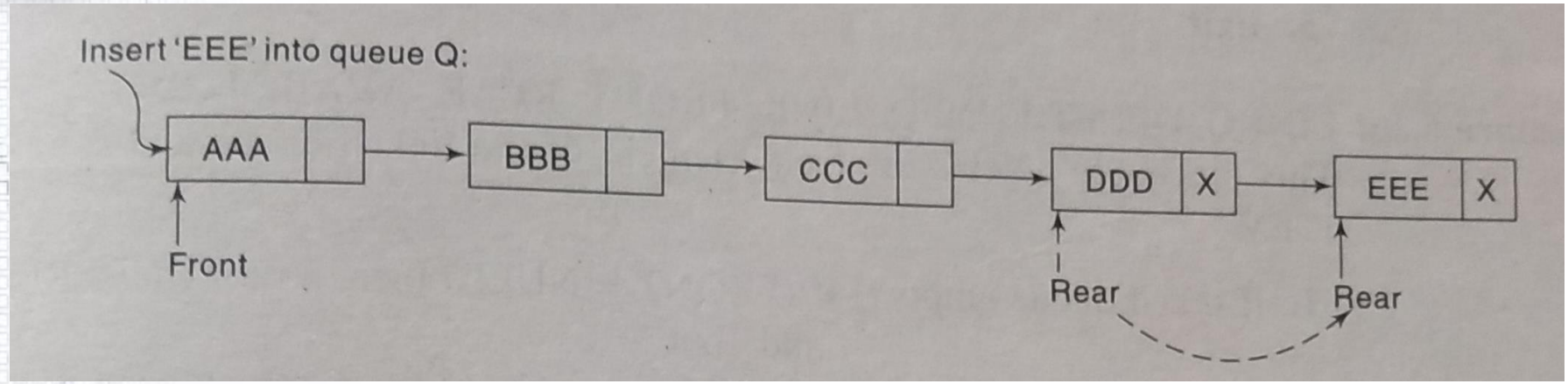
Queues : Delete

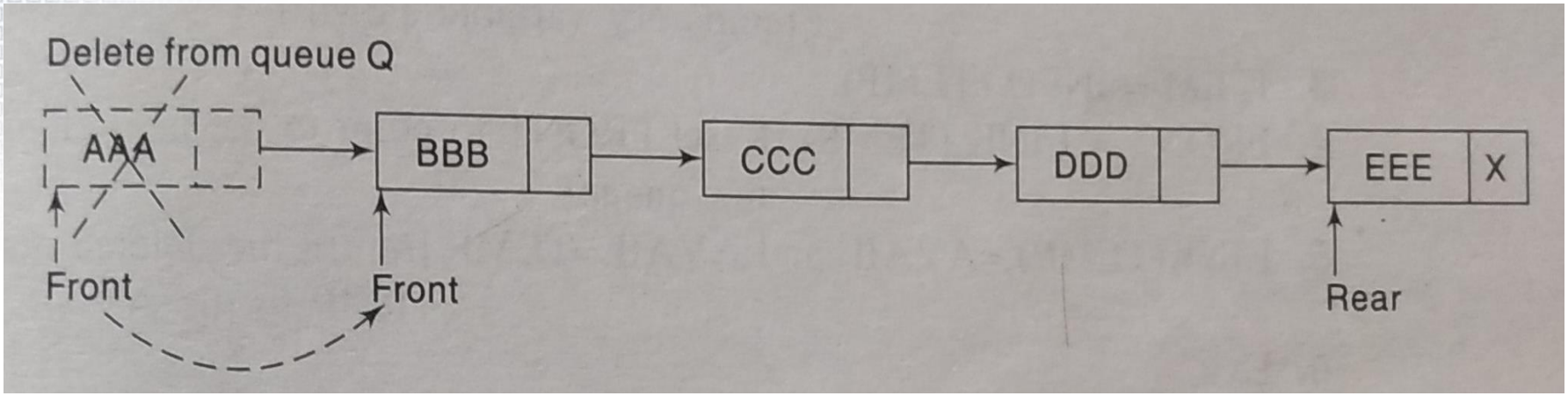
```
QDELETE(QUEUE, N, FRONT, REAR, ITEM)
This procedure deletes an element from a queue and assigns it to the variable
ITEM.

1. [Queue already empty?]
   If FRONT := NULL, then: Write: UNDERFLOW, and Return.
2. Set ITEM := QUEUE[FRONT].
3. [Find new value of FRONT.]
      If FRONT = REAR, then: [Queue has only one element to start.]
            Set FRONT := NULL and REAR := NULL.
      Else if FRONT = N, then:
            Set FRONT := 1.
      Else:
            Set FRONT := FRONT + 1.
   [End of If structure.]
4. Return.
```

Queues : Linked Representation



Queues : Linked Representation- Operations





Queues : Linked Representation- Insert

LINKQ_INSERT(INFO,LINK, FRONT, REAR,AVAIL,ITEM)
This procedure inserts an ITEM into a linked queue

1. [Available space?] 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 and LINK[NEW]=NULL
   [Copies ITEM into new node]
4. If (FRONT = NULL) then FRONT = REAR = NEW
   [If Q is empty then ITEM is the first element in the queue Q]
   else set LINK[REAR] := NEW and REAR = NEW
   [REAR points to the new node appended to the end of the list]
5. Exit.

