

Lecture - 9

On

Queues

QUEUES

- A Queue is a linear list of elements in which deletions can take place only at one end, called the *front*, and insertions can take place only at the other end, called the *rear*. The terms “front” and “rear” are used in describing a linear list only when it implemented as a queue.
- Queues are also called first-in first-out (FIFO) lists, since the first element in a queue will be the first element out of the queue. In other words, the order in which elements enter a queue is the order in which they leave. This contrasts with stacks, which are Last-in First-out (LIFO) lists.

Queues abound in everyday life.

For example:

- The automobiles waiting to pass through an intersection form a queue, in which the first car in line is the first car through;
- the people waiting in line at a bank form a queue, where the first person in line is the first person to be waited on; and so on.
- An important example of a queue in computer science occurs in a timesharing system, in which programs with the same priority form a queue while waiting to be executed.
- Queue for printing purposes

Why Queue?

There are many practical situations which involve Queue

- ❑ The queue of people waiting to be served by an ATM machine
- ❑ Collection of documents sent to a shared printer

Basic operations that involved in Queue:

- Create queue, *Create* (Q)
- Identify either queue is empty
- Add new item in queue
- Delete item from queue
- Call first item in queue

Example :

Figure 6-15 (a) is a schematic diagram of a queue with 4 elements, where AAA is the front element and DDD is the rear element. Observe that the front and rear elements of the queue are also, respectively, the first and last elements of the list. Suppose an element is deleted from the queue. Then it must be AAA. This yields the queue in figure 6-15(b), where BBB is now the front element. Next, suppose EEE is added to the queue and then FFF is now the rear element. Now suppose another element is deleted from the queue; then it must be BBB, to yield the queue in fig.6-15(d). And so on. Observe that in such a data structure, EEE will be deleted before FFF because it has been placed in the queue before FFF. However, EEE will have to wait until CCC and DDD are deleted.

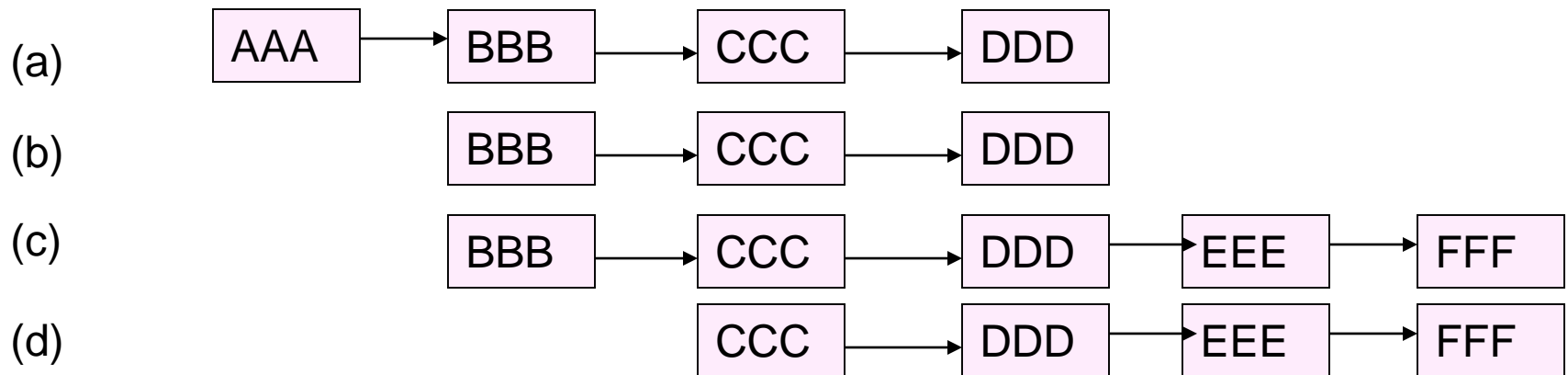


Fig. 6-15

REPRESENTATION OF QUEUE

- Queue may be represented in the computer in various ways, usually by means of one-way lists or linear arrays.
- Each of our queues will be maintained by a linear array QUEUE and two pointer variables:
- FRONT, containing the location of the front element of the queue;
- and REAR, containing the location of the rear element of the queue.
- The condition $\text{FRONT} = \text{NULL}$ will indicate that the queue is empty.

