Parul® University

Unit 2
Stacks and queue

Prof. vishwa Dabhi, Assistant Professor
Computer science and Engineering

Topic-2

Queues

Topics Covered

- ADT Queue
- Types of Queues:
  - Simple Queue
  - Circular Queue
  - Priority Queue
- Operations on each types of queues:
  - Algorithms and their analysis

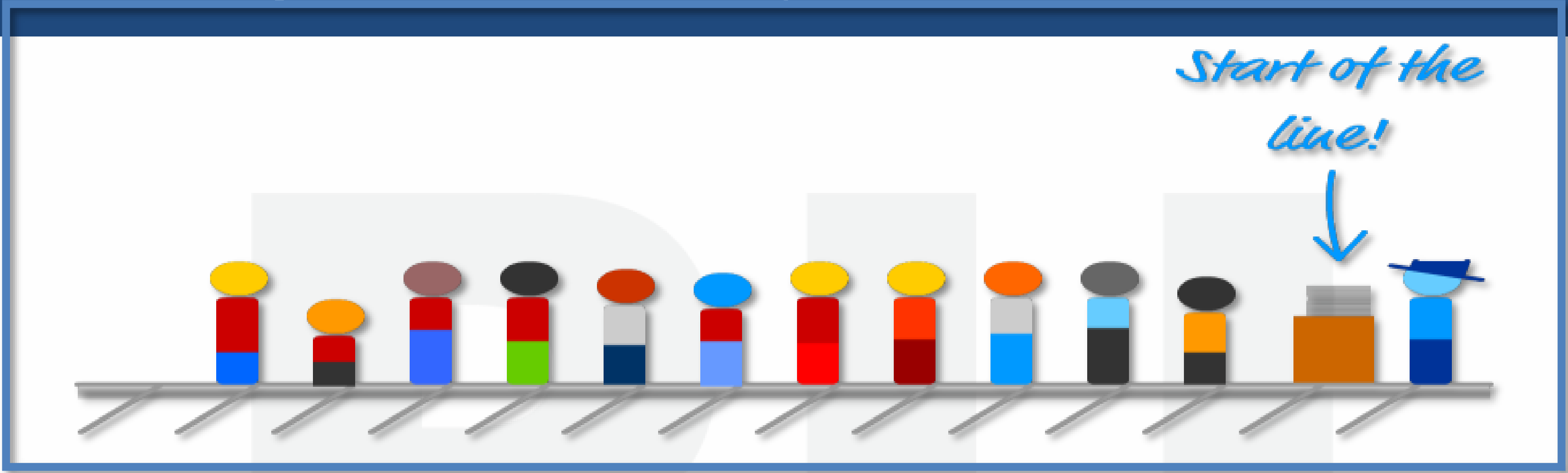
ADT(Abstract Data Type)

➢The queue abstract data type is defined by the following structure and operations:

▪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 rear.

➢Queues are also called as FIFO (First In First Out) list.

Pictorial Representation of Queue



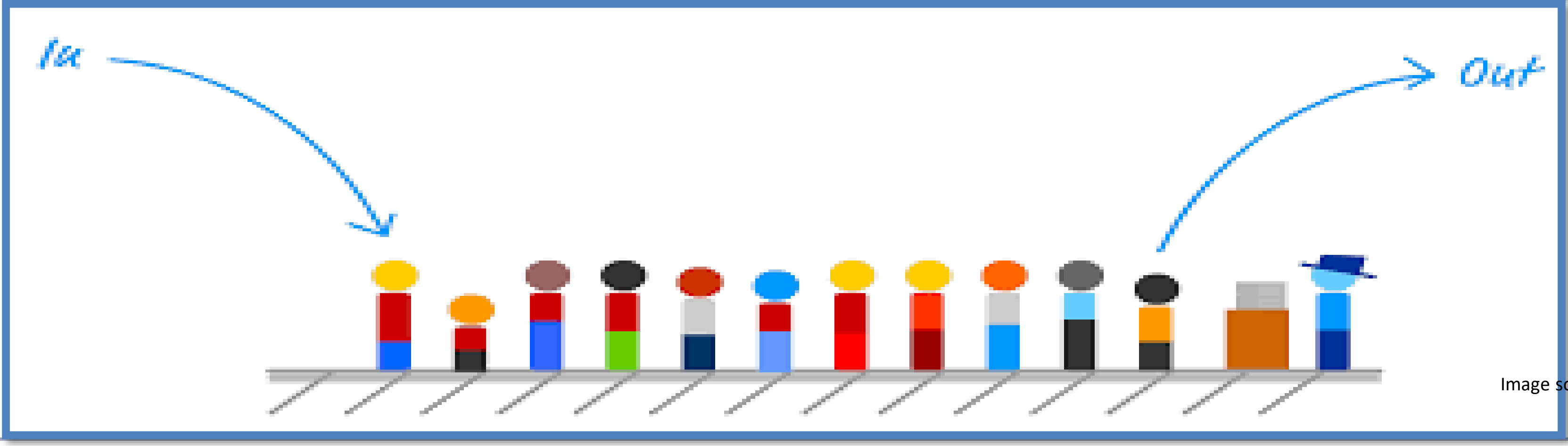


Image source : Google

Examples of Queue in day to day life

- Railway reservation counter
- Movie theatre ticket counter
- An Escalator
- A Car wash, etc.

▪Before getting the service, one has to wait in the queue. After getting the service, one leaves the queue.

▪Service is provided at one end(front/head) and people join at the other end(rear/back/tail).

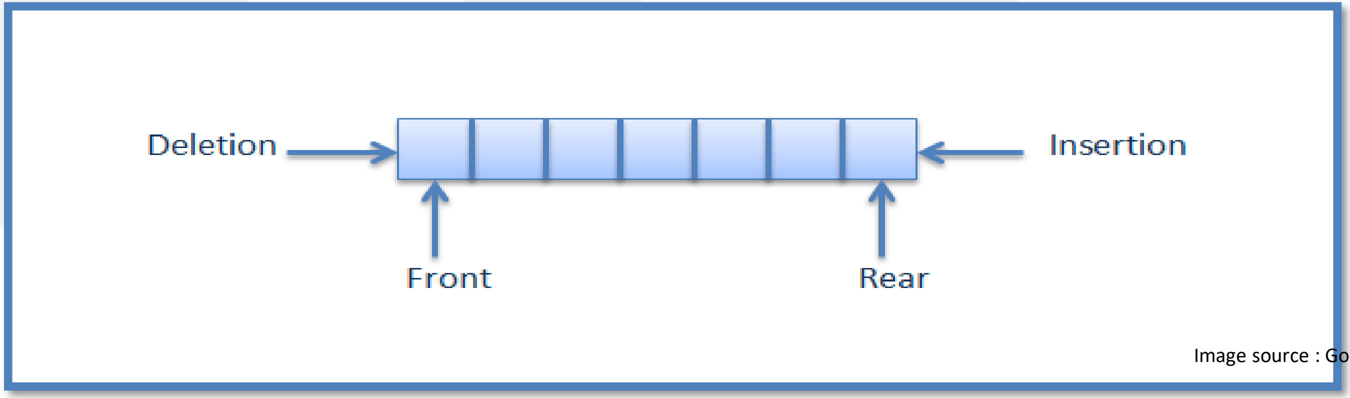


Image source : Google

Application of Queues in Computing

- Scheduling of processes (Round Robin Algorithm).
- Spooling (to maintain a queue of jobs to be printed).
- A queue of client processes waiting to receive the service from the server process).
- An input stream.

