Data Structure

Queue

Instructors:

Md Nazrul Islam Mondal
Department of Computer Science & Engineering
Rajshahi University of Engineering &
Technology Rajshahi-6204

Outline

- Queue
- Representation of Queue
- Linked Representation of Queue
- Deques
- Priority Queue

Queue

Queue

- First-in-First-out (**FIFO**)
- A linear list in which
  - **deletions** can take place only at one end of the list (The **front** of the list)
  - **Insertion** can take place only at the other end of the list (The **rear** of the list)

(b) Queue waiting for a bus

Representation of Queue

- **Empty Queue:** FRONT := NULL and REAR:= NULL

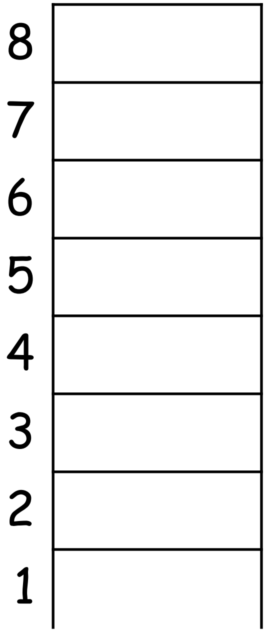


Fig. Queue

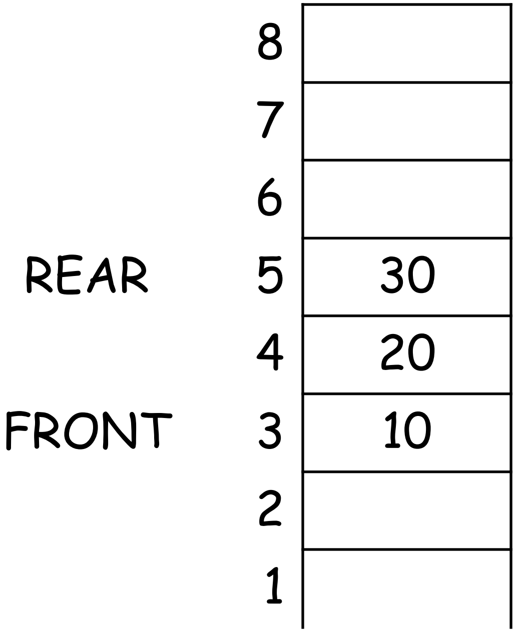
Representation of Queue

- **Delete an Element:**

  FRONT := FRONT + 1

- **Add an Element:**

  REAR := REAR + 1

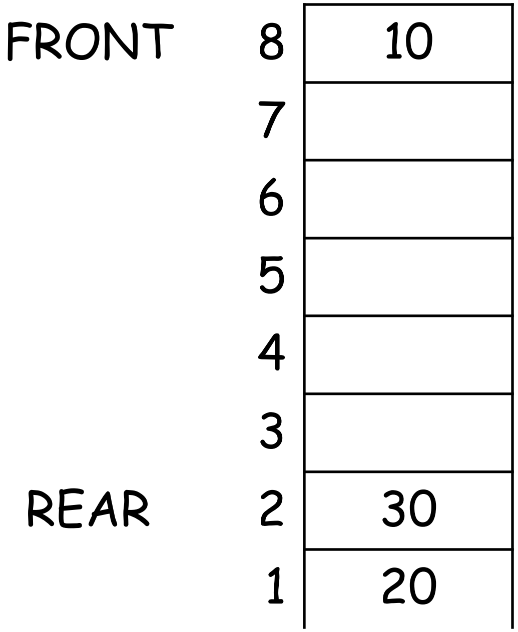


8
7
6
REAR 5 30
4 20
FRONT 3 10
2
1

Fig. Queue

Representation of Queue

- **Consider Circular Queue:**
  - If FRONT = N and delete an element , then
    - FRONT := 1



FRONT	8	10
	7	
	6	
	5	
	4	
	3	
REAR	2	30
	1	20

Fig. Queue

Representation of Queue

- **Consider Circular Queue:**
  - If REAR = N and add an element, then
    - REAR := 1