REPRESENTATION OF QUEUE

- when an element is deleted from the queue, the value of FRONT is increased by 1; this can be implemented by the assignment
- $\text{FRONT} := \text{FRONT} + 1$
- Similarly, whenever an element is added to the queue, the value of REAR is increased by 1; this can be implemented by the assignment
- $\text{REAR} := \text{REAR} + 1$
- This means that after N insertions, the rear element of the queue will occupy QUEUE[N] or, in other words, eventually the queue will occupy the last part of the array. This occurs even though the queue itself may not contain many elements.

REPRESENTATION OF QUEUE

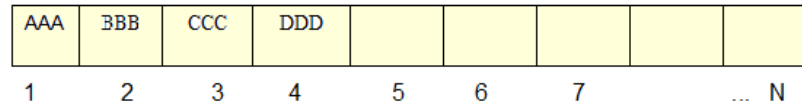
- Suppose we want to insert an element ITEM into a queue at the time the queue does occupy the past part of the array,
- i.e. when $REAR = N$. One way to do this is to simply move the entire queue to the beginning of the array, changing FRONT and REAR accordingly, and then inserting ITEM as above.
- Array QUEUE is circular, that is, that $QUEUE[1]$ comes after $QUEUE[N]$ in the array. With this assumption, we insert ITEM into the queue by assigning ITEM to $QUEUE[1]$. Specifically, instead of increasing REAR to $N+1$, we reset $REAR=1$ and then assign
 $QUEUE[REAR] := ITEM$
- Similarly, if $FRONT = N$ and an element of QUEUE is deleted, we reset $FRONT = 1$ instead of increasing FRONT to $N + 1$.
- Suppose that our queue contains only one element, i.e., suppose that
- $FRONT = REAR = NULL$
- And suppose that the element is deleted. Then we assign
- $FRONT := NULL$ and $REAR := NULL$
To indicate that the queue is empty.

REPRESENTATION OF QUEUE

QUEUE

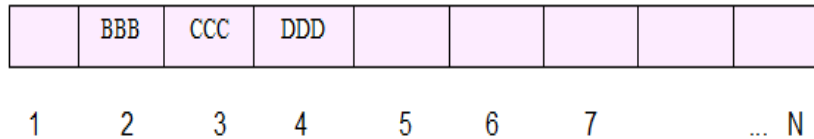
FRONT : 1

REAR : 4



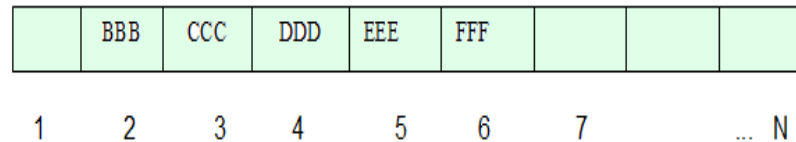
FRONT : 2

REAR : 4



FRONT : 2

REAR : 6



FRONT : 3

REAR : 6

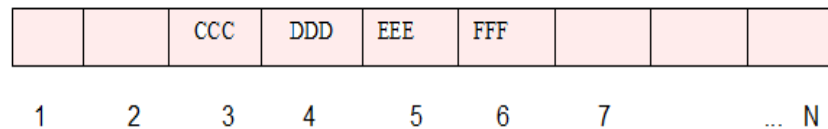


Fig : 6.16 Array representation of a queue

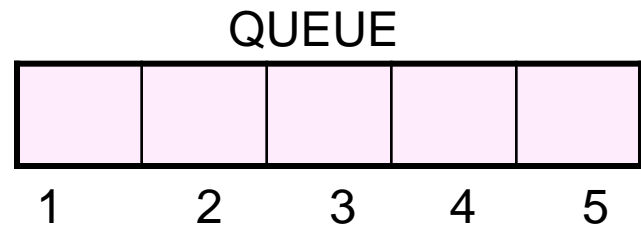
Example

Figure 6-17 shows how a queue may be maintained by a circular array QUEUE with $N = 5$ memory locations. Observe that the queue always occupies consecutive locations except when it occupies locations at the beginning and at the end of the array. If the queue is viewed as a circular array, this means that it still occupies consecutive locations. Also, as indicated by fig. 6-17(m), the queue will be empty only when $\text{FRONT} = \text{REAR}$ and an element is deleted. For this reason, NULL is assigned to FRONT and REAR in fig. 6-17(m).