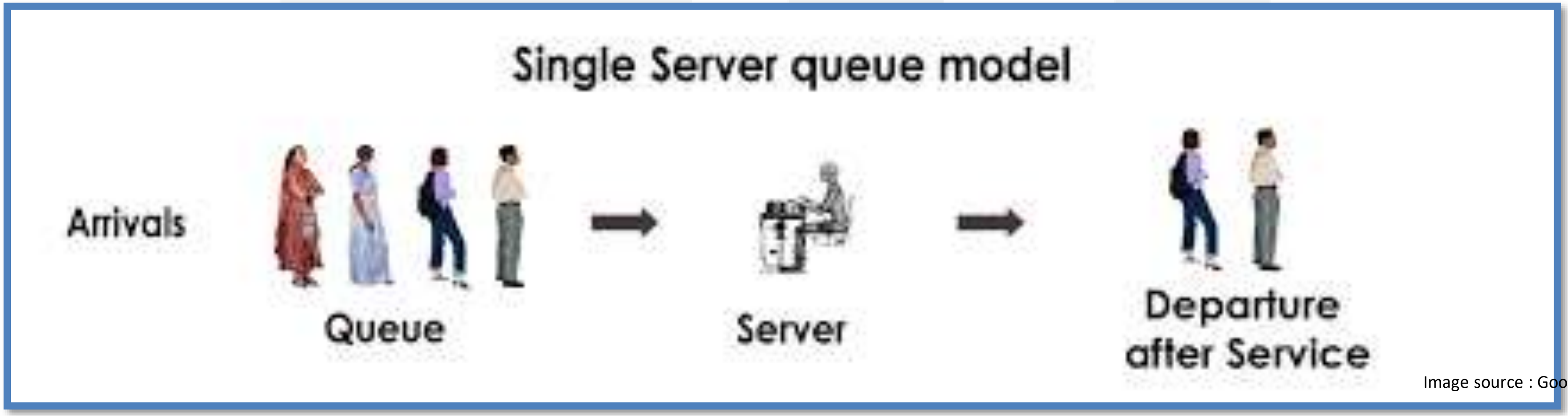


Image source : Google

Simple Queue / Linear Queue

- Simple queue defines the simple operation of queue in which insertion occurs at the rear of the list and deletion occurs at the front of the list.

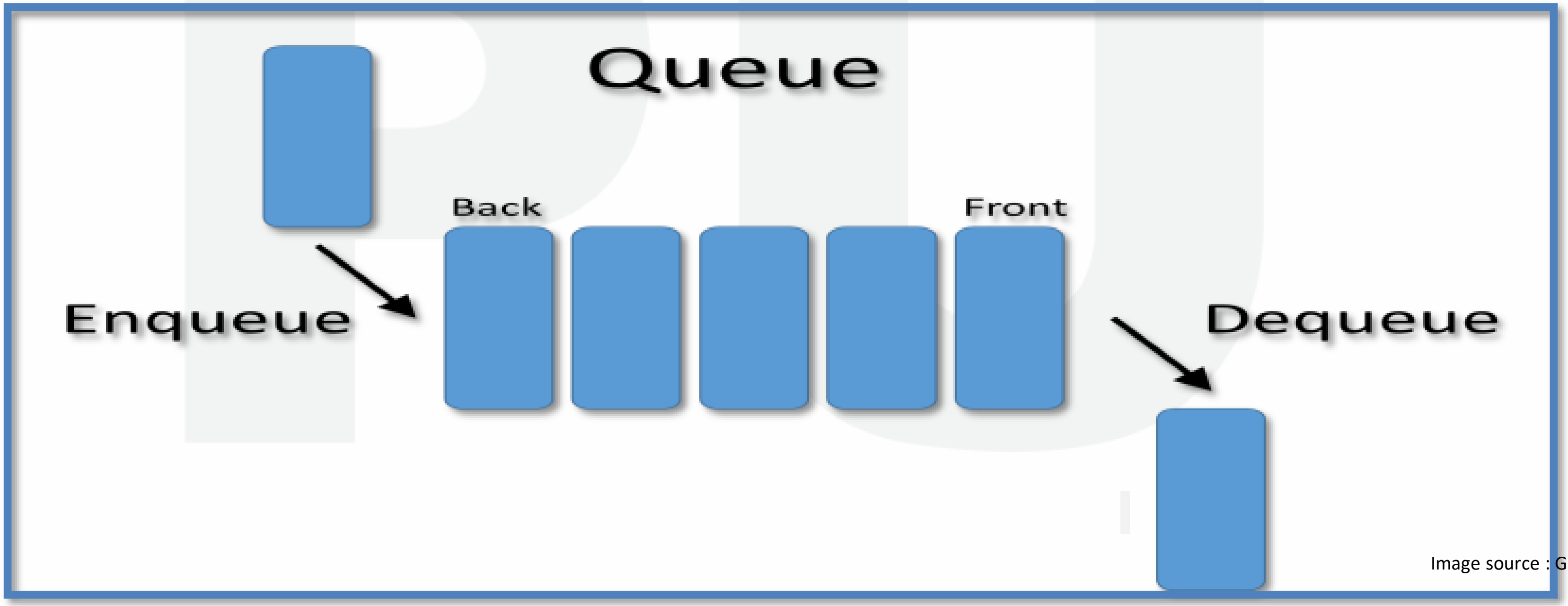


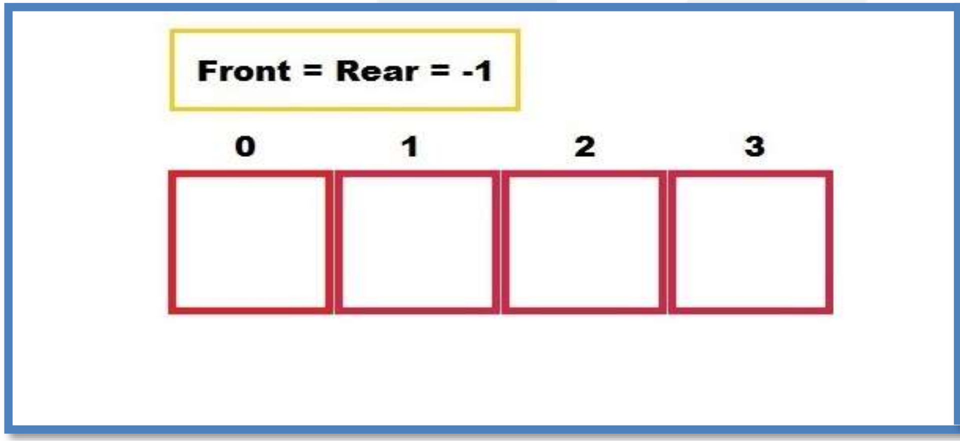
Image source : Google

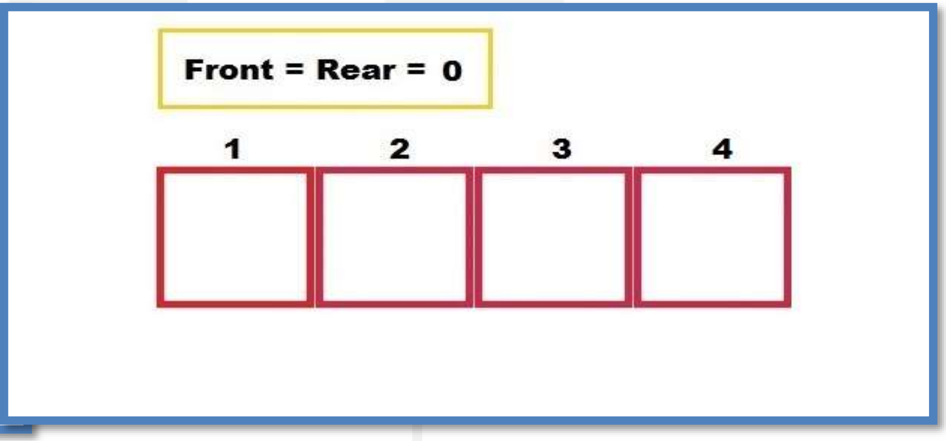
Operations on a queue:

- initialize(): Initializes a queue by setting the value of front and rear to -1.
- enqueue(): Inserts an element at the rear end of queue.
- dequeue()/delete(): Deletes an element from the front end of queue.
- empty(): Returns true(1) if queue is empty and returns false(0) otherwise.
- full(): Returns true(1) if queue is full and returns false(0) otherwise.
- print(): Printing queue elements.

