QUEUE ADT

- A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.
- Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue)
- Queue is referred to be as First In First Out list. i.e., the data item stored first will be accessed first.
- For example, people waiting in line for a rail ticket form a queue.

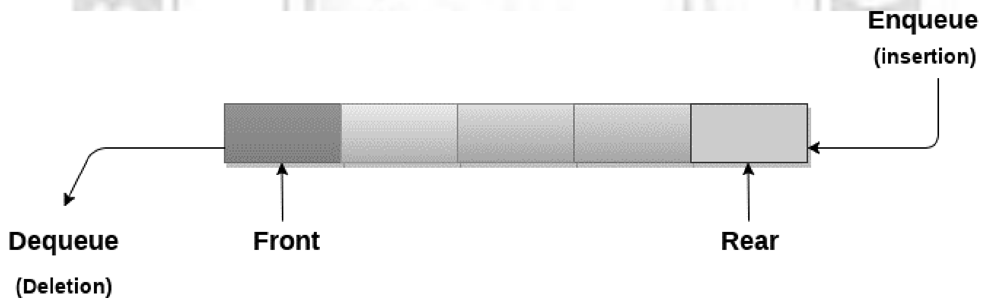


Fig. 3.27: Data Structure of Queue

Applications of Queue

Due to the fact that queue performs actions on first in first out basis which is quite fair for the ordering of actions. There are various applications of queues discussed as below.

- Queues are widely used as waiting lists for a single shared resource like printer, disk, CPU.
- Queues are used in asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.

- Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.
- Queue are used to maintain the play list in media players in order to add and remove the songs from the play-list.
- Queues are used in operating systems for handling interrupts.

Basic Operations in Queue

- **Enqueue** operation: The term "enqueue" refers to the act of adding a new element to a queue.
- **Dequeue** operation: Dequeue is the process of deleting an item from a queue. We must delete the queue member that was put first since the queue follows the FIFO principle. **Front** Operation: This works similarly to the peek operation in stacks in that it returns the value of the first element without deleting it.
- **isEmpty** Operation: The isEmpty() function is used to check if the Queue is empty or not.
- **isFull** Operation: The isFull() function is used to check if the Queue is full or not.

Types of Queue

- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

Simple Queue or Linear Queue

- In Linear Queue, an insertion takes place from one end while the deletion occurs from another end. The end at which the insertion takes place is known as the rear end, and the end at which the deletion takes place is known as front end. It strictlyfollows the FIFO rule.
- The major drawback of using a linear Queue is that insertion is done only from the rear end. If the first three elements are deleted from the Queue, we cannot insert more elements even though the space is available in a Linear Queue. In this case, the linear Queue shows the overflow condition as the rear is pointing to the last element of the Queue

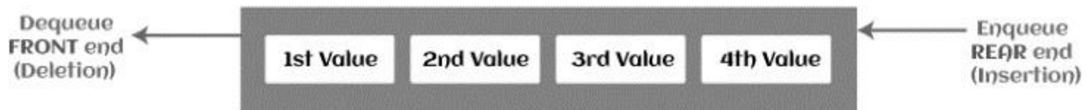


Fig. 3.28 Representation of Linear Queue

Circular Queue

- In Circular Queue, all the nodes are represented as circular. It is similar to the linear Queue except that the last element of the queue is connected to the first element. It is also known as Ring Buffer, as all the ends are connected to another end.
- The drawback that occurs in a linear queue is overcome by using the circular queue. If the empty space is available in a circular queue, the new element can be added in an empty space by simply incrementing the value of rear. The main advantage of using the circular queue is better memory utilization.

Fig. 3.29 Representation of circular Queue

Priority Queue

- It is a special type of queue in which the elements are arranged based on the priority. It is a special type of queue data structure in which every element has a priority associated with it. Suppose some elements occur with the same priority, they will be arranged according to the FIFO principle.
- Insertion in priority queue takes place based on the arrival, while deletion in the priority queue occurs based on the priority. Priority queue is mainly used to implement the CPU scheduling algorithms.