(a) Initially empty :

FRONT : 0

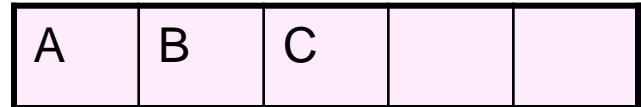
REAR : 0



(b) A, B and then C inserted:

FRONT : 1

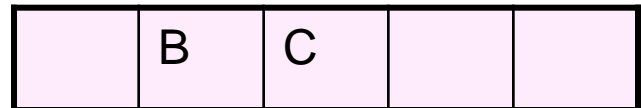
REAR : 3



(c) A deleted :

FRONT : 2

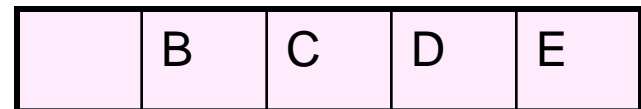
REAR : 3



(d) D and then E inserted :

FRONT : 2

REAR : 5

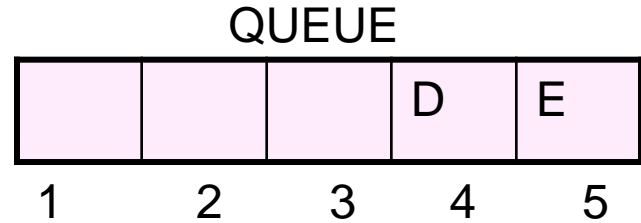


Example

(e) B and C deleted :

FRONT : 4

REAR : 5



(f) F inserted :

FRONT : 4

REAR : 1



(g) D deleted :

FRONT : 5

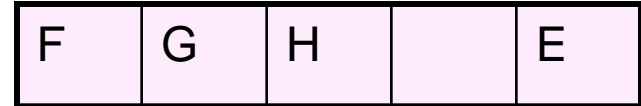
REAR : 1



(h) G and then H inserted :

FRONT : 5

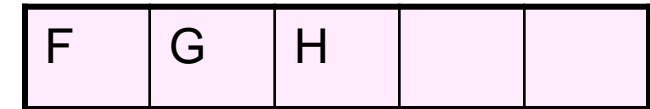
REAR : 3



(i) E deleted :

FRONT : 1

REAR : 3



(j) F deleted :

FRONT : 2

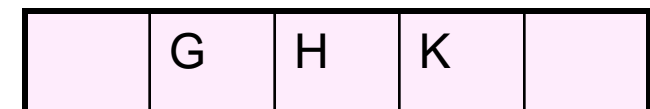
REAR : 3



(k) K inserted :

FRONT : 2

REAR : 4

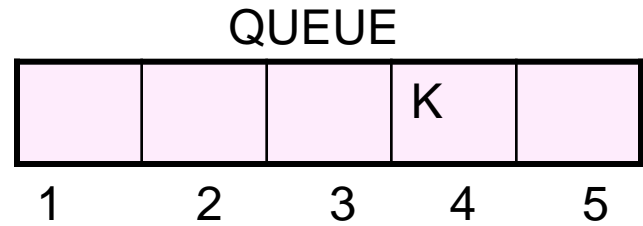


Example

(l) G and H deleted :

FRONT : 4

REAR : 5



(m) K deleted, QUEUE is empty :

FRONT : 4

REAR : 1



Fig :6.17

QINSERT

QINSERT(Queue,N,FRONT,REAR,ITEM)

This procedure inserts an element ITEM into a queue.

[Queue already fill]

1. 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.