➢ We can perform initialization operation of queue by giving initial value to front and rear pointer in queue by following rules:

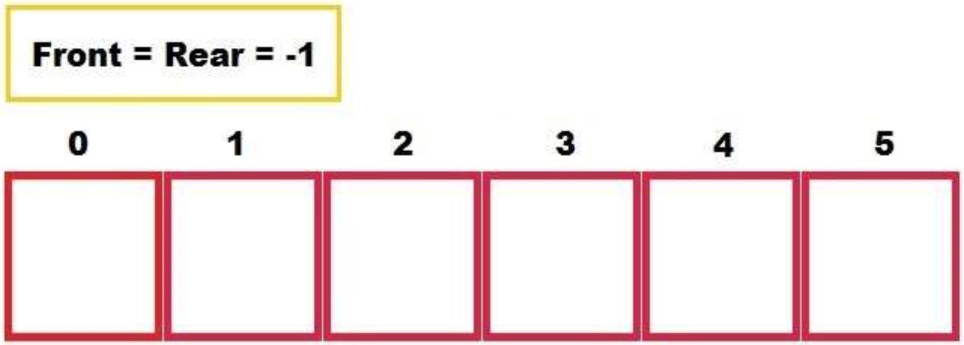
- Initialize front and rear to 0 if the element starts from 1.
- Initialize front and rear to -1 if the element starts from 0.





