Queue

Concept of queues using the analogies given below:

- People moving on an escalator. The people who got on the escalator first will be the first one to step out of it.
- People waiting for a bus. The first person standing in the line will be the first one to get into the bus.
- People standing outside the ticketing window of a cinema hall. The first person in the line will get the ticket first and thus will be the first one to move out of it.
- Luggage kept on conveyor belts. The bag which was placed first will be the first to come out at the other end.
- Cars lined at a toll bridge. The first car to reach the bridge will be the first to leave.

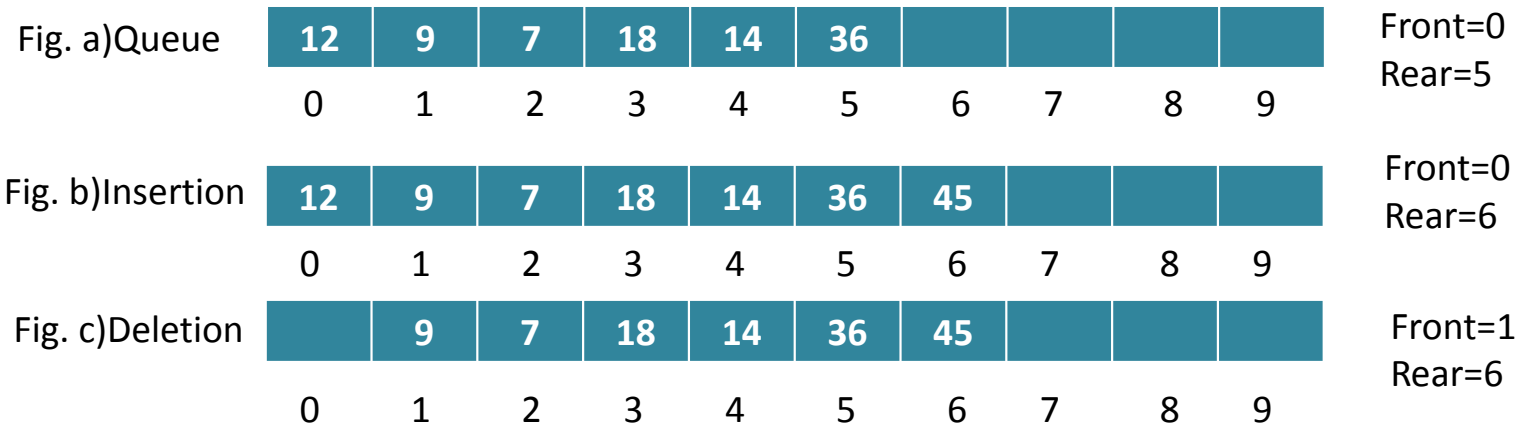
Contd.

In all previous examples, we see that the element at the first position is served first. Same is the case with queue data structure. A queue is a FIFO (First-In, First-Out) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end called the REAR and removed from the other end called the FRONT.

Contd.

ARRAY REPRESENTATION OF QUEUES

Queues can be easily represented using linear arrays. As stated earlier, every queue has front and rear variables that point to the position from where deletions and insertions can be done, respectively.



Algorithm to insert an element in a queue

1. IF REAR = MAX-1
   Write OVERFLOW Goto Step 4
   [END OF IF]
2. IF FRONT = -1 and REAR = -1
   SET FRONT = REAR = 0
   ELSE
   SET REAR = REAR + 1
   [END OF IF]
3. SET QUEUE[REAR] = NUM
4. EXIT

Algorithm to delete an element from a queue

1. IF FRONT = -1 OR FRONT > REAR

   Write UNDERFLOW

   ELSE

   SET VAL = QUEUE[FRONT]

   SET FRONT = FRONT + 1

   [END OF IF]

2. EXIT

LINKED REPRESENTATION OF QUEUES

Advantage: dynamically memory allocation

In a linked queue, every element has two parts, one that stores the data and another that stores the address of the next element. The START pointer of the linked list is used as FRONT. Here, we will also use another pointer called REAR, which will store the address of the last element in the queue. All insertions will be done at the rear end and all the deletions will be done at the front end. If FRONT = REAR = NULL, then it indicates that the queue is empty.