QDELETE

ITEM.QDELETE(Queue,N,FRONT,REAR,ITEM)

This procedure deletes an element from the queue and assigns it to the variable

- [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

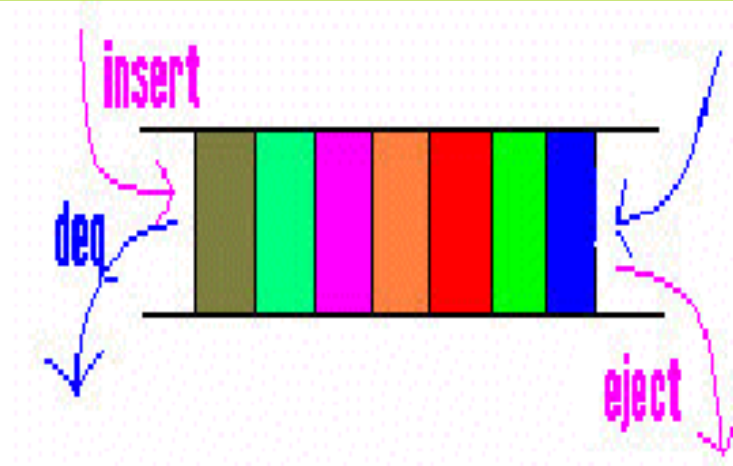
[End of if structure]

4. Return

Deque

- The mathematical model of a **Deque** (usually pronounced like "deck") is an irregular acronym (Operation) of **double-ended queue**.
- Double-ended queues are a kind of sequence containers.
- Elements can be efficiently added and removed from any of its ends (either the beginning or the end of the sequence).

•The model allows data to be entered and withdrawn from the front and rear of the data structure.



Dequeues

- Insertions *and* deletions can occur at *either* end but not in the middle
- Implementation is similar to that for queues
- Deques are not heavily used
- You should know what a deque is, but we won't explore them much further

Double-Ended-QUE

- There are two variations of deque

- Input-restricted deque

An input restricted deque is a deque which allows insertion at only one end of the list but allows deletion at both ends of the list.

- Output-restricted deque

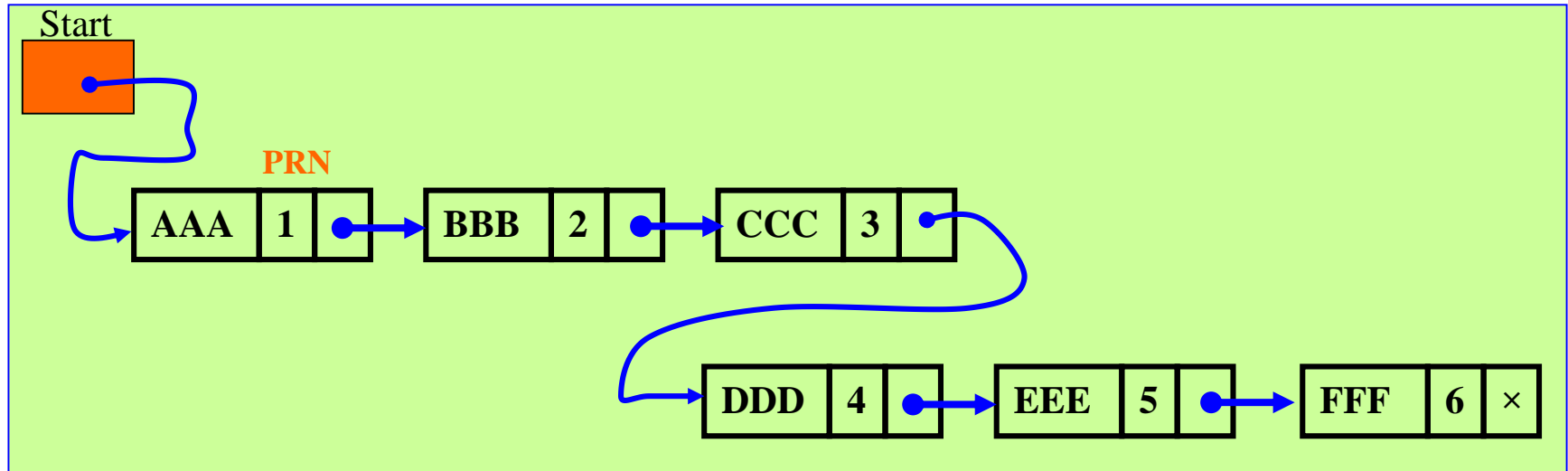
An output restricted deque is a deque which allows deletion at only one end of the list but allows insertion at both ends of the list

Priority Queue

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

One way list representation

- Each node contains three items of information
- A node X proceeds with a node Y in the list when
 - X has higher priority than Y **or**
 - Both has same priority but X was added in the list before Y.



- Property:
 - First node will processed first