Example of Simple Queue

- Empty Queue

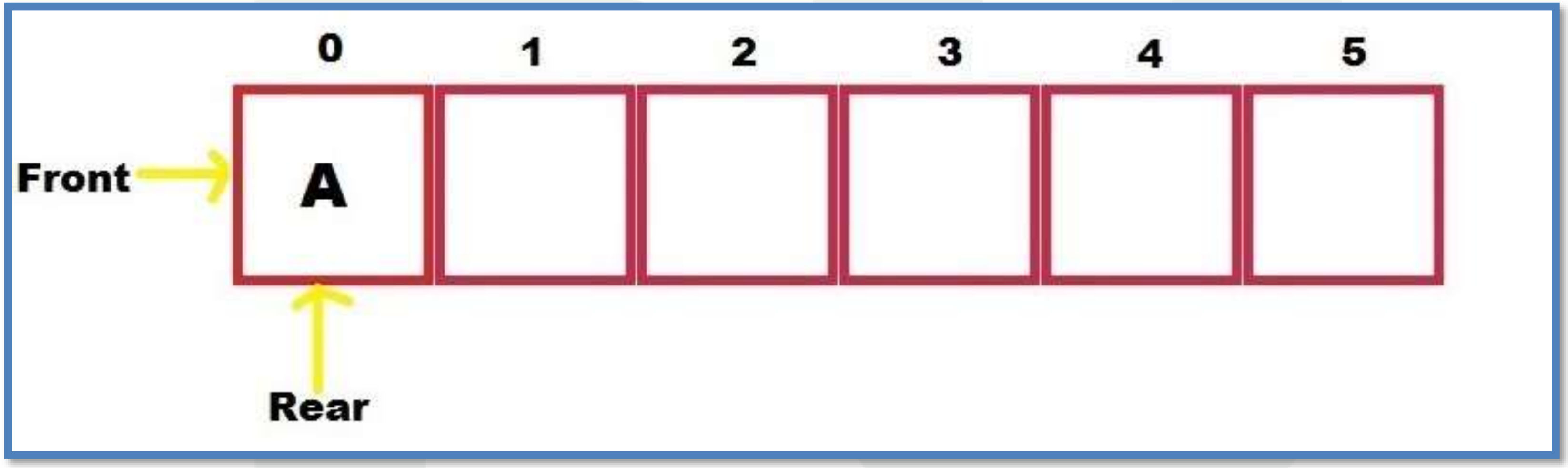


Front = Rear = -1

0 1 2 3 4 5

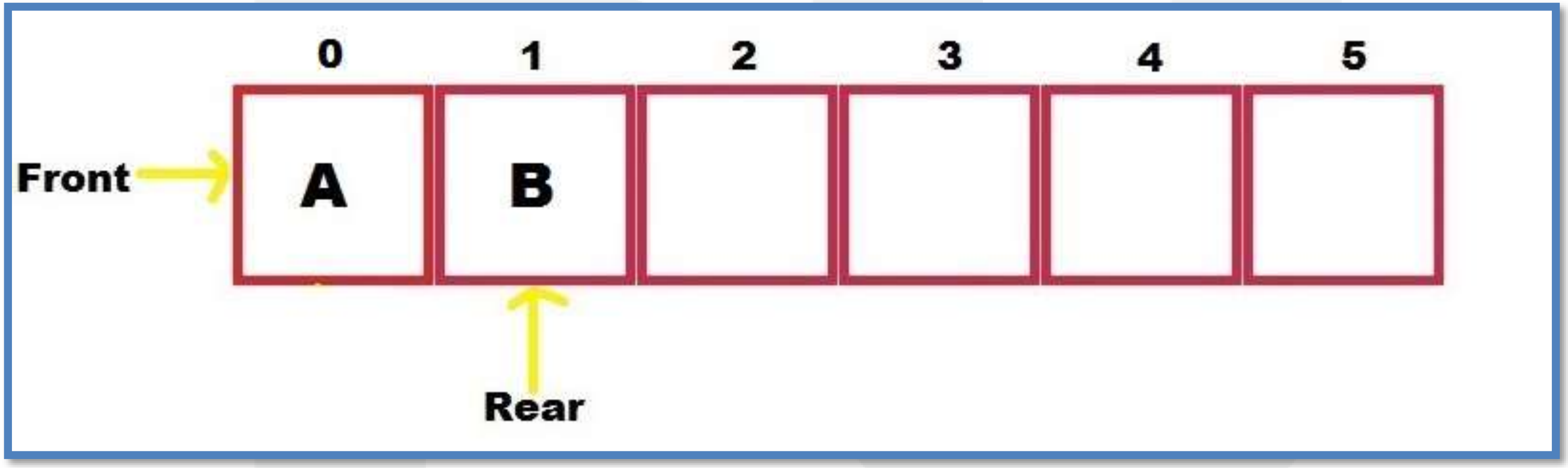
Example of Simple Queue

- Insert A



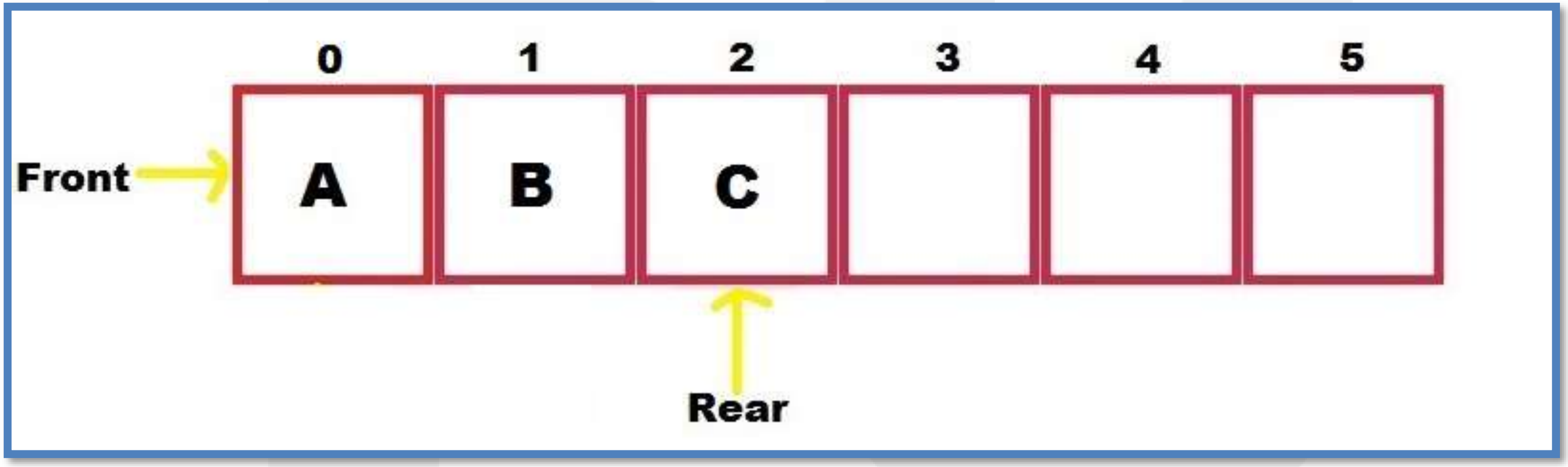
Example of Simple Queue

- Insert B



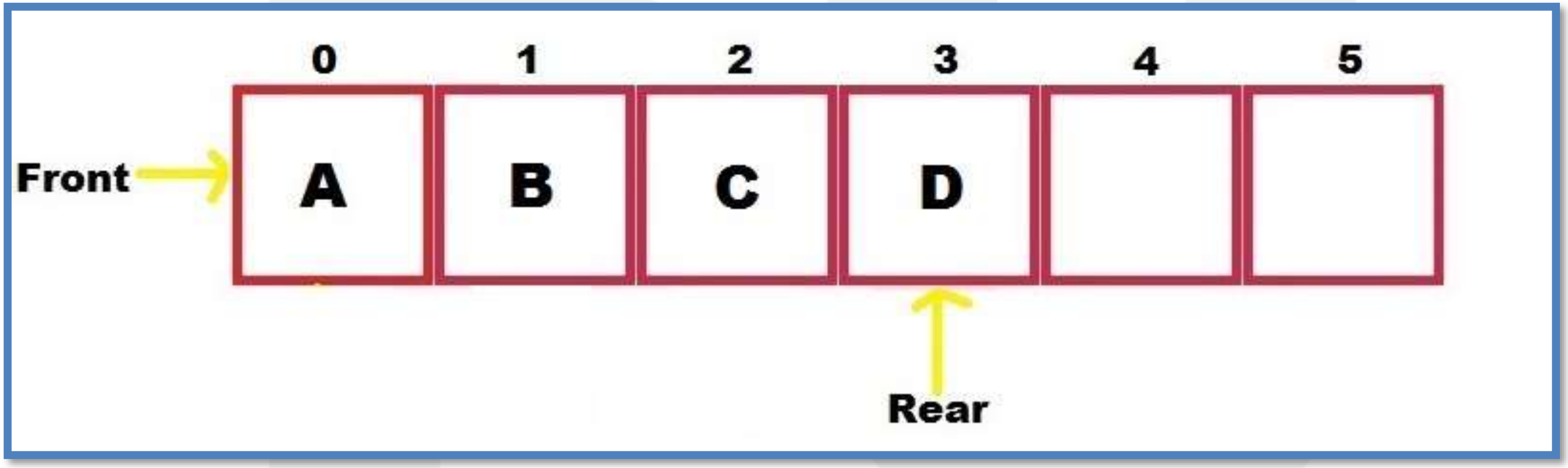
Example of Simple Queue

- Insert C



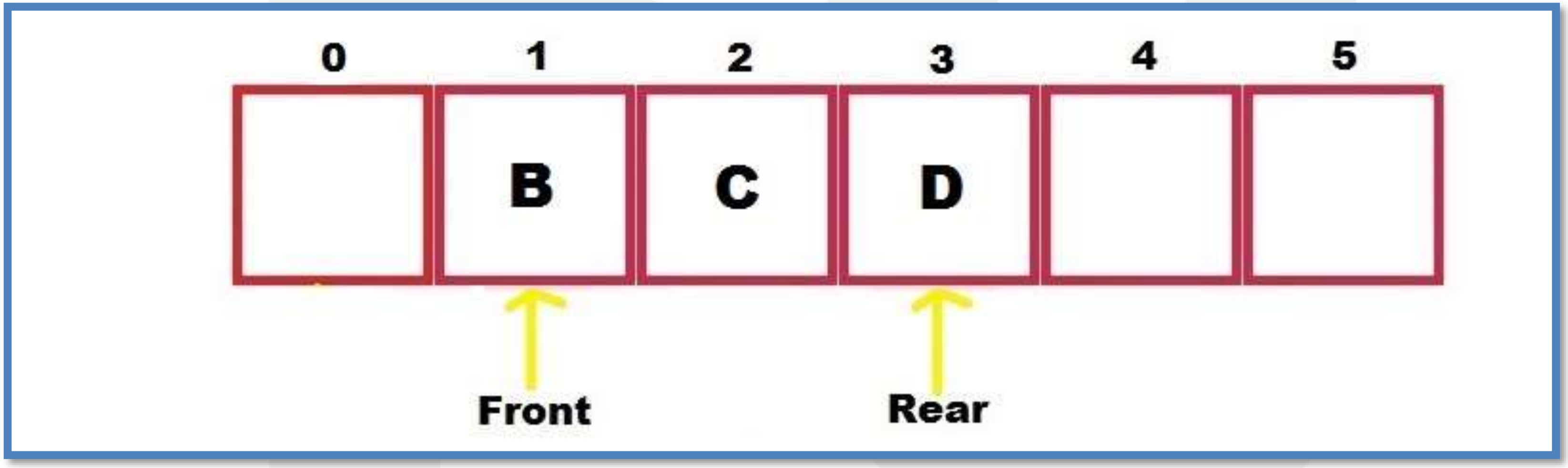
Example of Simple Queue

- Insert D



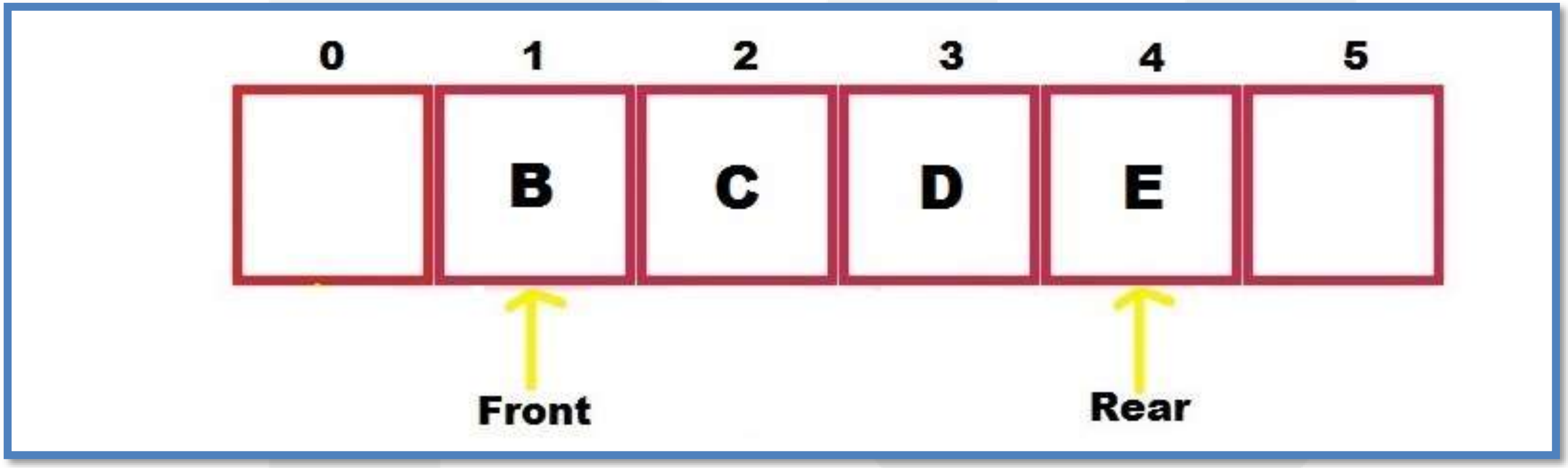
Example of Simple Queue

- Delete A



Example of Simple Queue

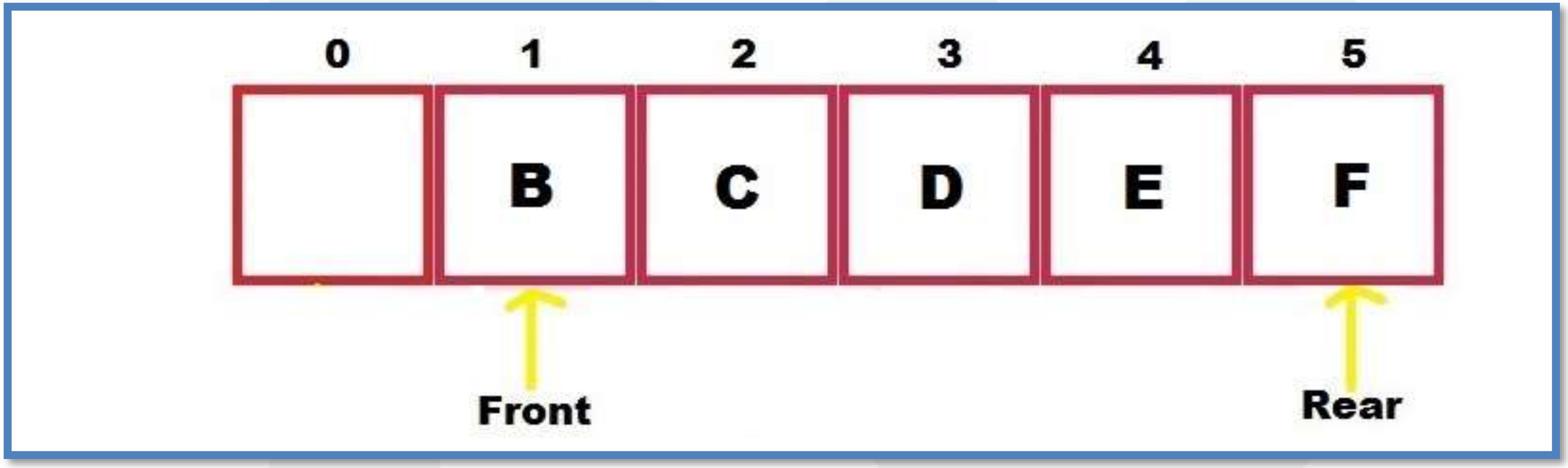
- Insert E



0	1	2	3	4	5
	B	C	D	E	

Example of Simple Queue

- Insert F



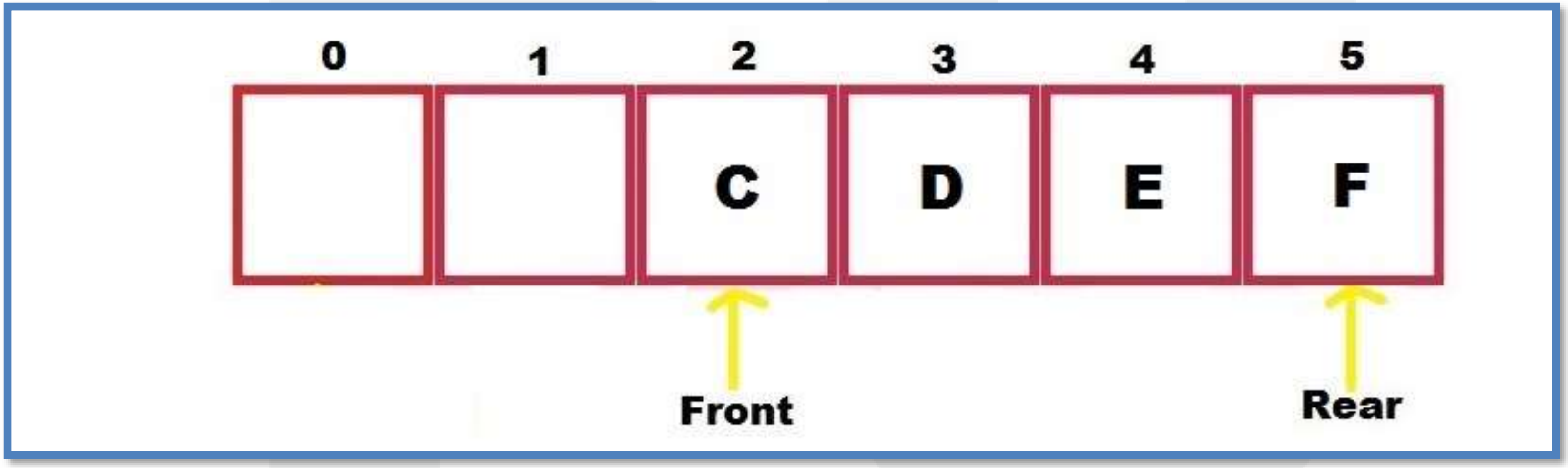
0	1	2	3	4	5
	B	C	D	E	F

Front

Rear

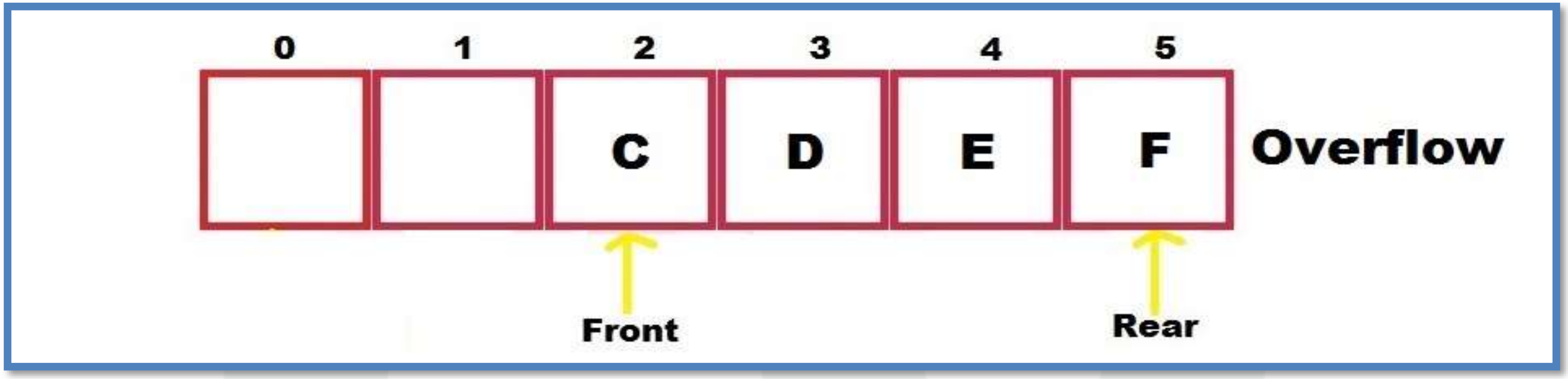
Example of Simple Queue

- Delete B



Example of Simple Queue

- Insert G