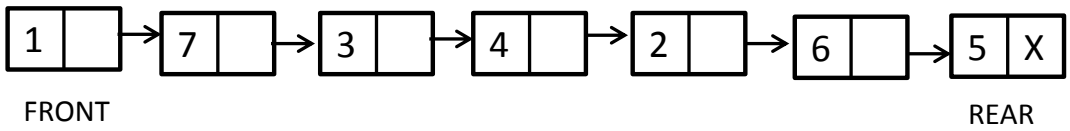


Fig. Linked queue

Operations on Linked Queues

A queue has two basic operations:

- **insert**
- **delete**

The insert operation adds an element to the end of the queue, and the delete operation removes an element from the front or the start of the queue. Apart from this, there is another operation peek which returns the value of the first element of the queue.

Insert Operation

The insert operation is used to insert an element into a queue. The new element is added as the last element of the queue.

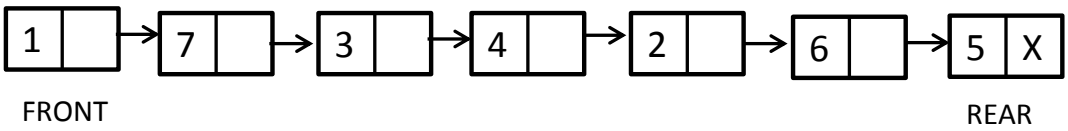


Fig. Linked queue

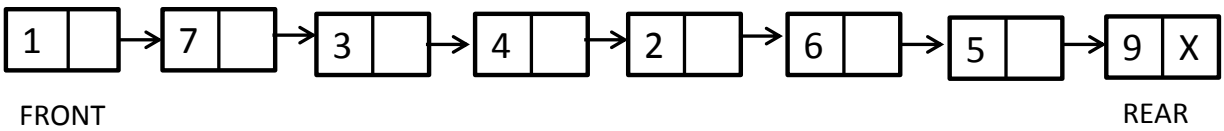


Fig. Linked queue after inserting a new node

Algorithm to insert an element in a linked queue

1. Allocate memory for the new node and name it as PTR
2. SET PTR -> DATA = VAL
3. IF FRONT = NULL
   SET FRONT = REAR = PTR
   SET FRONT -> NEXT = REAR -> NEXT = NULL
   ELSE
   SET REAR -> NEXT = PTR
   SET REAR = PTR
   SET REAR -> NEXT = NULL
   [END OF IF]
4. END

Delete Operation

The delete operation is used to delete the element that is first inserted in a queue, i.e., the element whose address is stored in FRONT. However, before deleting the value, we must first check if FRONT=NULL because if this is the case, then the queue is empty and no more deletions can be done. If an attempt is made to delete a value from a queue that is already empty, an underflow message is printed.

Contd.

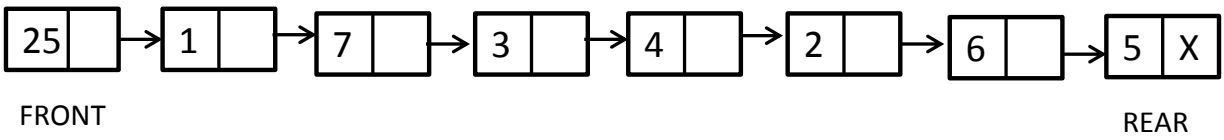


Fig. Linked queue

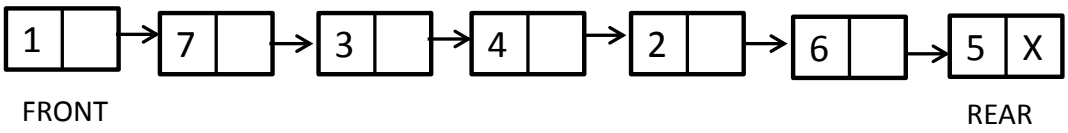


Fig. Linked queue after deletion of an element

Algorithm to delete an element from a linked queue

1. IF FRONT = NULL Write "Underflow“ &
   Go to Step 5
2. SET PTR = FRONT
3. SET FRONT = FRONT -> NEXT
4. FREE PTR
5. END

Contd.