REAR	8	30
	7	20
FRONT	6	10
	5	
	4	
	3	
	2	
	1	

Fig. Queue

Representation of Queue

- **Queue Contain only one Element:**

  FRONT = REAR ≠ NULL

- **Now Delete an element, then**

  FRONT = REAR = NULL

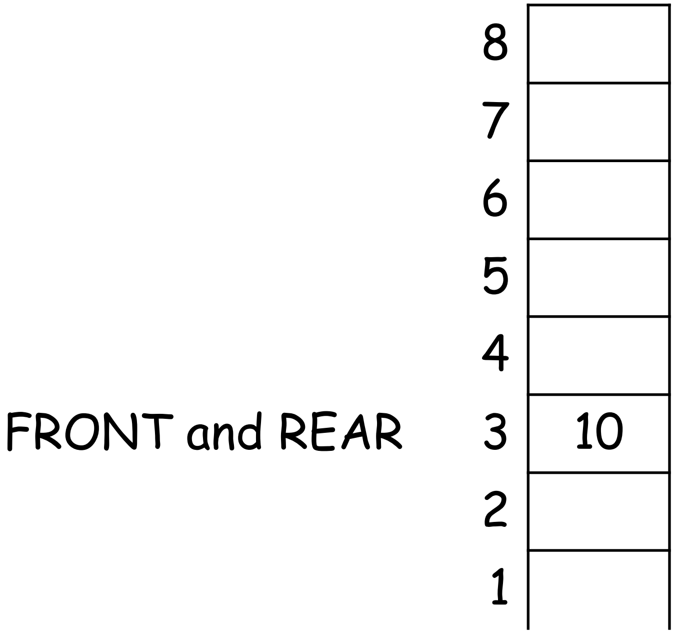


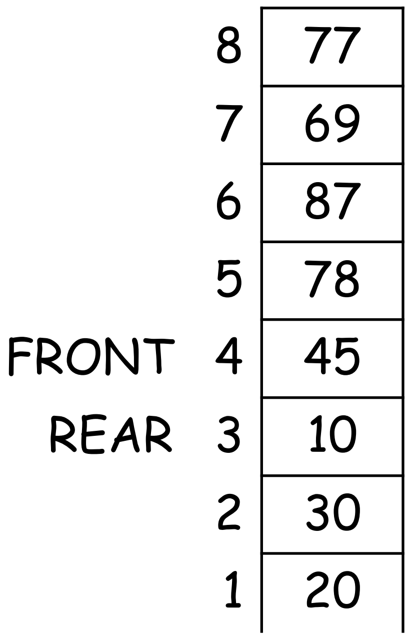
Fig. Queue

Representation of Queue

- **Queue Full:**

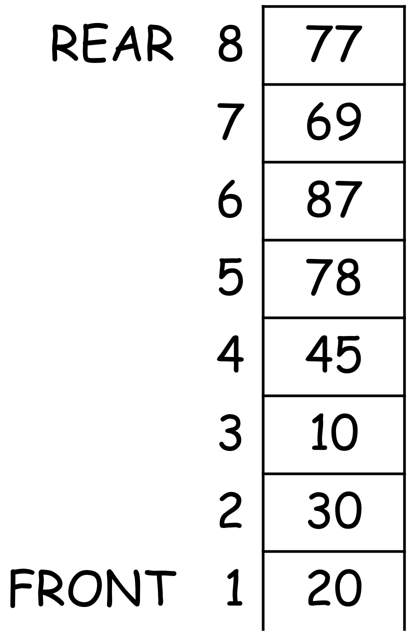
  FRONT =1 and REAR =N

  FRONT = REAR +1



	8	77
	7	69
	6	87
	5	78
FRONT	4	45
REAR	3	10
	2	30
	1	20

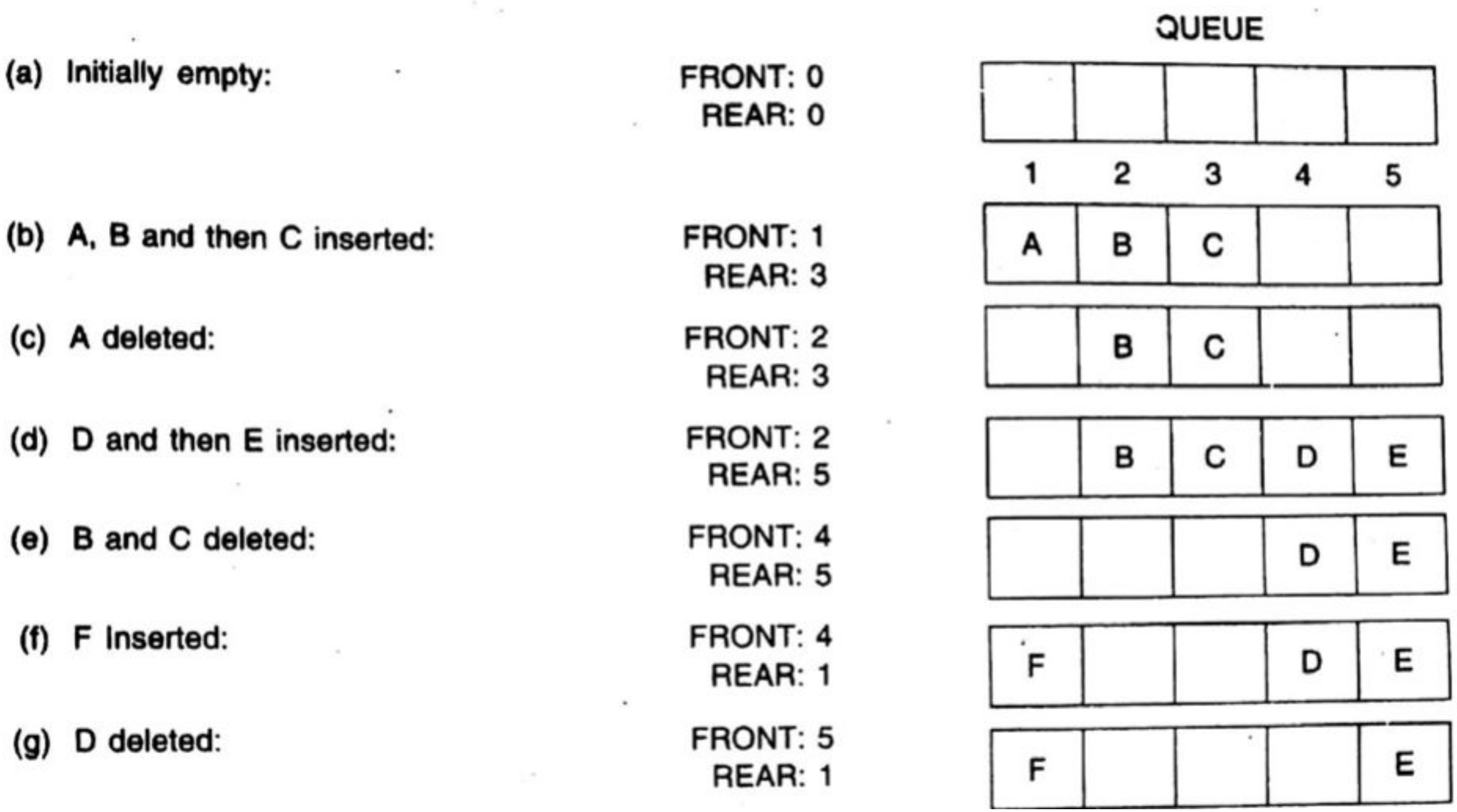
Fig. Queue



REAR	8	77
	7	69
	6	87
	5	78
	4	45
	3	10
	2	30
FRONT	1	20

Fig. Queue

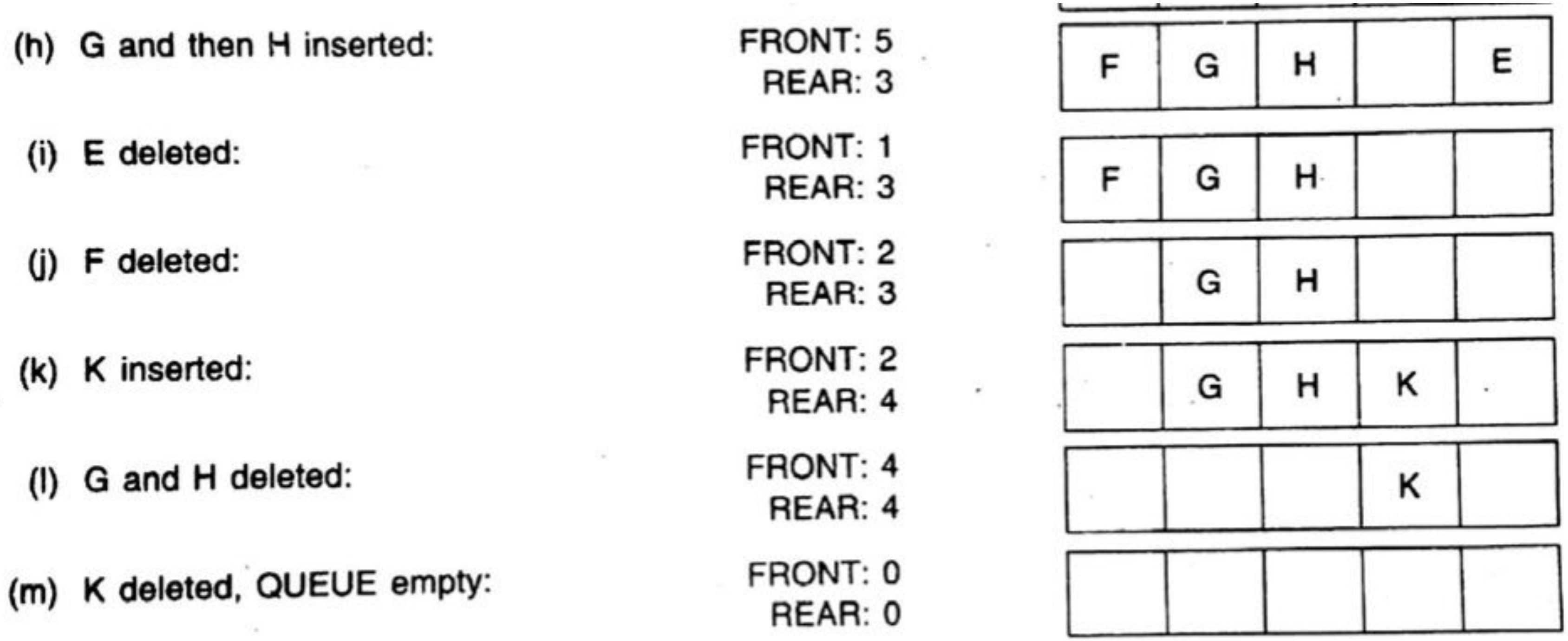
Representation of Queue