•Disadvantage of linear queue is that, in linear queue if the queue is fully occupied and the queue's initial place is empty and if want to insert other character then it is not possible even though the initial cells are empty.

Algorithms

- INSERT(Q, F, R, N, Y)

```
1.  If R>=N then write: overflow and return
2.  R=R+1
3.  Q[R] = Y
4.  If F==-1 then
        F=0 and return
```

- DELETE(Q, F, R)

```
1.  If F==-1 the write: Underflow and return
2.  Y = Q[F]
3.  If F==R then F=-1 and R=-1 else F=F+1
4.  Return [Y]
```

- Here,

Q = Queue

F = Front

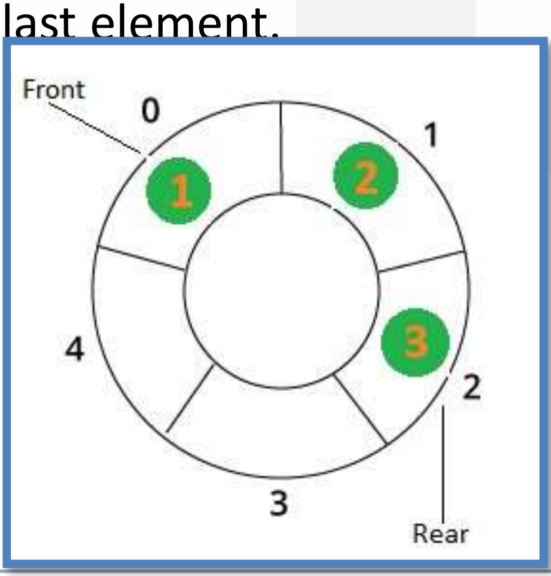
R = Rear

N = Size of Queue

Y = Element or item that can be traced on the queue

Circular Queue

- ➢ In a normal Queue, once queue becomes full, we can not insert the next element even if there is a space in front of queue.
- ➢The solution to this is, whenever the rear end gets to the end of the array, it should be wrapped around to the beginning of the array.
- ➢Now the array can be viewed as a circle where the first position will follow the last element.