Types of Queues:

A queue data structure can be classified into the following types:

- Circular Queue
- Dequeue
- Priority Queue
- Multiple Queue

Circular Queue:

In linear queues, we have discussed so far that insertions can be done only at one end called the REAR and deletions are always done from the other end called the FRONT.

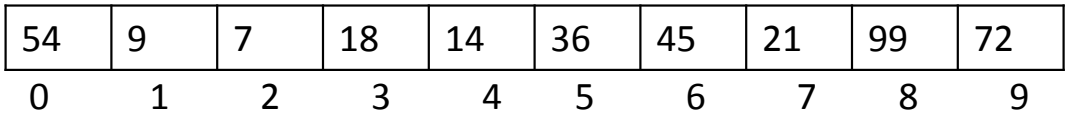
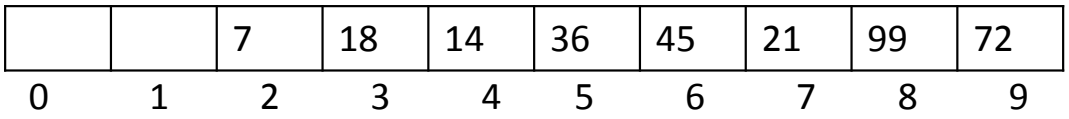


Fig. Linear Queue

Now, if you want to insert another value, it will not be possible because the queue is completely full. There is no empty space where the value can be inserted. Consider a scenario in which two successive deletions are made.

Contd.



Front=2
Rear=9

Fig. Queue after two successive deletions

Suppose we want to insert a new element in the queue shown above. Even though there is space available, the overflow condition still exists because the condition REAR = MAX – 1 still holds true. This is a major drawback of a linear queue. To resolve this problem, we have two solutions. First, shift the elements to the left so that the vacant space can be occupied and utilized efficiently. But this can be very time-consuming, especially when the queue is quite large.

Contd.

The second option is to use a circular queue. In the circular queue, the first index comes right after the last index. Conceptually, you can think of a circular queue.

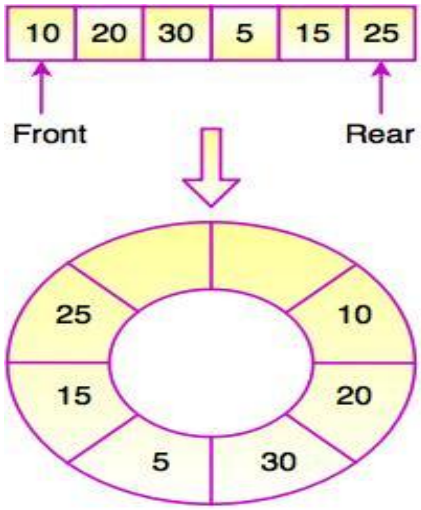


Fig. Circular Queue

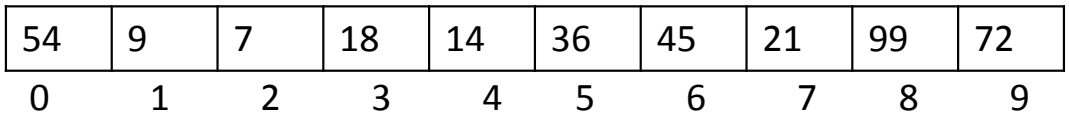
The circular queue will be full only when FRONT = 0 and REAR = Max – 1. A circular queue is implemented in the same manner as a linear queue is implemented. The only difference will be in the code that performs insertion and deletion operations.