QUEUE

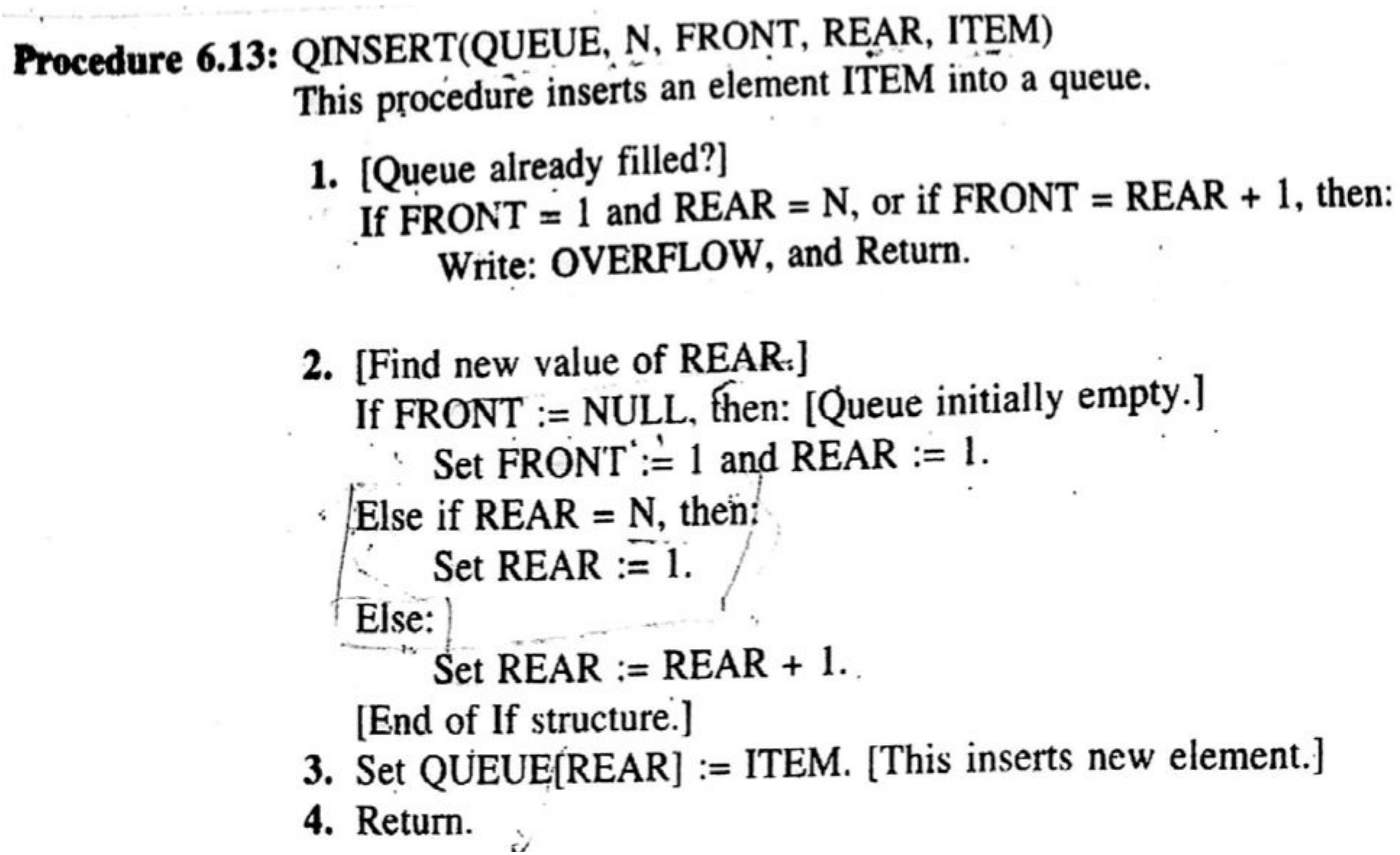
		1	2	3	4	5
(a) Initially empty:	FRONT: 0 REAR: 0					
(b) A, B and then C inserted:	FRONT: 1 REAR: 3	A	B	C		
(c) A deleted:	FRONT: 2 REAR: 3		B	C		
(d) D and then E inserted:	FRONT: 2 REAR: 5		B	C	D	E
(e) B and C deleted:	FRONT: 4 REAR: 5				D	E
(f) F Inserted:	FRONT: 4 REAR: 1	F			D	E
(g) D deleted:	FRONT: 5 REAR: 1	F				E

Representation of Queue



(h) G and then H inserted:	FRONT: 5 REAR: 3	F	G	H		E
(i) E deleted:	FRONT: 1 REAR: 3	F	G	H		
(j) F deleted:	FRONT: 2 REAR: 3		G	H		
(k) K inserted:	FRONT: 2 REAR: 4		G	H	K	
(l) G and H deleted:	FRONT: 4 REAR: 4				K	
(m) K deleted, QUEUE empty:	FRONT: 0 REAR: 0					