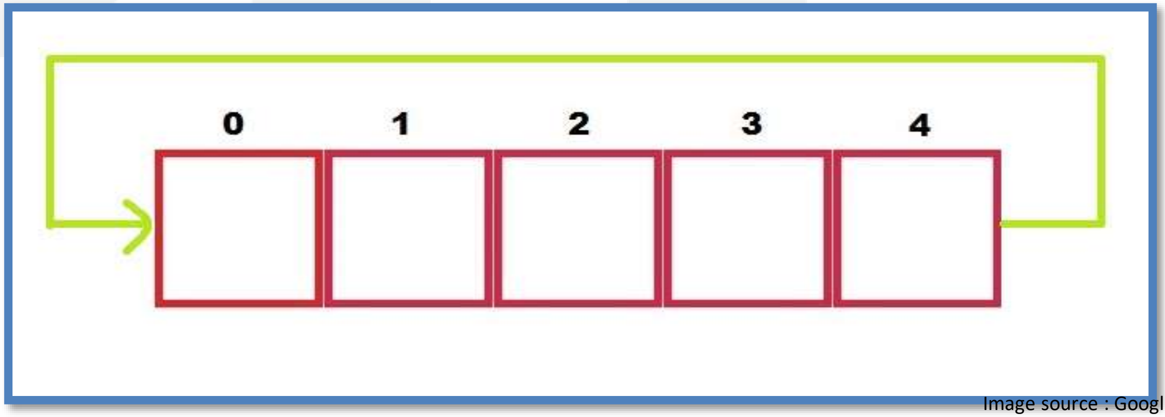
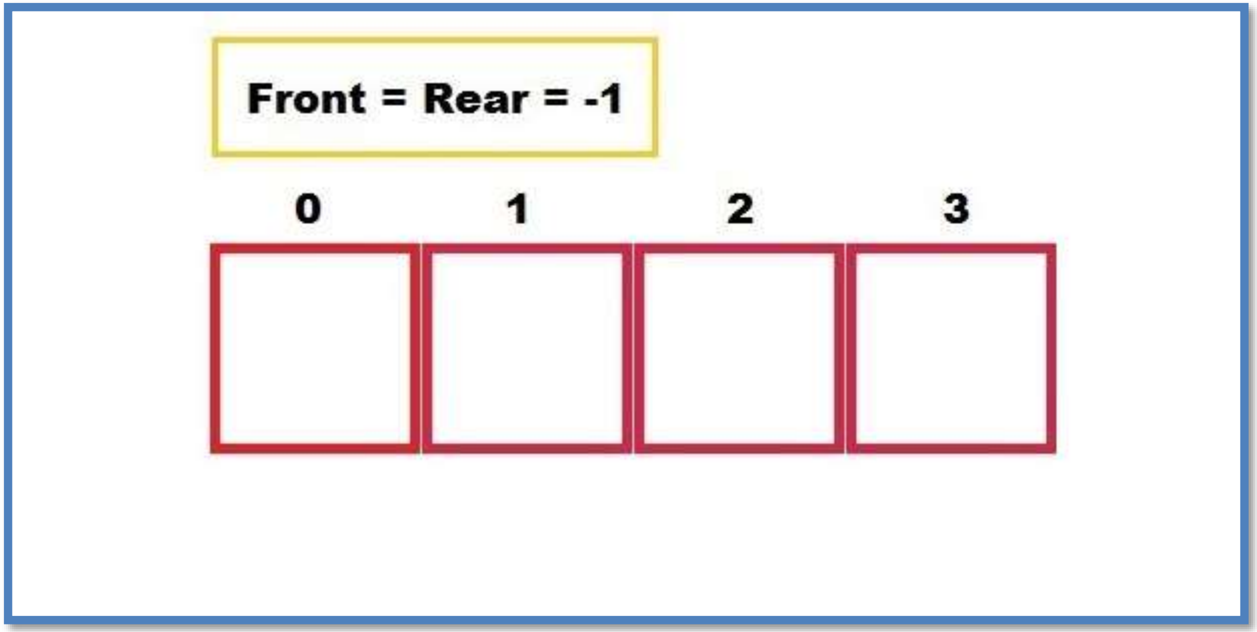