Queues : Linked Representation- Delete

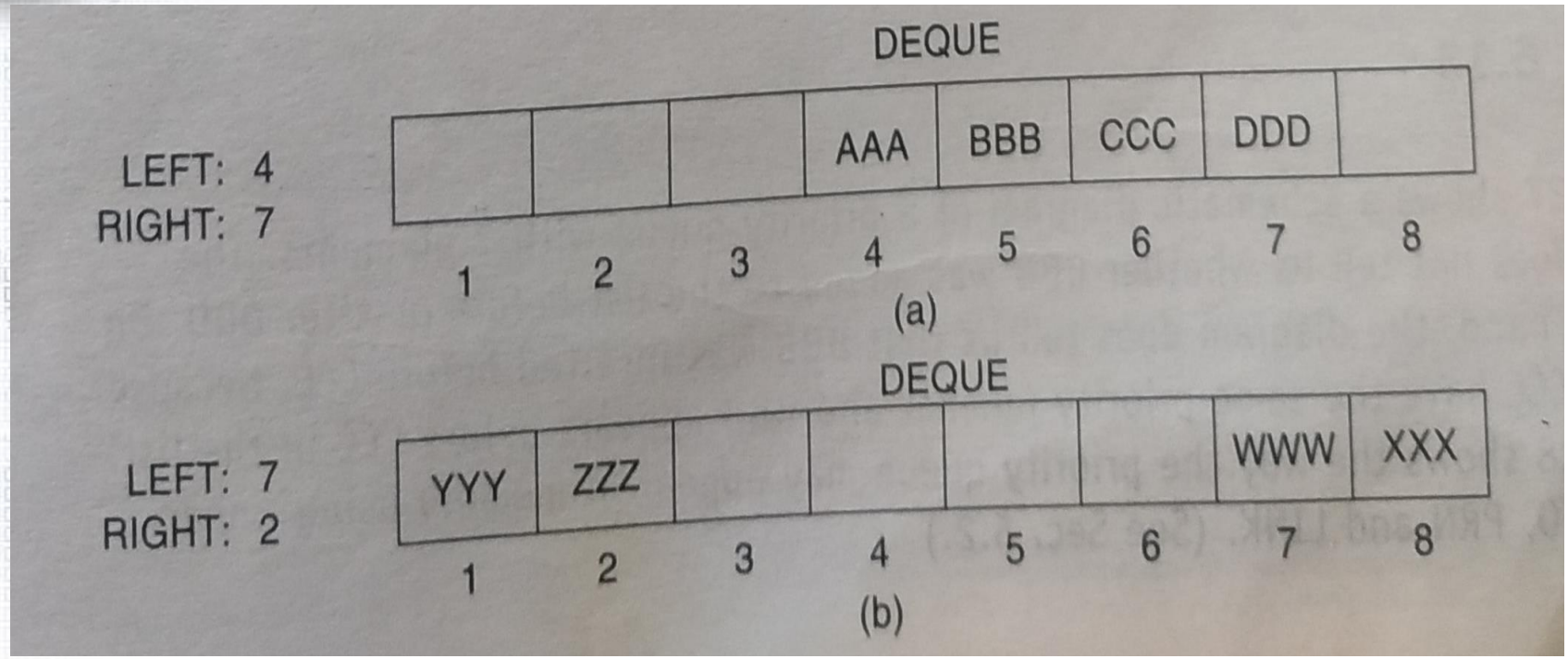
LINKQ_DELETE (INFO, LINK, FRONT, REAR, AVAIL, ITEM)
This procedure deletes the front element of the linked queue and stores it in ITEM

1. [Linked queue empty?] if (FRONT = NULL) then Write: UNDERFLOW and Exit
2. Set TEMP = FRONT [If linked queue is nonempty, remember FRONT in a temporary variable TEMP]
3. ITEM = INFO (TEMP)
4. FRONT = LINK (TEMP) [Reset FRONT to point to the next element in the queue]
5. LINK(TEMP) =AVAIL and AVAIL=TEMP [return the deleted node TEMP to the AVAIL list]
6. Exit.

Queues : Deques

- Double ended queue
- Elements can be added or removed at either end but not in the middle
- Representation using circular array DEQUE with pointer LEFT and RIGHT
- Two variants of deque, Input-restricted deque and Output-restricted deque

Queues : Deques



Queues : Priority Queue