Representation of Queue

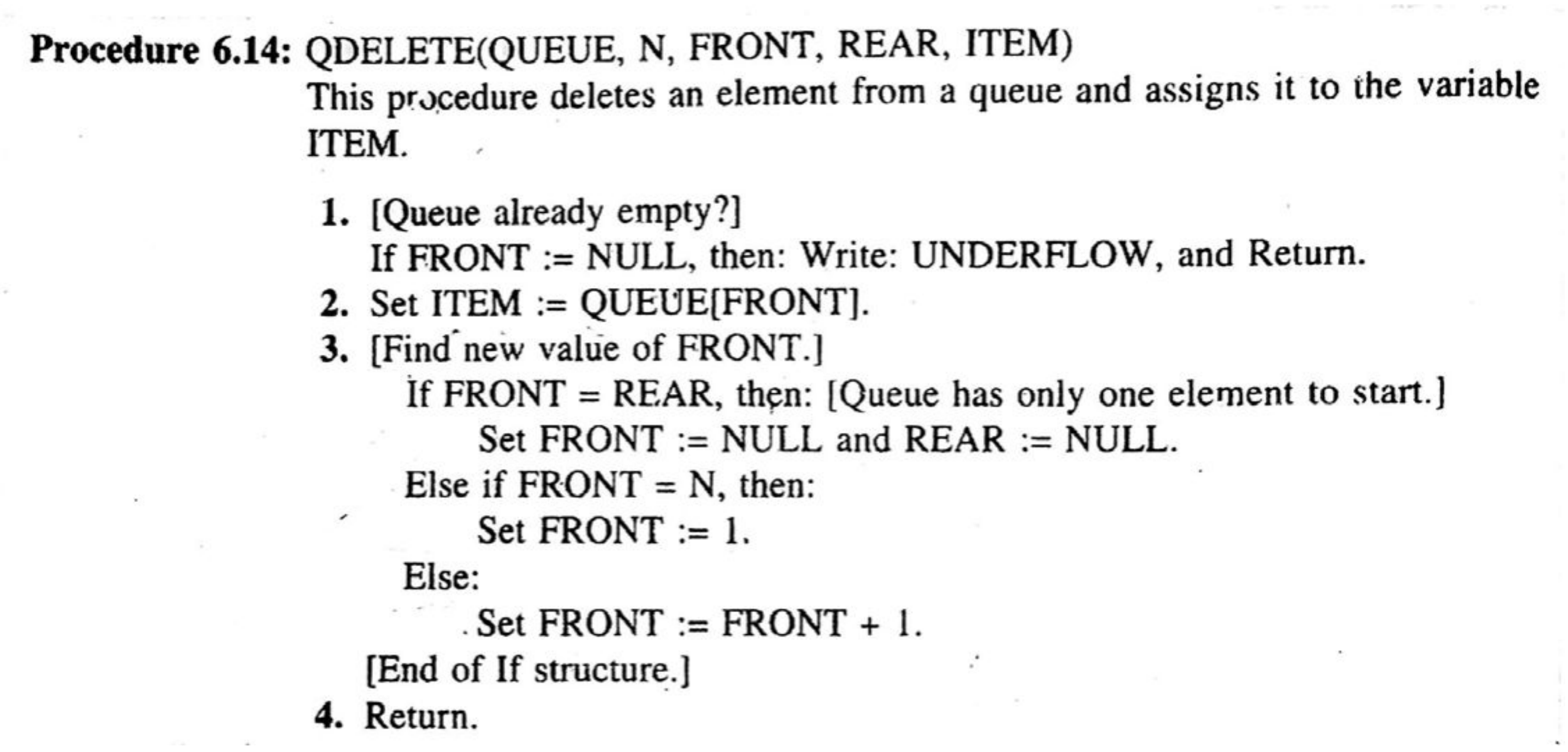


Procedure 6.13: QINSERT(QUEUE, N, FRONT, REAR, ITEM)
This procedure inserts an element ITEM into a queue.

```
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.
```