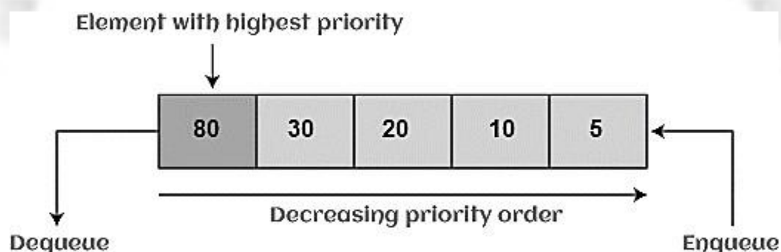


Fig. 3.30 Representation of Priority Queue

There are two types of priority queue

1. **Ascending priority queue** - In ascending priority queue, elements can be inserted in arbitrary order, but only smallest can be deleted first.
2. **Descending priority queue** - In descending priority queue, elements can be inserted in arbitrary order, but only the largest element can be deleted first.

Deque (or, Double Ended Queue)

- In Deque or Double Ended Queue, insertion and deletion can be done from both ends of the queue either from the front or rear. It means that we can insert and delete elements from both front and rear ends of the queue. Deque can be used as a palindrome checker means that if we read the string from both ends, then the string would be the same.
- Deque can be used both as stack and queue as it allows the insertion and deletion operations on both ends.

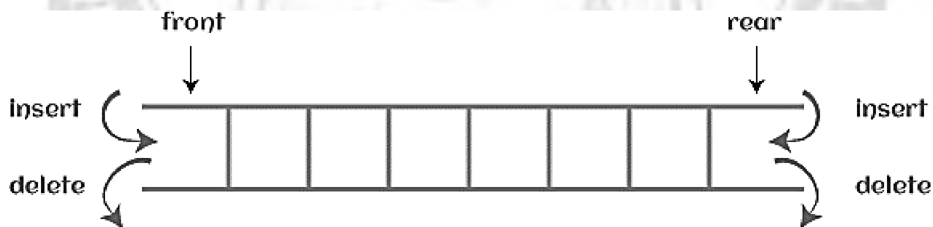


Fig. 3.31 Representation of Double Ended Queue

There are two types of Deque

- **Input restricted Deque** - As the name implies, in input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.
- **Output restricted Deque** - As the name implies, in output restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.

PRIORITY QUEUES

- A priority queue is an abstract data type that behaves similarly to the normal queue except that each element has some priority, i.e., the element with the highest priority would come first in a priority queue. The priority of the elements in a priority queue will determine the order in which elements are removed from the priority queue.

- The priority queue supports only comparable elements, which means that the elements are either arranged in an ascending or descending order.
- For example, suppose we have some values like 1, 3, 4, 8, 14, 22 inserted in a priority queue with an ordering imposed on the values is from least to the greatest. Therefore, the 1 number would be having the highest priority while 22 will be having the lowest priority.

Characteristics of a Priority queue

A priority queue is an extension of a queue that contains the following characteristics:

- Every element in a priority queue has some priority associated with it.
- An element with the higher priority will be deleted before the deletion of the lesser priority.
- If two elements in a priority queue have the same priority, they will be arranged using the FIFO principle.

Let's understand the priority queue through an example.

We have a priority queue that contains the following

values:1, 3, 4, 8, 14, 22

All the values are arranged in ascending order. Now, we will observe how the priority queue will look after performing the following operations:

poll(): This function will remove the highest priority element from the priority queue. In the above priority queue, the '1' element has the highest priority, so it will be removed from the priority queue.

add(2): This function will insert '2' element in a priority queue. As 2 is the smallest element among all the numbers so it will obtain the highest priority.

poll(): It will remove '2' element from the priority queue as it has the highest priority queue.

add(5): It will insert 5 element after 4 as 5 is larger than 4 and lesser than 8, so it will obtain the third highest priority in a priority queue.

Types of Priority Queue

There are two types of priority queue:

Ascending order priority queue

In ascending order priority queue, a lower priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in an ascending order like 1,2,3,4,5; therefore, the smallest number, i.e., 1 is given as the highest priority in a priority queue.