Insert an element in a circular queue

For insertion, we have to check for the following three conditions:

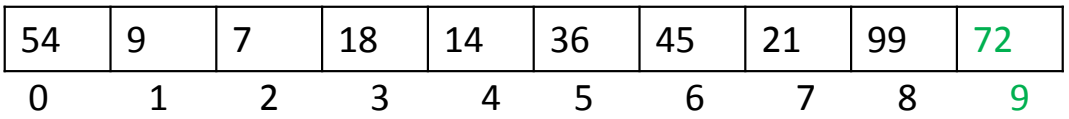
- If FRONT = 0 and REAR = MAX – 1, then the circular queue is full
- If REAR != MAX – 1 , then REAR will be incremented and the value will be inserted that position
- If FRONT != 0 and REAR = MAX – 1, then it means that the queue is not full. So, set REAR = 0 and insert the new element there

Contd.



Front=0
Rear=9

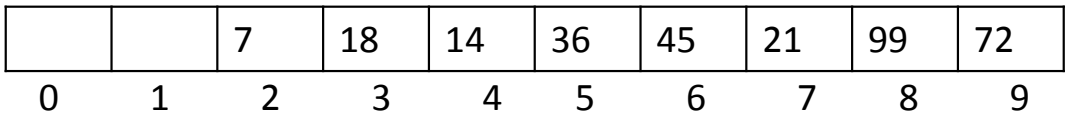
Fig. a) Full queue



Front=0
Rear=8

Increment rear so that it points to location 9 and insert the value there

Fig. b) Queue with vacant locations



Front=2
Rear=9
Set Rear=0 & insert value there

Fig. b) Queue with vacant locations

Algo: Insert an element in a Circular Queue

1. IF FRONT = 0 and REAR = MAX - 1
   Write "OVERFLOW" & Goto Step 4
2. IF FRONT = -1 and REAR = -1
   SET FRONT = REAR = 0
   ELSE IF REAR= MAX-1 and FRONT != 0
   SET REAR = 0
   ELSE
   SET REAR = REAR + 1
   [END OF IF]
3. SET QUEUE[REAR] = VAL
4. EXIT

Delete an element from a circular queue

To delete an element, again we check for three conditions.

- If FRONT = –1, then there are no elements in the queue. So, an underflow condition will be reported.

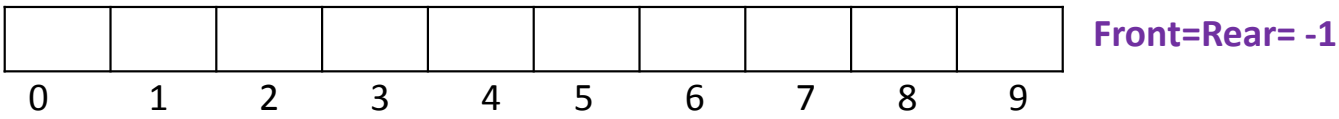
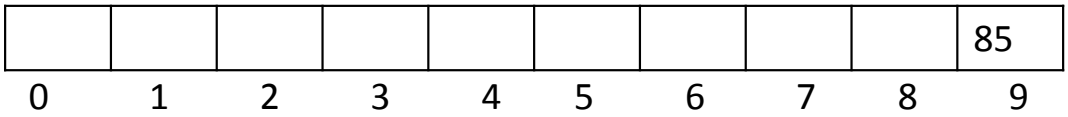


Fig. Empty Queue

Contd.

- If the queue is not empty and FRONT = REAR, then after deleting the element at the front the queue becomes empty and so FRONT and REAR are set to –1.