Representation of Queue



Procedure 6.14: 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.
```

Linked Representation of Queue

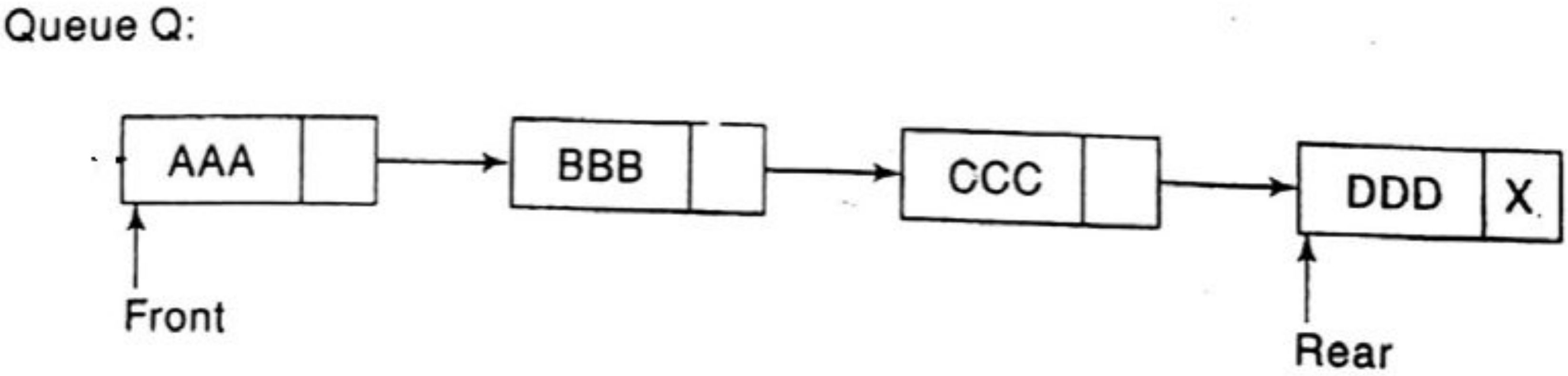


Fig. 6.22

Linked Representation of Queue

Insert 'EEE' into queue Q:

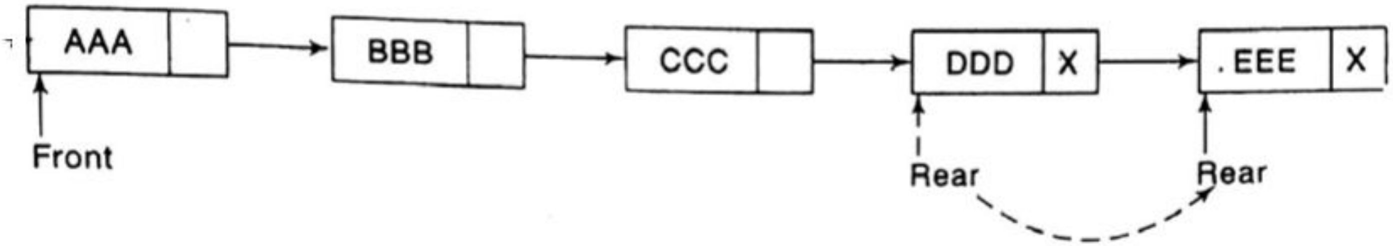


Fig. 6.23

Delete from queue Q

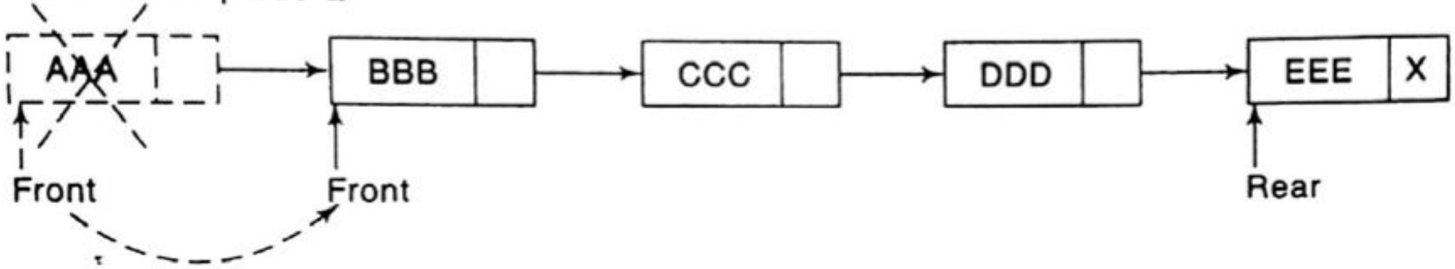
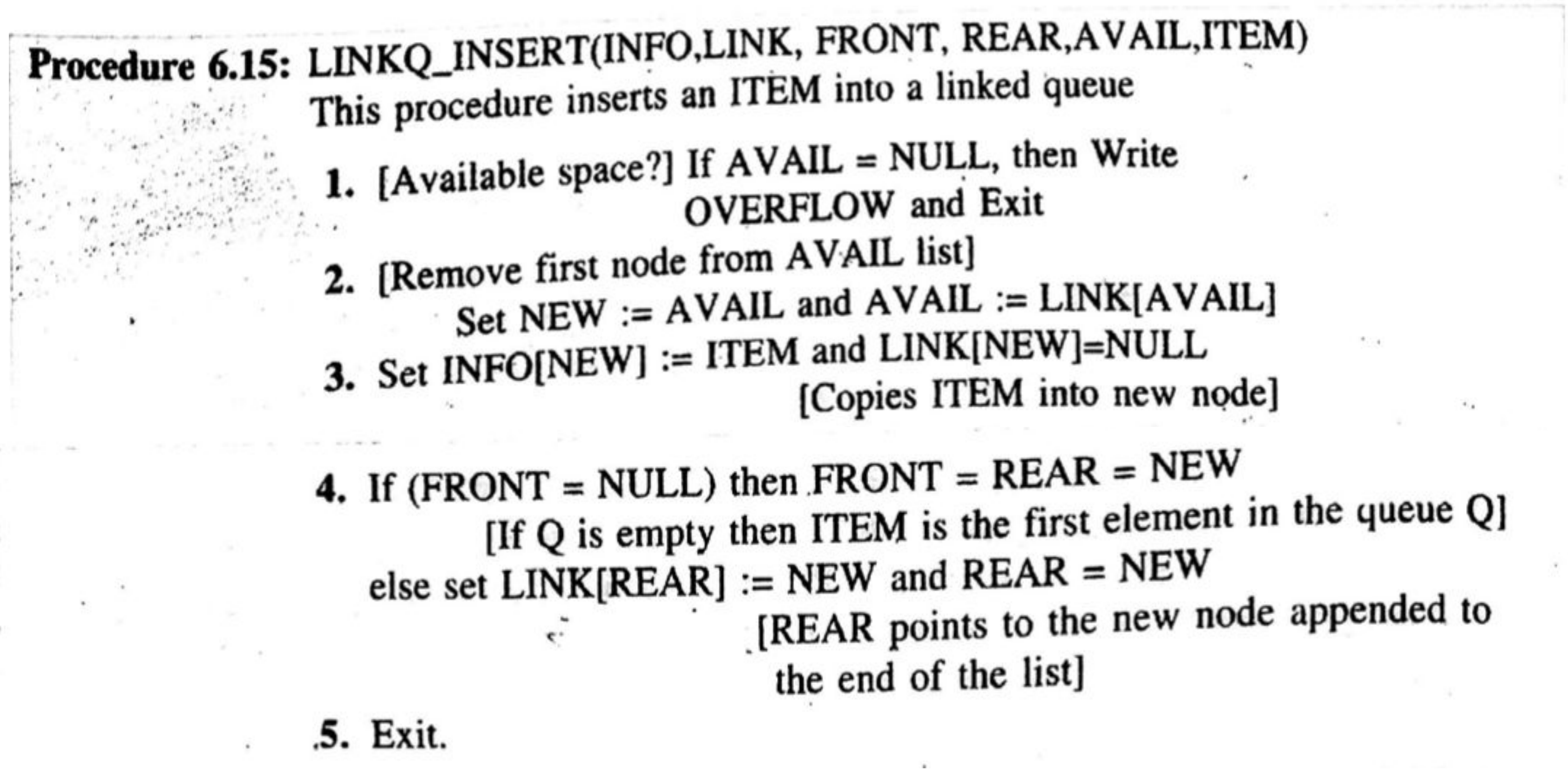


Fig. 6.24

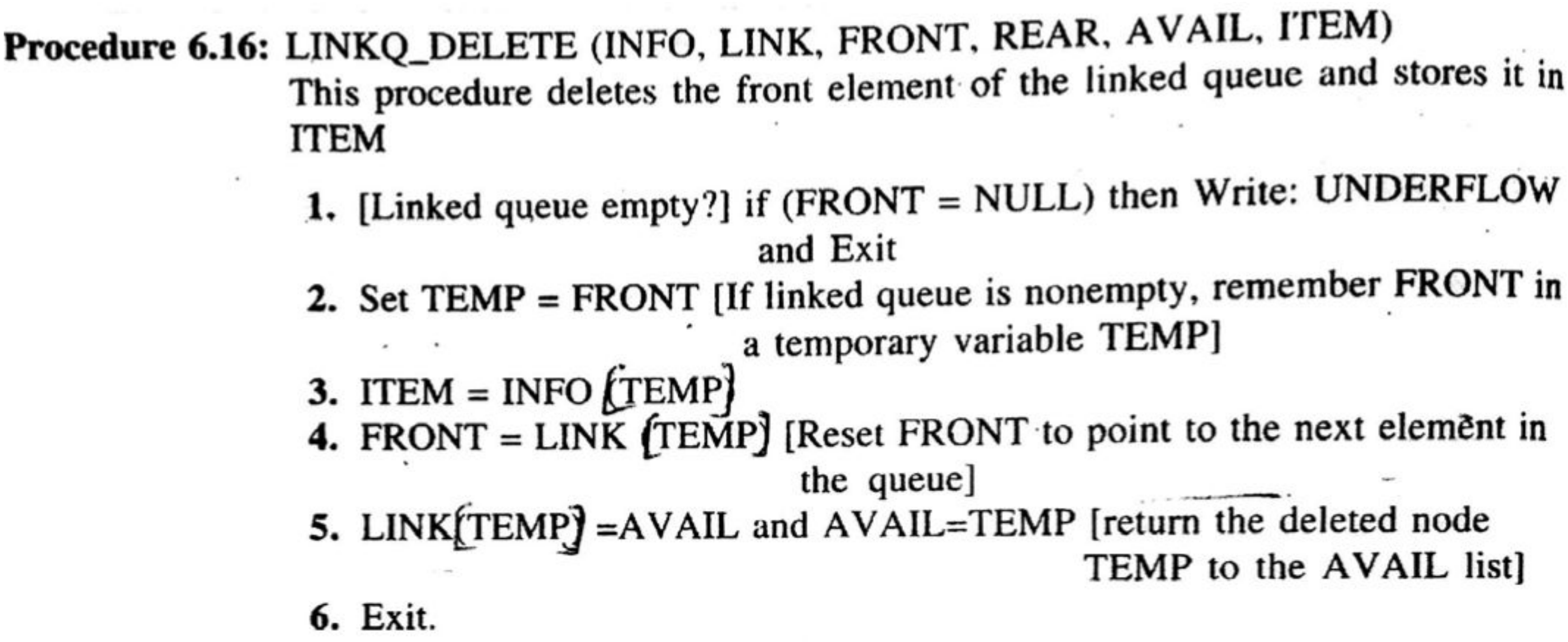
Linked Representation of Queue



```
Procedure 6.15: 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.
```

Linked Representation of Queue

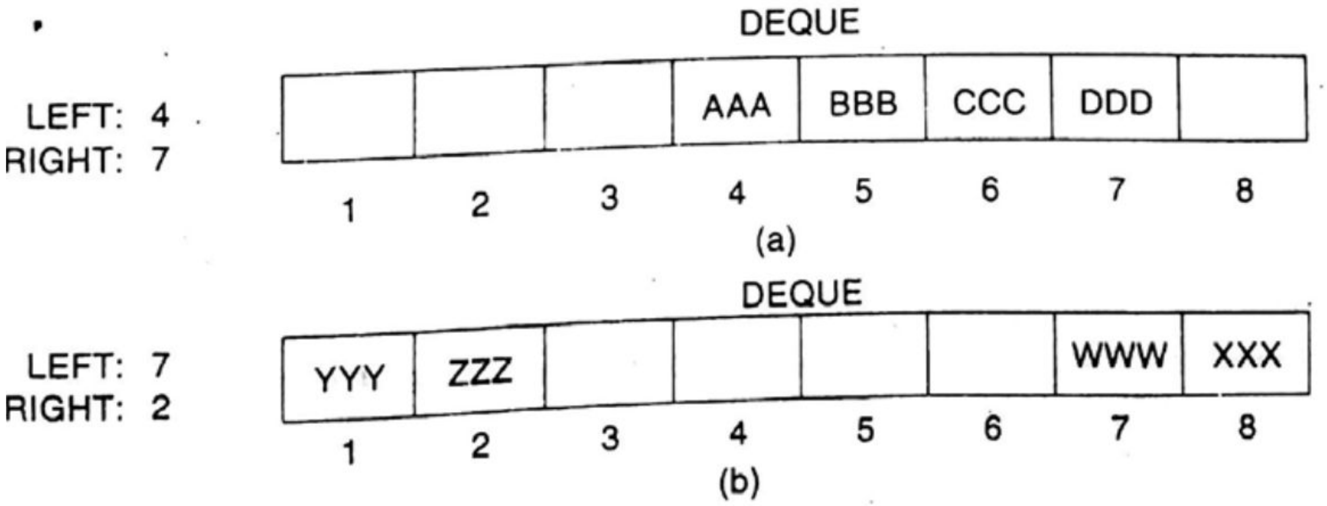


Procedure 6.16: LINKQ_DELETE (INFO, LINK, FRONT, REAR, AVAIL, ITEM)
This procedure deletes the front element of the linked queue and stores it in ITEM