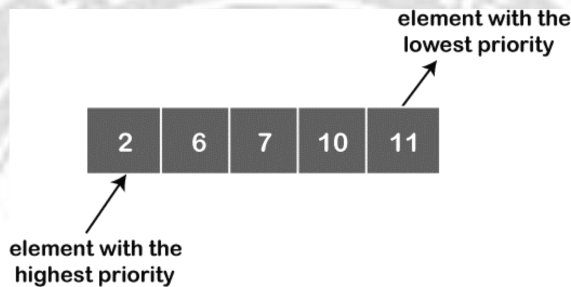


Fig. 3.32 Ascending order priority queue

Descending order priority queue:

In descending order priority queue, a higher priority number is given as a higher priority in a priority. For example, we take the numbers from 1 to 5 arranged in descending order like 5, 4, 3, 2, 1; therefore, the largest number, i.e., 5 is given as the highest priority in a priority queue.

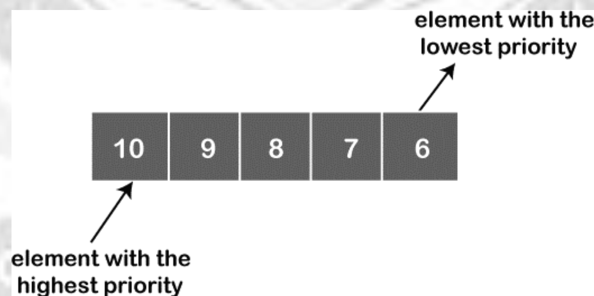


Fig. 3.33 Descending order priority queue

Representation of priority queue

Now, let's see how to represent the priority queue through a one-way list.

We will create the priority queue by using the list given below in which INFO list contains the data elements, PRN list contains the priority numbers of each data element available in the INFO list, and LINK basically contains the address of the next node.

	INFO	PNR	LINK
0	200	2	4
1	400	4	2
2	500	4	6
3	300	1	0
4	100	2	5
5	600	3	1
6	700	4	

Let's create the priority queue step by step.

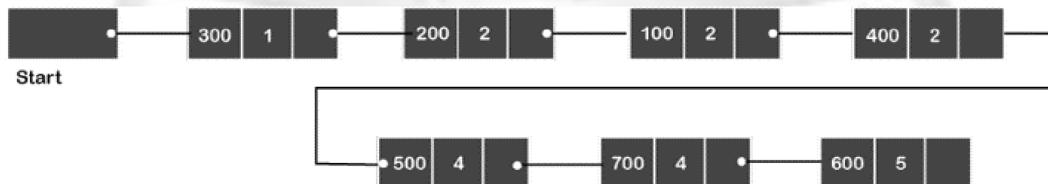
In the case of priority queue, lower priority number is considered the higher priority, i.e., lower priority number = higher priority.

Step 1: In the list, lower priority number is 1, whose data value is 333, so it will be inserted in the list as shown in the below diagram:

Step 2: After inserting 333, priority number 2 is having a higher priority, and data values associated with this priority are 222 and 111. So, this data will be inserted based on the FIFO principle; therefore 222 will be added first and then 111.

Step 3: After inserting the elements of priority 2, the next higher priority number is 4 and data elements associated with 4 priority numbers are 444, 555, 777. In this case, elements would be inserted based on the FIFO principle; therefore, 444 will be added first, then 555, and then 777.

Step 4: After inserting the elements of priority 4, the next higher priority number is 5, and the value associated with priority 5 is 666, so it will be inserted at the end of the queue.



Implementation of Priority Queue

- The priority queue can be implemented in four ways that include arrays, linked list, heap data structure and binary search tree. The heap data structure is the most efficient way of implementing the priority queue, so we will implement the

priority queue using a heap data structure in this topic. Now, first we understand the reason why heap is the most efficient way among all the other data structures.