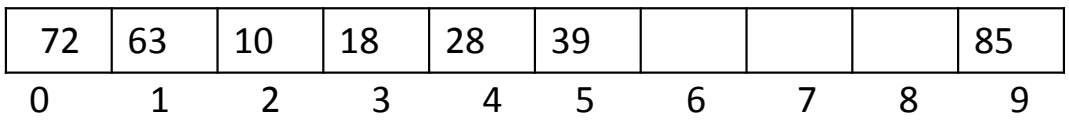


FRONT=REAR= 9

Delete the element(85) and set REAR = FRONT = -1

Fig. Queue with a single element

- If the queue is not empty and FRONT = MAX–1, then after deleting the element at the front, FRONT is set to 0.



FRONT= 9
REAR= 5

Delete the element(85) and set FRONT= 0

Fig. Queue where FRONT = MAX-1 before deletion

Algo: Delete an element from a circular queue

1. IF FRONT = -1 Write "UNDERFLOW“ & Goto Step 4.
2. SET VAL = QUEUE[FRONT]
3. IF FRONT = REAR
   SET FRONT = REAR = -1
   ELSE
     IF FRONT = MAX -1
          SET FRONT = 0
     ELSE
          SET FRONT = FRONT + 1
     [END of IF]
   [END OF IF]
4. EXIT

Double-ended-queue(deque)

A deque is a list in which the elements can be inserted or deleted at either end. It is also known as a *head-tail linked list* because elements can be added to or removed from either the front (head) or the back (tail) end.

However, no element can be added and deleted from the middle. In the computer's memory, a deque is implemented using either a circular array or a circular doubly linked list. In a deque, two pointers are maintained, LEFT and RIGHT, which point to either end of the deque. The elements in a deque extend from the LEFT end to the RIGHT end and since it is circular, Dequeue[N–1] is followed by Dequeue[0].

Contd.

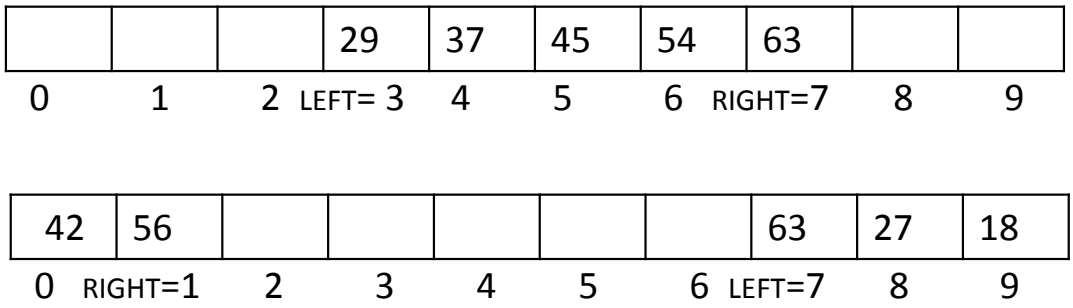


Fig. Double-ended queues

There are *two* variants of a double-ended queue.

- Input restricted deque: in this dequeue, insertions can be done only at one of the ends, while deletions can be done from both ends.
- Output restricted deque: in this dequeue, deletions can be done only at one of the ends, while insertions can be done on both ends.

Priority Queues

A priority queue is a data structure in which each element is assigned a priority. The priority of the element will be used to determine the order in which the elements will be processed. The general rules of processing the elements of a priority queue are

- An element with higher priority is processed before an element with a lower priority
- Two elements with the same priority are processed on a first-come-first-served (FCFS) basis

Contd.

A priority queue can be thought of as a modified queue in which when an element has to be removed from the queue, the one with the highest-priority is retrieved first. The priority of the element can be set based on various factors. Priority queues are widely used in operating systems to execute the highest priority process first. The priority of the process may be set based on the CPU time it requires to get executed completely. For example, if there are three processes, where the first process needs 5 ns to complete, the second process needs 4 ns, and the third process needs 7 ns, then the second process will have the highest priority and will thus be the first to be executed. However, CPU time is not the only factor that determines the priority, rather it is just one among several factors. Another factor is the importance of one process over another.

Contd.