```
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.
```

Deque (“deck” or “dequeue”)

- A linear list in which elements can be added or removed at either end but not in the middle.
- Double ended queue.
- Assume, our deque is maintained by a circular array DEQUE with pointers LEFT and RIGHT



Deque (“deck” or “dequeue”)

- Deque is **empty**: LEFT = NULL or RIGHT = NULL
- Deque is **full:** LEFT = 1 and RIGHT=N or LEFT = RIGHT+1

LEFT = 1	2	3	5					16
RIGHT = N	1	2	3	....	....	...	...	N

- Deleted two elements from left , LEFT = 3
- Added two elements on the right, RIGHT = 2

LEFT = 3	2	3	5					16
RIGHT = 2	1	2	3	....	....	...	...	N

Deque (“deck” or “dequeue”)

Algorithm: DEQINSR (DEQUE, N, LEFT, RIGHT, ITEM)

This procedure inserts an ITEM into a deque at right end.

```
1. [Deque already filled]
   IF LEFT = 1 and RIGHT = N, or If LEFT = RIGHT + 1, then:
         Write: OVERFLOW, and Return.
2. IF RIGHT=NULL, then: [Deque initially empty]
             Set LEFT:= 1 and RIGHT :=1.
   Else if RIGHT = N, then:
             Set RIGHT:=1.
   Else:
             Set RIGHT := RIGHT + 1.
   [End of If statement]
3. Set DEQUE[RIGHT]:= ITEM.
4. Return.
```

Deque (“deck” or “dequeue”)

Algorithm: DEQINSL (DEQUE, N, LEFT, RIGHT, ITEM)

This procedure inserts an ITEM into a deque at left end.

```
1. [Deque already filled]
 IF LEFT = 1 and RIGHT = N, or If LEFT = RIGHT + 1, then:
        Write: OVERFLOW, and Return.
2. IF LEFT=NULL, then: [Deque initially empty]
            Set LEFT:= 1 and RIGHT :=1.
 Else if LEFT = 1, then:
            Set LEFT:=N.
 Else:
            Set LEFT := LEFT - 1.
 [End of If statement]
3. Set DEQUE[LEFT]:= ITEM.
4. Return.
```

Deque (“deck” or “dequeue”)

Algorithm6.6: DEQDELR (DEQUE, N, LEFT, RIGHT, ITEM)

This procedure deletes an ITEM from a deque at right end and assigns it to the variable ITEM.

```
1. [Deque already Empty]
 IF RIGHT = NULL then: Write: UNDERFLOW, and Return.
2. Set ITEM:= DEQUE[RIGHT]
3. IF RIGHT=LEFT, then: [Deque contains only one element]
        Set LEFT:= NULL and RIGHT :=NULL.
 Else if RIGHT = 1, then:
        Set RIGHT:=N.
 Else:
        Set RIGHT := RIGHT - 1.
 [End of If statement]
4. Return.
```

Deque (“deck” or “dequeue”)

Algorithm6.6: DEQDELL (DEQUE, N, LEFT, RIGHT, ITEM)

This procedure deletes an ITEM from a deque at left end and assigns it to the variable ITEM.