Analysis of complexities using different implementations

- A comparative analysis of different implementations of priority queue is given in table 3.4

Table 3.4 Analysis of priority queue

Operations	peek	insert	delete
Linked List	O(1)	O(n)	O(1)
Binary Heap	O(1)	O(log n)	O(log n)
Binary Search Tree	O(1)	O(log n)	O(log n)

Heap

- A heap is a tree-based data structure that forms a complete binary tree, and satisfies the heap property.
- If A is a parent node of B, then A is ordered with respect to the node B for all nodes A and B in a heap.
- It means that the value of the parent node could be more than or equal to the value of the child node, or the value of the parent node could be less than or equal to the value of the child node.
- Therefore, we can say that there are two types of heaps:

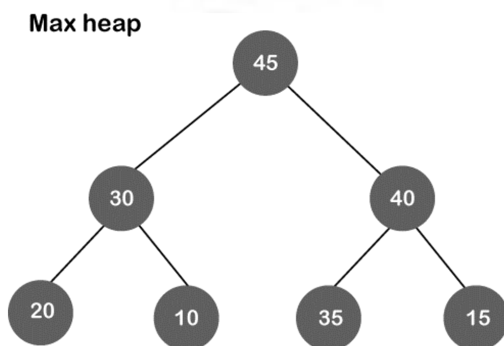


Fig. 3.34 Max heap

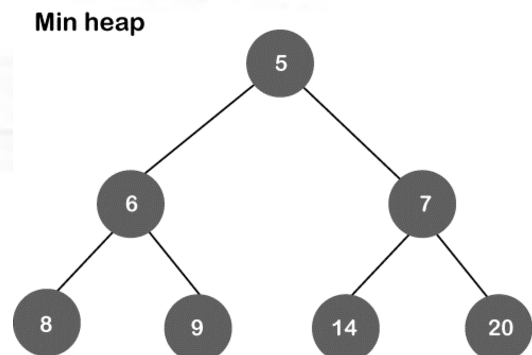


Fig. 3.35 Min heap

- **Max heap:** The max heap is a heap in which the value of the parent node is greater than the value of the child nodes.

- **Min heap:** The min heap is a heap in which the value of the parent node is less than the value of the child nodes.

Priority Queue Operations

The common operations that we can perform on a priority queue are

- Insertion,
- Deletion and
- Peek

Let's see how we can maintain the heap data structure.

Inserting the element in a priority queue (max heap)

If we insert an element in a priority queue, it will move to the empty slot by looking from top to bottom and left to right.

If the element is not in a correct place then it is compared with the parent node; if it is found out of order, elements are swapped. This process continues until the element is placed in a correct position.

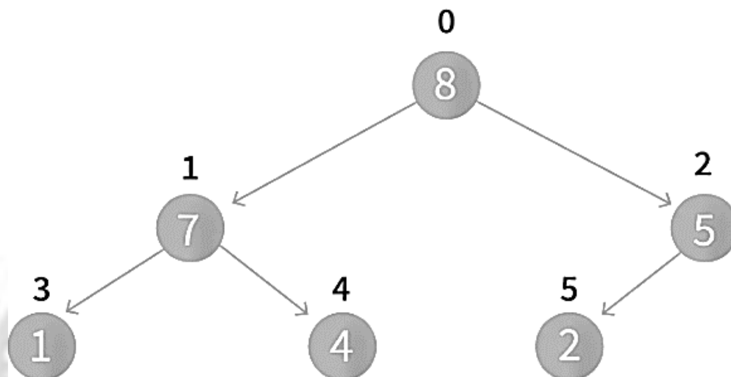
Algorithm 3.18

```
START
If(no node):
    Create node
Else:
    Insert node at end of heap
Heapify
END
```

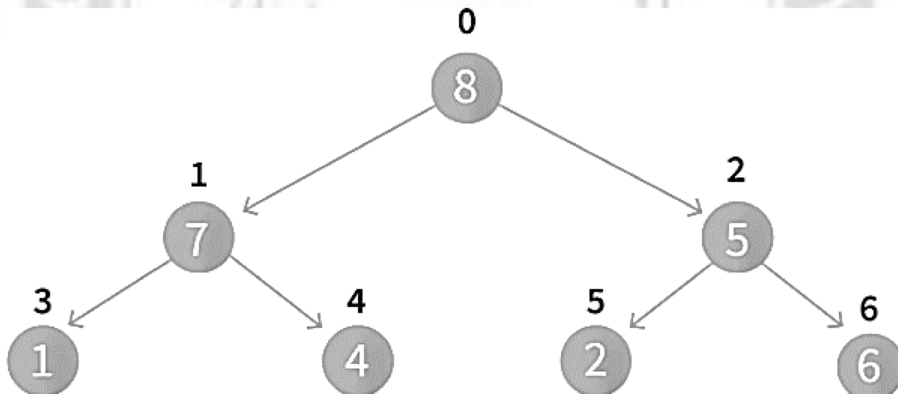
Let us now see with an example how this works:

Example 3.10

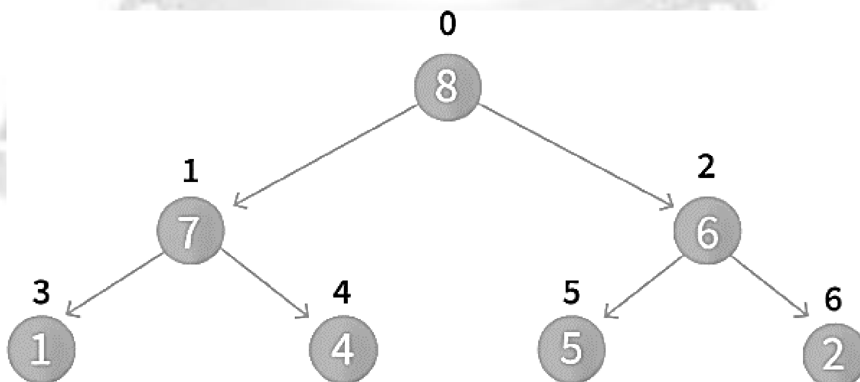
Let’s say the elements are 1,4,2,7,8,5. The max-heap of these elements would look like:



Now, let's try to insert a new element, 6. Since there are nodes present in the heap, we insert this node at the end of heap so it looks like this:



Then, heapify operation is implemented. After which, the heap will look like this:



Removing the minimum element from the priority queue

In a max heap, the maximum element is the root node. When we remove the root node, it creates an empty slot. The last inserted element will be added in this empty slot. Then, this element is compared with the child nodes, i.e., left-child and right child, and swap with the smaller of the two. It keeps moving down the tree until the heap property is restored.

Algorithm 3.19:

START

If node that needs to be deleted is a leaf node:

Remove the node

Else:

Swap node that needs to be deleted with the last leaf node present.

Remove the node

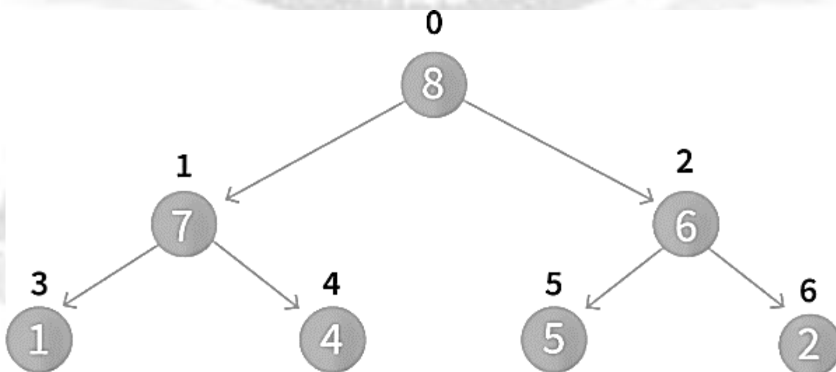
Heapify

END

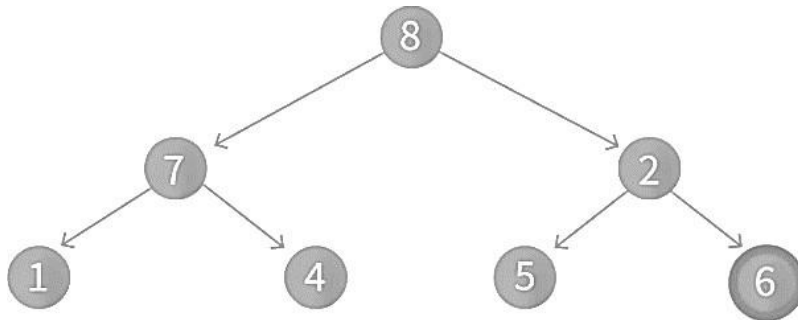
Let us now see with an example how this works:

Example: 3.11

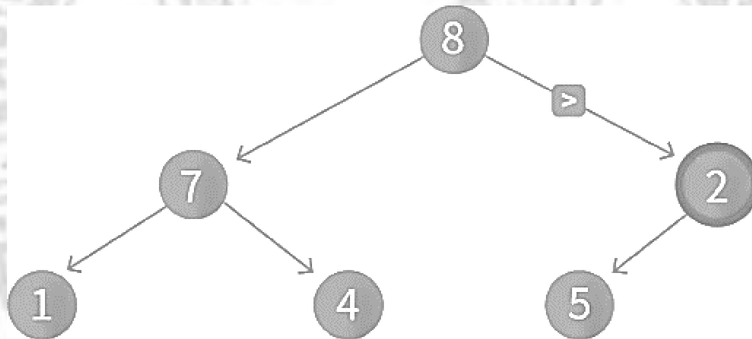
Let's say the elements are 1,4,2,7,8,5,6. The max-heap of these elements would look like:



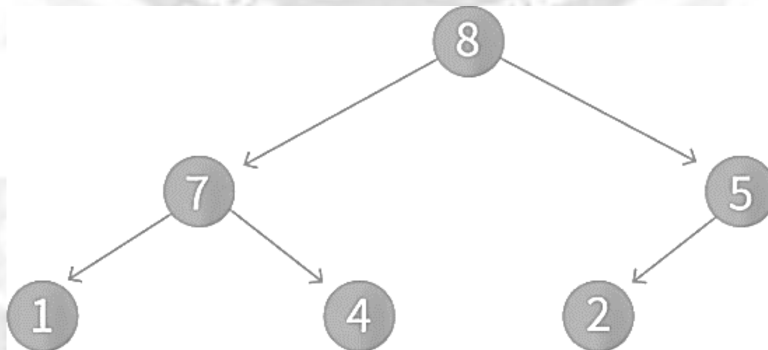
Now, let's try to delete an element, 6. Since this is not a leaf node, we swap it with the last leaf node so it looks like this:



Then, we remove the leaf node so it looks like this:



Then, heapify operation is implemented. After which, the heap will look like this:



Peeking the element from a priority queue

This will return the maximum element if a max-heap is used and the minimum number if a min-heap is used. To do both of these, we return root node. This is because in the max-heap or the min-heap the maximum or minimum element will be present at the root node respectively.

Applications of Priority Queue

The following are the applications of a Priority Queue:

- It is used in Djikstra's Algorithm – To find the shortest path between nodes in a graph.
- It is used in Prim's Algorithm – To find the Minimum Spanning Tree in a weighted undirected graph.
- It is used in Heap Sort – To sort the Heap Data Structure
- It is used in Huffman Coding – A Data Compression Algorithm
- It is used in Operating Systems for:
  - Priority Scheduling – Where processes must be scheduled according to their priority.
  - Load Balancing – Where network or application traffic must be balanced across multiple servers.
  - Interrupt Handling – When a current process is interrupted, a handler is assigned to the same to rectify the situation immediately.
- A* Search Algorithm – A graph traversal and a path search algorithm

QUEUE IMPLEMENTATION

Array implementation of Queue

- Queue can be represented easily by using linear arrays.
- There are two variables i.e. front and rear that are implemented in the case of every queue.
- Front and rear variables point to the position from where insertions and deletions are performed in a queue.
- Initially, the value of front and queue is -1 which represents an empty queue.

Array representation of a queue containing 5 elements along with the respective values of front and rear, is shown in figure 3.36.

The above figure 3.36 shows the queue of characters forming the English word "HELLO". Since, No deletion is performed in the queue till now, therefore the value of front remains -1. However, the value of rear increases by one every time an insertion is performed in the queue. After inserting an element into the queue shown in the above

figure, the queue will look something like following. The value of rear will become 5 while the value of front remains same.