Implementation of a Priority Queue

There are two ways to implement a priority queue. We can either use a sorted list to store the elements so that when an element has to be taken out, the queue will not have to be searched for the element with the highest priority or we can use an unsorted list so that insertions are always done at the end of the list. Every time when an element has to be removed from the list, the element with the highest priority will be searched and removed. While a sorted list takes O(n) time to insert an element in the list, it takes only O(1) time to delete an element. On the contrary, an unsorted list will take O(1) time to insert an element and O(n) time to delete an element from the list.

Contd.

Linked Representation of a Priority Queue

In the computer memory, a priority queue can be represented using arrays or linked lists. When a priority queue is implemented using a linked list, then every node of the list will have three parts:

a) the information or data part

b) the priority number of the element

c) the address of the next element.

If we are using a sorted linked list, then the element with the higher priority will precede the element with the lower priority.

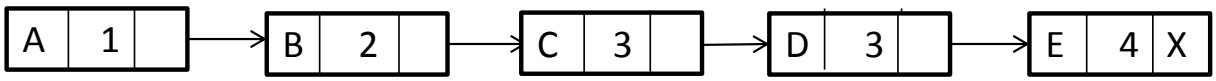


Fig. Priority Queue

Contd.

Lower priority number means higher priority. For example, if there are two elements A and B, where A has a priority number 1 and B has a priority number 5, then A will be processed before B as it has higher priority than B.

The priority queue in previous Fig. is a sorted priority queue having five elements. From the queue, we cannot make out whether A was inserted before E or whether E joined the queue before A because the list is not sorted based on FCFS. Here, the element with a higher priority comes before the element with a lower priority. However, we can definitely say that C was inserted in the queue before D because when two elements have the same priority the elements are arranged and processed on FCFS principle.

Insertion:

When a new element has to be inserted in a priority queue, we have to traverse the entire list until we find a node that has a priority lower than that of the new element. The new node is inserted before the node with the lower priority. However, if there exists an element that has the same priority as the new element, the new element is inserted after that element.

Contd.

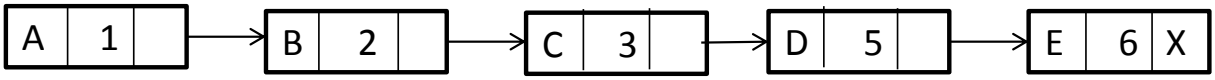


Fig. Priority Queue

If we have to insert a new element with data = F and priority number = 4, then the element will be inserted before D that has priority number 5, which is lower priority than that of the new element. So, the priority queue now becomes as shown below.

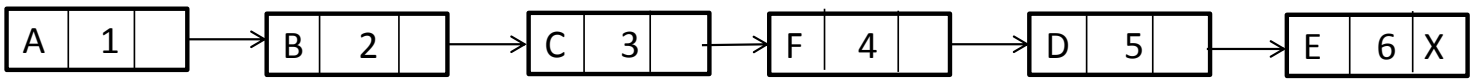


Fig. Priority queue after insertion of a new node

Contd.

If we have a new element with data = F and priority number = 2, then the element will be inserted after B, as both these elements have the same priority but the insertions are done on FCFS basis.

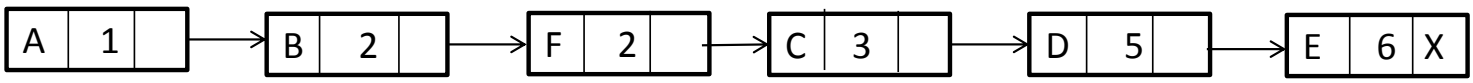


Fig. Priority queue after insertion of a new node

Deletion

Deletion is a very simple process in this case. The first node of the list will be deleted and the data of that node will be processed first.