```
1. [Dequeue already Empty]
   IF LEFT = NULL then: Write: UNDERFLOW, and Return.
2. Set ITEM:= DEQUE[LEFT]
3. IF RIGHT=LEFT, then: [Dequeue contains only one element]
           Set LEFT:= NULL and RIGHT :=NULL.
   Else if LEFT = N, then:
           Set LEFT:=1.
   Else:
           Set LEFT := LEFT + 1.
   [End of If statement]
4. Return.
```

Deque (“deck” or “dequeue”)

- Two type of deque –
  - Input-restricted deque
    - Allows insertions at only one end but allows deletions at both ends of the list
  - Output-restricted deque
    - Allows deletion at only one end but allows insertions at both ends of the list

PRIORITY QUEUE

Priority Queue

- A priority queue is a collection of elements such that each element has been **assigned a priority** and such that the order in which elements are **deleted and processed** comes from the following rules
  - An element of **higher priority is processed** before any element of lower priority.
  - Two elements with the **same priority** are processed according to the order in which they were added to the queue.

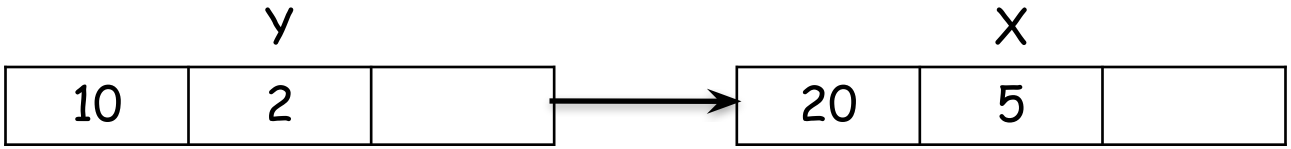
- A prototype of a priority queue is a timesharing system:
  - Programs of high priority are processed first
  - Programs of same priority form a standard queue

Priority Queue (One way List Representation)

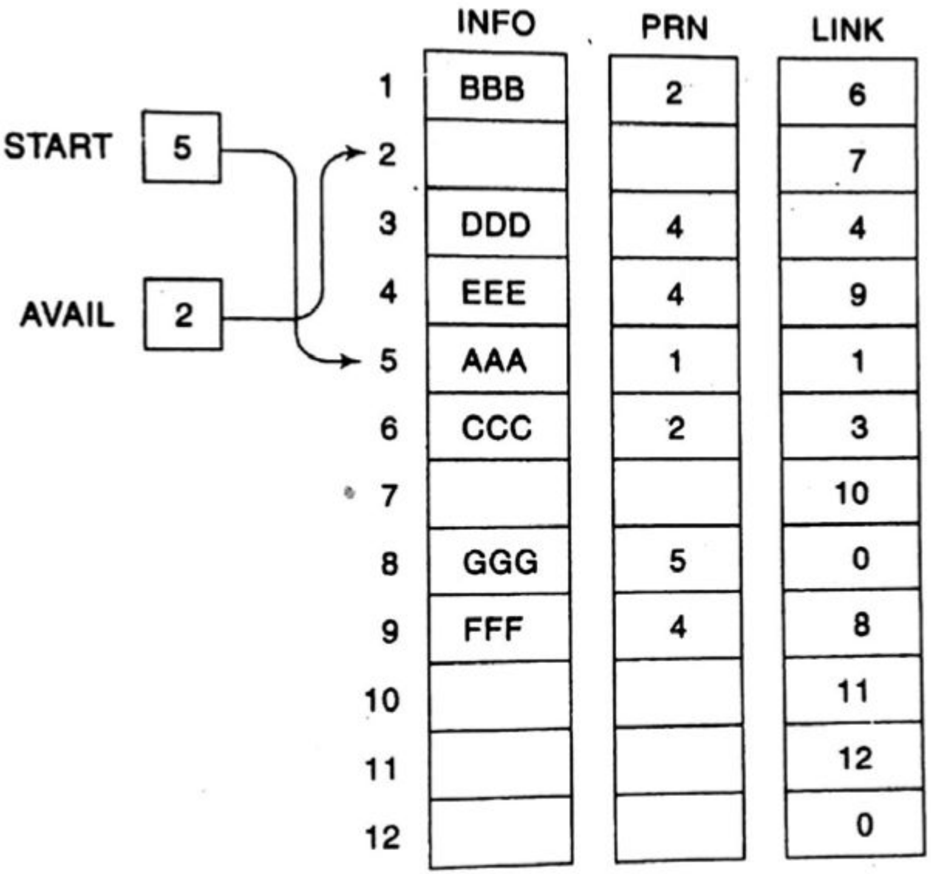
- Each node in the list will contain **three items of information**: an information field INFO, a priority number PRN and a link number LINK

INFO	PRN	LINK

- A node **X precedes a node Y** in the list (1) when X has higher priority than Y or (2) when both have the same priority but X was added to the list before Y.

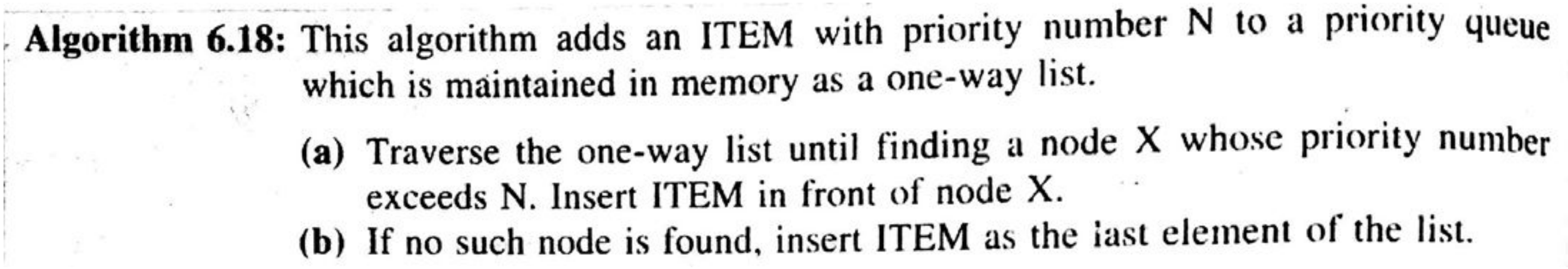


Priority Queue
(One way List Representation)



	INFO	PRN	LINK
1	BBB	2	6
2			7
3	DDD	4	4
4	EEE	4	9
5	AAA	1	1
6	CCC	2	3
7			10
8	GGG	5	0
9	FFF	4	8
10			11
11			12
12			0

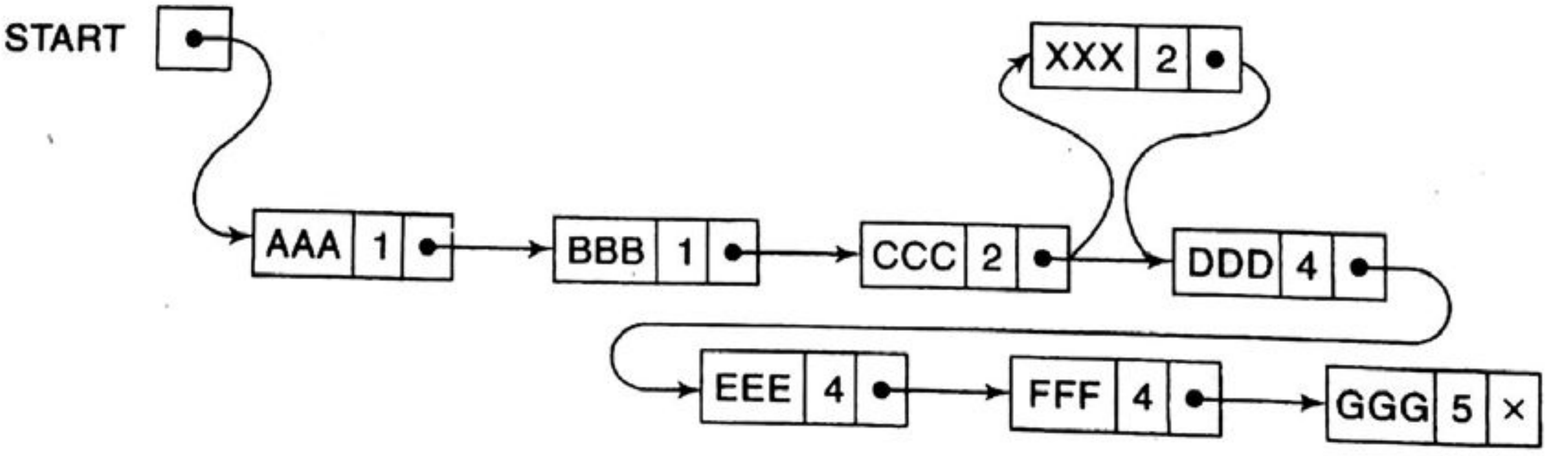
Priority Queue
(One way List Representation)



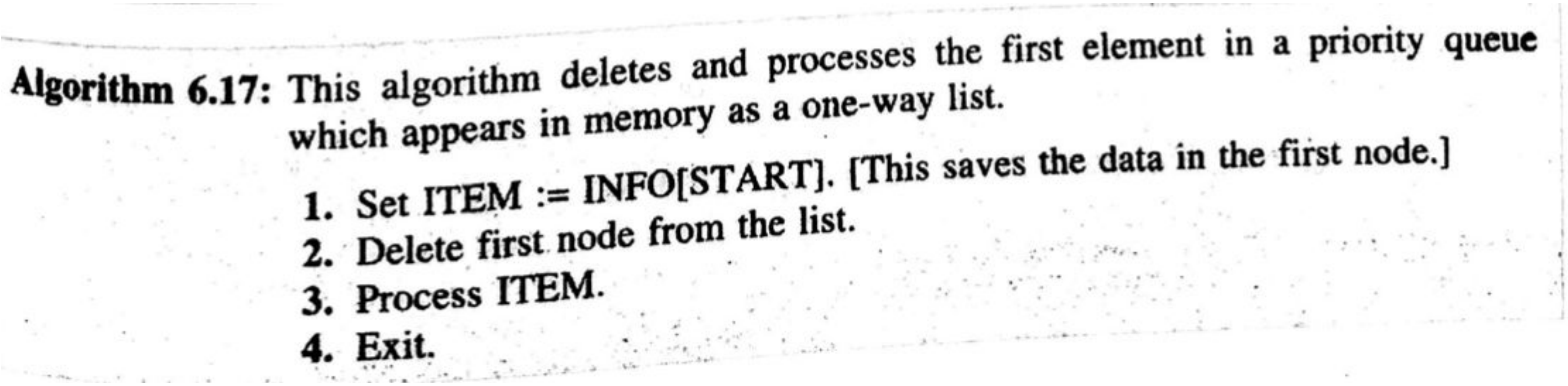
Algorithm 6.18: This algorithm adds an ITEM with priority number N to a priority queue which is maintained in memory as a one-way list.

(a) Traverse the one-way list until finding a node X whose priority number exceeds N. Insert ITEM in front of node X.

(b) If no such node is found, insert ITEM as the last element of the list.



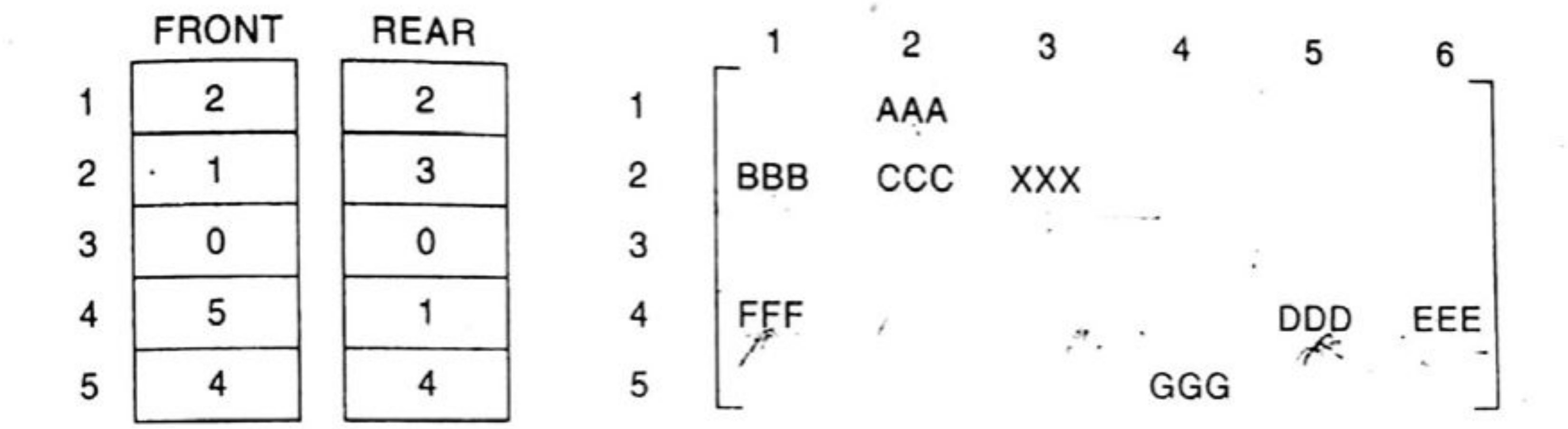
Priority Queue (One way List Representation)



Algorithm 6.17: This algorithm deletes and processes the first element in a priority queue which appears in memory as a one-way list.