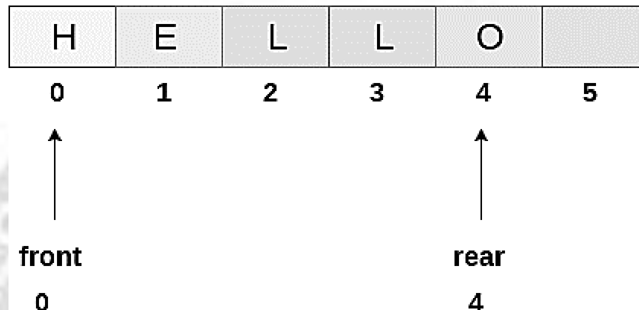


Fig. 3.36: Array representation of Queue

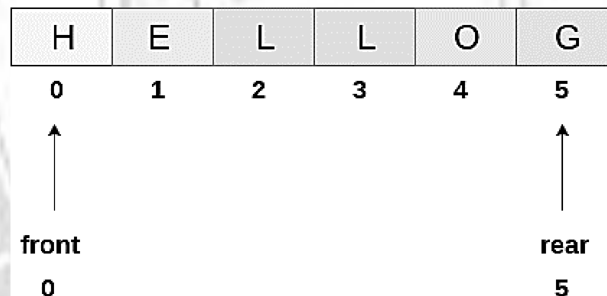


Fig. 3.37: Queue after inserting an element

After deleting an element, the value of front will increase from -1 to 0. However, the queue will look something like following.

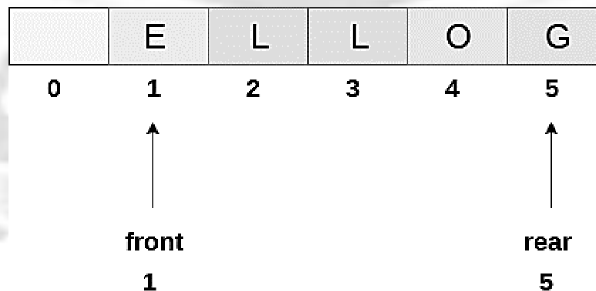


Fig. 3.38: Queue after deleting an element

Algorithm to insert any element in a queue

- Check if the queue is already full by comparing rear to max - 1. if so, then return an overflow error.

- If the item is to be inserted as the first element in the list, in that case set the value of front and rear to 0 and insert the element at the rear end.
- Otherwise keep increasing the value of rear and insert each element one by one having rear as the index.

Algorithm 3.20

Step 1: IF REAR = MAX - 1

Write OVERFLOW

Go to step

[END OF IF]

Step 2: IF FRONT = -1 and REAR = -

1SET FRONT = REAR = 0

ELSE

SET REAR = REAR + 1

[END OF IF]

Step 3: Set QUEUE[REAR] = NUM

Step 4: EXIT

Implementation using C Function:

```
void insert (int queue[], int max, int front, int rear, int item)
{
    if (rear + 1 == max)
    {
        printf("overflow");
    }
    else
    {
        if(front == -1 && rear == -1)
        {
            front = 0;
```

```
rear = 0;
```

```
        }
        else
        {
            rear = rear + 1;
        }
        queue[rear]=item;
    }
}
```

Algorithm to delete an element from the queue

- If, the value of front is -1 or value of front is greater than rear, write an underflow message and exit.
- Otherwise, keep increasing the value of front and return the item stored at the front end of the queue at each time.

Algorithm 3.21

Step 1: IF FRONT = -1 or FRONT > REAR

Write UNDERFLOW

ELSE

SET VAL = QUEUE[FRONT]

SET FRONT = FRONT + 1

[END OF IF]

Step 2: EXIT

Implementation using C Function

```
int delete (int queue[], int max, int front, int rear)
{
    int y;
    if (front == -1 || front > rear)
    {
        printf("underflow");
```

```
    }
    else
    {
        y = queue[front];
        if(front == rear)
        {
            front = rear = -1;
            else
            front = front + 1;
        }
        return y;
    }
}
```

Drawback of array implementation

- **Memory wastage:** The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.
- **Deciding the array size:** On of the most common problem with array implementation is the size of the array which requires to be declared in advance. Due to the fact that, the queue can be extended at runtime depending upon the problem, the extension in the array size is a time taking process and almost impossible to be performed at runtime since a lot of reallocations take place.

Linked List implementation of Queue

- One of the alternative of array implementation is linked list implementation of queue.
- The storage requirement of linked representation of a queue with n elements is O(n) while the time requirement for operations is O(1).
- In a linked queue, each node of the queue consists of two parts i.e. data part and the link part.