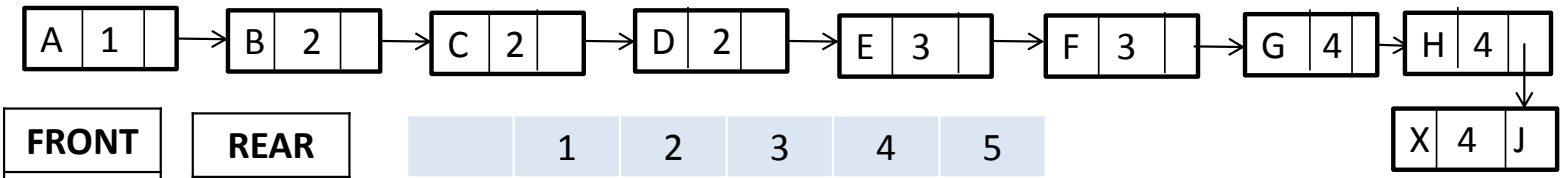
Contd.

Array Representation of a Priority Queue:

When arrays are used to implement a priority queue, then a separate queue for each priority number is maintained. Each of these queues will be implemented using circular arrays or circular queues. Every individual queue will have its own FRONT and REAR pointers.

We use a two-dimensional array for this purpose where each queue will be allocated the same amount of space. Look at the two-dimensional representation of a priority queue. Given the FRONT and REAR values of each queue, the two-dimensional matrix can be formed as:

Contd.



FRONT
3
1
4
4

REAR
3
3
5
1

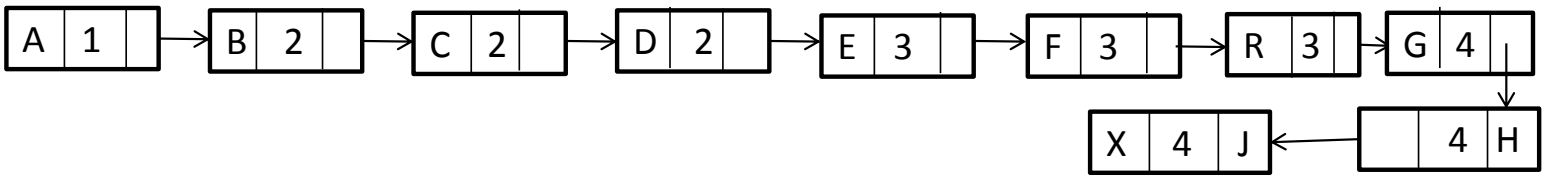
	1	2	3	4	5
1			A		
2	B	C	D		
3				E	F
4	J			G	H

Fig. Priority queue matrix

FRONT[K] and REAR[K] contain the front and rear values of row K, where K is the priority number. Note that here we are assuming that the row and column indices start from 1, not 0. Obviously, while programming, we will not take such assumptions.

Insertion: To insert a new element with priority K in the priority queue, add the element at the rear end of row K, where K is the row number as well as the priority number of that element. For example, if we have to insert an element R with priority number 3, then what will be the Priority queue matrix?

Contd.



FRONT
3
1
4
4

REAR
3
3
1
1

	1	2	3	4	5
1			A		
2	B	C	D		
3	R			E	F
4	J			G	H

Fig. Priority queue matrix after insertion of a new element

Deletion:

To delete an element, we find the first non-empty queue and then process the front element of the first non-empty queue. In our priority queue, the first non-empty queue is the one with priority number 1 and the front element is A, so A will be deleted and processed first. In technical terms, find the element with the smallest K, such that FRONT[K] != NULL.

Contd.

Multiple Queue

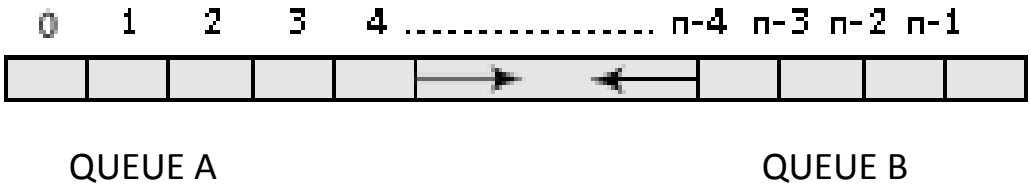


Fig. Multiple Queues