```
1. Set ITEM := INFO[START]. [This saves the data in the first node.]
2. Delete first node from the list.
3. Process ITEM.
4. Exit.
```

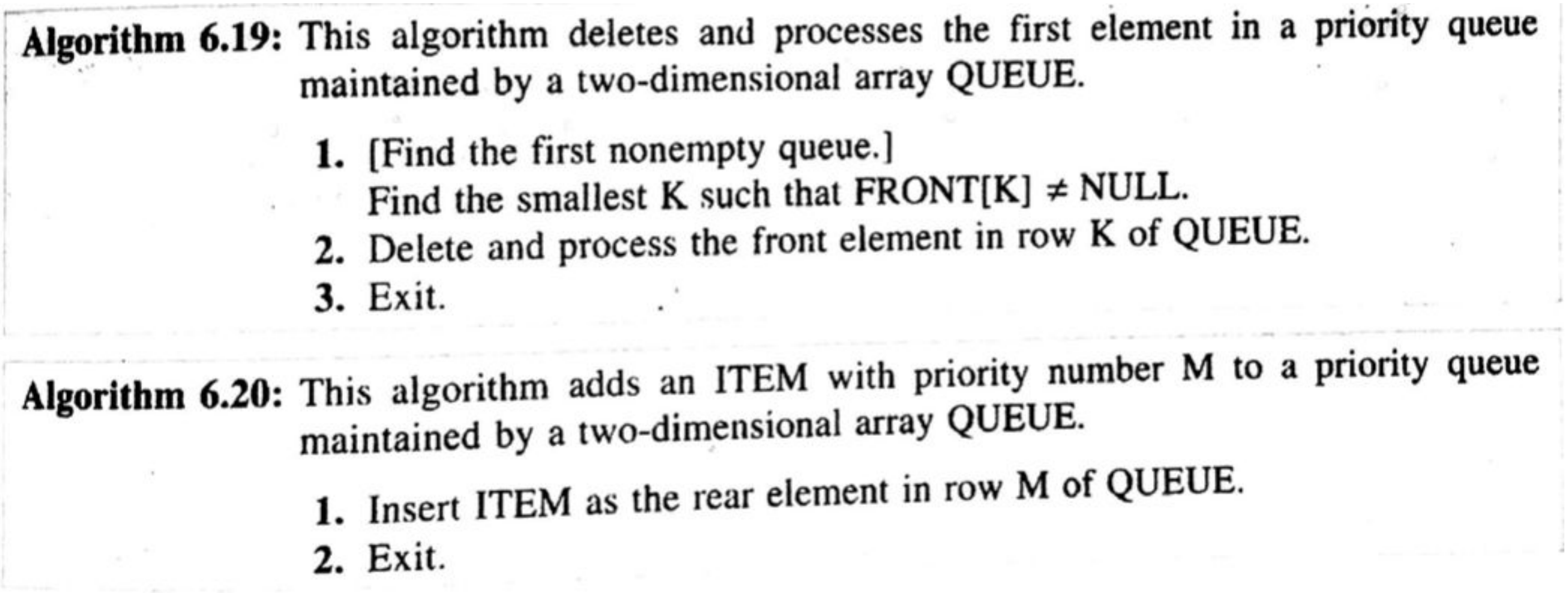
Priority Queue (Array Representation)



	FRONT	REAR
1	2	2
2	1	3
3	0	0
4	5	1
5	4	4

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & & \text{AAA} & & & & \\ 2 & \text{BBB} & \text{CCC} & \text{XXX} & & & \\ 3 & & & & & & \\ 4 & \text{FFF} & & & & \text{DDD} & \text{EEE} \\ 5 & & & & \text{GGG} & & \end{array}$$

Priority Queue (Array Representation)



Algorithm 6.19: This algorithm deletes and processes the first element in a priority queue maintained by a two-dimensional array QUEUE.

```
1. [Find the first nonempty queue.]
   Find the smallest K such that FRONT[K] ≠ NULL.
2. Delete and process the front element in row K of QUEUE.
3. Exit.
```

Algorithm 6.20: This algorithm adds an ITEM with priority number M to a priority queue maintained by a two-dimensional array QUEUE.

```
1. Insert ITEM as the rear element in row M of QUEUE.
2. Exit.
```

Any Query?

Data Structure