- Each element of the queue points to its immediate next element in the memory.
- In the linked queue, there are two pointers maintained in the memory i.e. front pointer and rear pointer.
- The front pointer contains the address of the starting element of the queue while the rear pointer contains the address of the last element of the queue.
- Insertion and deletions are performed at rear and front end respectively.
- If front and rear both are NULL, it indicates that the queue is

empty.The linked representation of queue is shown in the figure 3.39

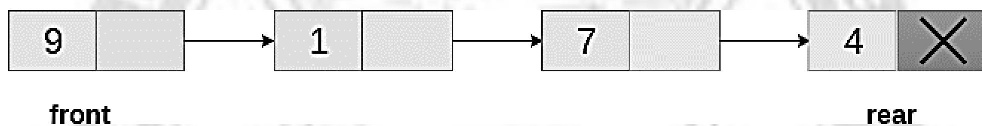


Fig. 3.39 Linked list representation of queue

Insertion on Linked Queue

- The insert operation append the queue by adding an element to the end of the queue.
- The new element will be the last element of the queue.
- There can be the two scenario of inserting this new node ptr into the linked queue.
- In the first scenario, we insert element into an empty queue. In this case, the condition front = NULL becomes true. Now, the new element will be added as the only element of the queue and the next pointer of front and rear pointer both, will point to NULL.
- In the second case, the queue contains more than one element. The condition front = NULL becomes false. In this scenario, we need to update the end pointer rear so that the next pointer of rear will point to the new node ptr.
- Since, this is a linked queue, hence we also need to make the rear pointer point to the newly added node ptr.
- Also make the next pointer of rear point to NULL.

Algorithm 3.22

Step 1: Allocate the space for the new node PTR

Step 2: SET PTR -> DATA = VAL

Step 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]

Step 4: END

Implementation using C Function

```
void insert(struct node *ptr, int item; )
{
   ptr = (struct node *) malloc (sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW\n");
      return;
   }
   else
   {
      ptr -> data = item;
      if(front == NULL)
      {
         front = ptr;
         rear = ptr;
         front -> next = NULL;
         rear -> next = NULL;
      }
```

```
        else
        {
            rear -> next = ptr;
            rear = ptr;
            rear->next = NULL;
        }
    }
}
```

APPLICATIONS OF QUEUE

There are several algorithms that use queues to give efficient running times. Some simple examples of queue usage are follows.

Type declarations for queue-array implementation

```
struct queue_record
{
unsigned int q_max_size; /* Maximum # of elements */
/* until Q is full */
unsigned int q_front;
unsigned int q_rear;
unsigned int q_size; /* Current # of elements in Q */
element_type *q_array;
};
typedef struct queue_record * QUEUE
```

Routine to test whether a queue is empty-array implementation

```
int
is_empty( QUEUE Q )
{
return( Q->q_size == 0 );
}
```

Routine to make an empty queue-array implementation

```
void
make_null ( QUEUE Q )
{
Q->q_size = 0;
Q->q_front = 1;
Q->q_rear = 0;
}
```

Routines to enqueue-array implementation

```
unsigned int
succ( unsigned int value, QUEUE Q )
{
if( ++value == Q->q_max_size )
value = 0;
return value;
}
void
enqueue( element_type x, QUEUE Q )
{
if( is_full( Q ) )
error("Full queue");
else
{
Q->q_size++;
Q->q_rear = succ( Q->q_rear, Q
);Q->q_array[ Q->q_rear ] = x;
}
}
```

Other applications

- When jobs are submitted to a printer, they are arranged in order of arrival. Thus, essentially, jobs sent to a line printer are placed on a queue.
- Virtually every real-life line is (supposed to be) a queue. For instance, lines at ticket counters are queues, because service is first-come first-served.
- Another example concerns computer networks. There are many network setups of personal computers in which the disk is attached to one machine, known as the file server. Users on other machines are given access to files on a first-come first-served basis, so the data structure is a queue.
- Calls to large companies are generally placed on a queue when all operators are busy.
- In large universities, where resources are limited, students must sign a waiting list if all terminals are occupied. The student who has been at a terminal the longest is forced off first, and the student who has been waiting the longest is the next user to be allowed on.