Image source : Google

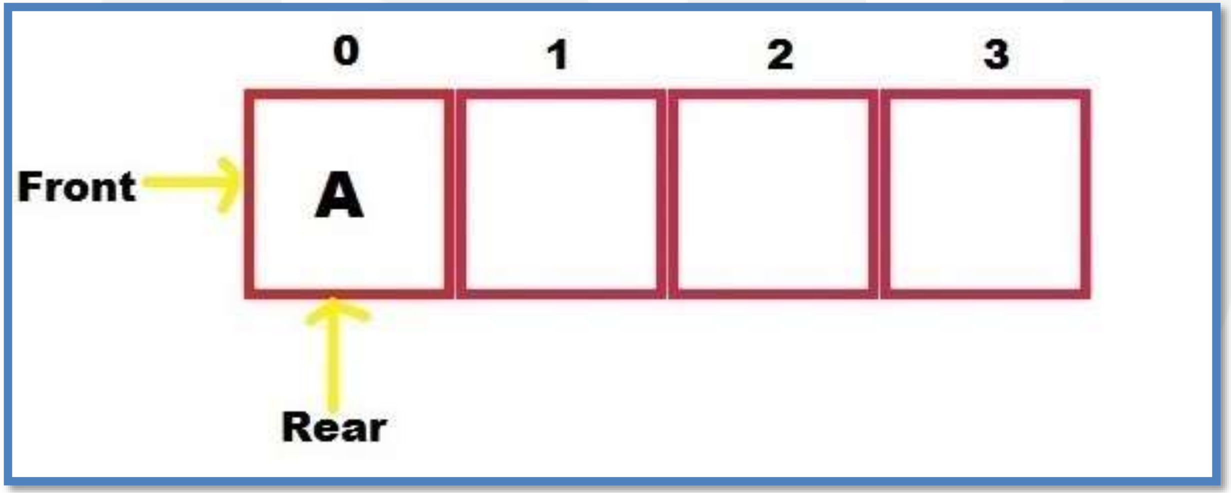
Example of Circular Queue

- Empty



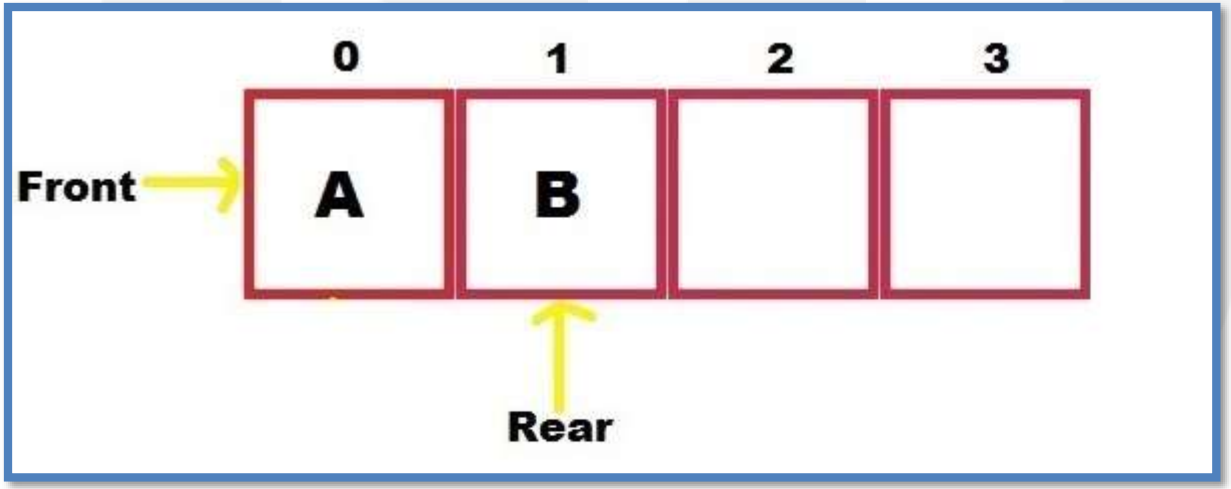
Example of Circular Queue

- Insert A



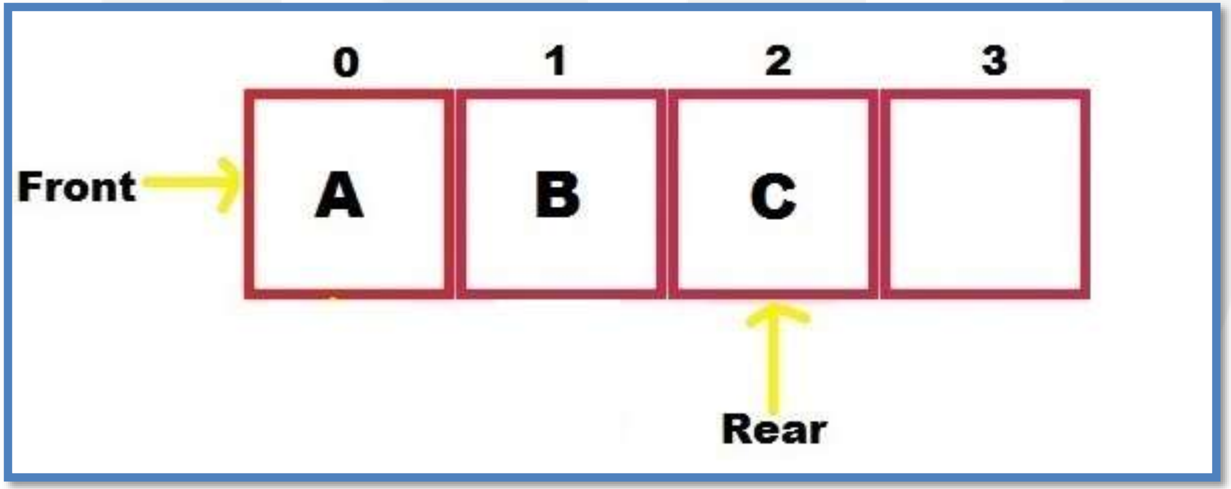
Example of Circular Queue

- Insert B



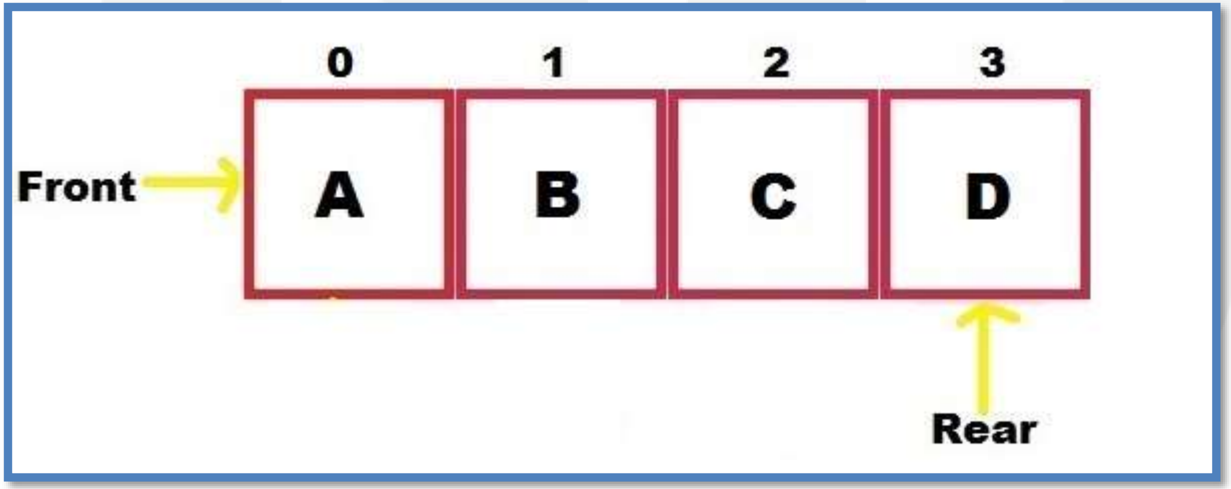
Example of Circular Queue

- Insert C



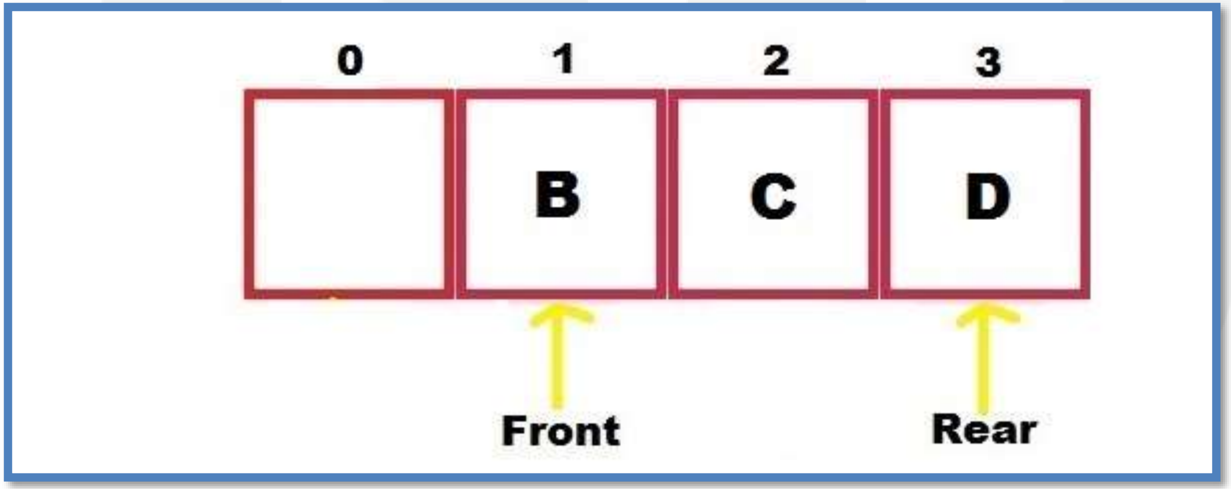
Example of Circular Queue

- Insert D



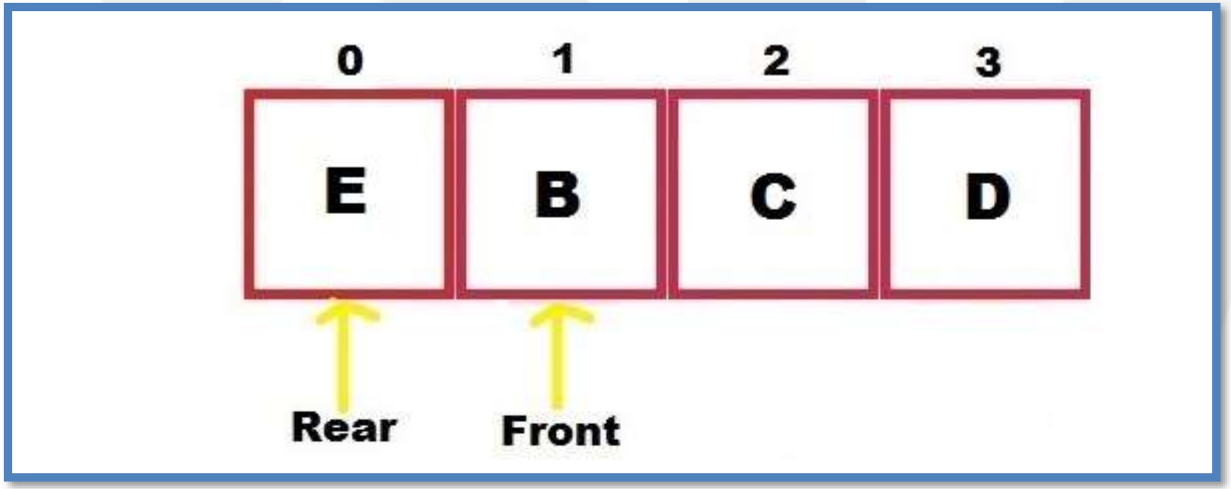
Example of Circular Queue

- Delete A



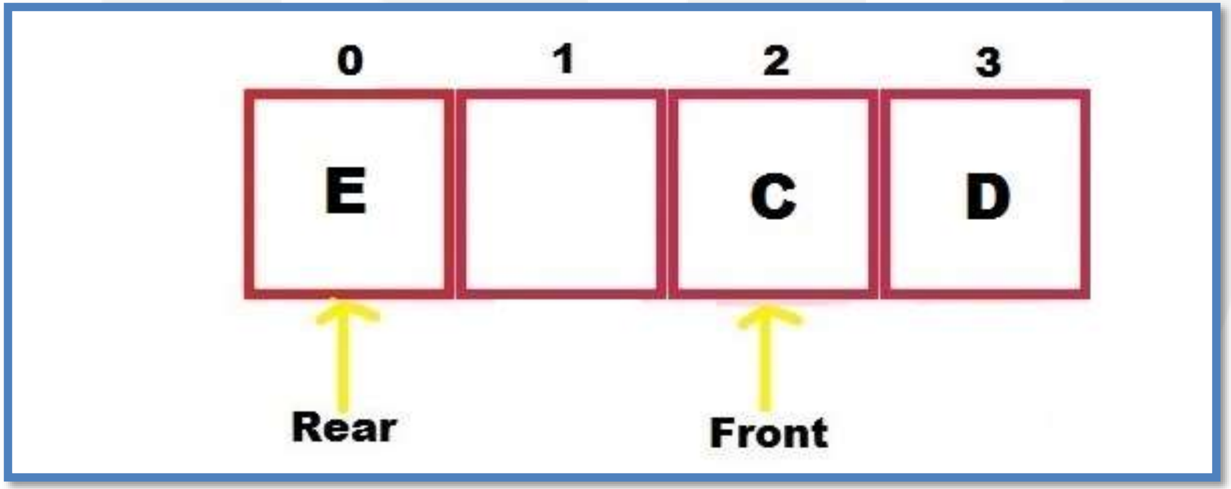
Example of Circular Queue

- Insert E



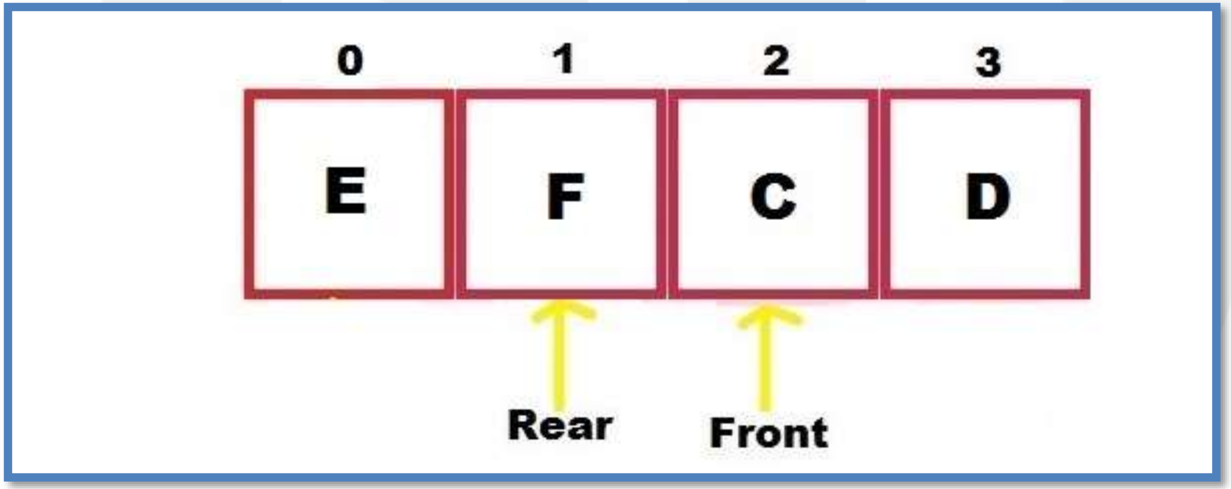
Example of Circular Queue

- Delete B



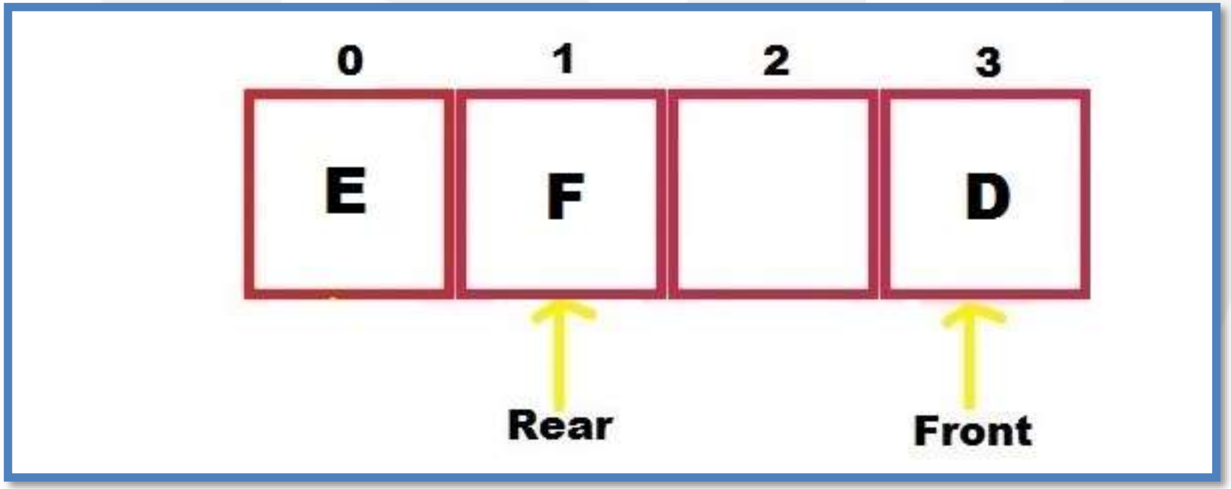
Example of Circular Queue

- Insert F



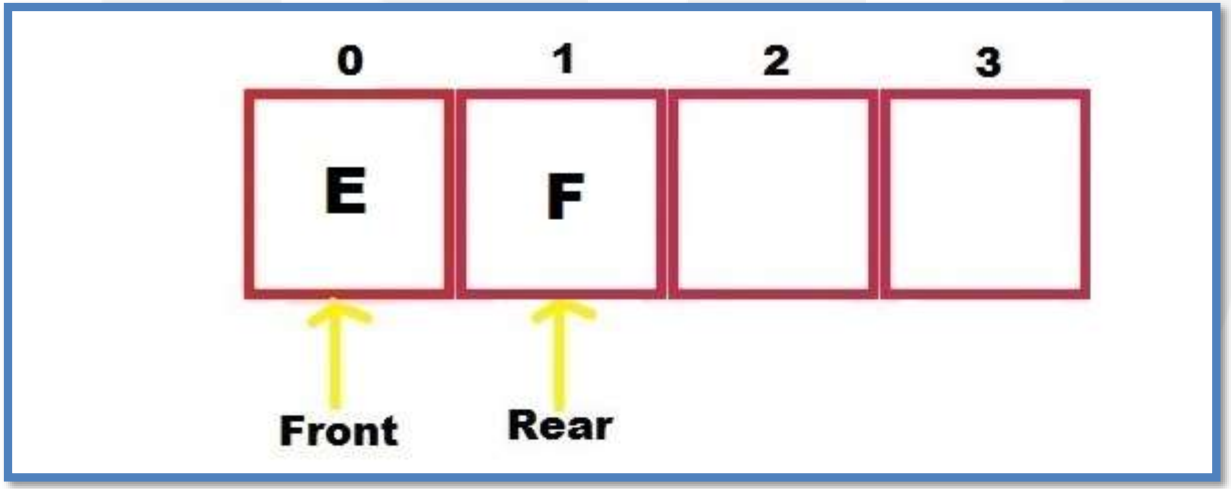
Example of Circular Queue

- Delete C



Example of Circular Queue

- Delete D



Algorithms

➢ CQINSERT(Q, F, R, N, Y)

```
1.  If R==N then R=0 else R=R+1
2.  If R=F-1 then write: Overflow and return
3.  Q[R] = Y
4.  If F==-1 then
        F=0 and return
```

➢ Here,

Q = Queue

F = Front

R = Rear

N = Size of Queue

Y = Element or item that can be traced on the queue

➢CQDELETE(Q, F, R, N)

```
1.  If F==-1 the write: Underflow and return
2.  Y = Q[F]
3.  If F==R then F=-1 and R=-1 and return [Y]
4.  If F==N then F=0 else F=F+1 and return[Y]
```

Dequeue

- ➢ Dequeue is the short notation used for double ended queue.
- ➢The usual practice is: for insertion of element we use one end called rear and for deletion of elements we use front end.
- ➢In dequeue we make use of both the ends for insertion of the elements as well as we can use both the ends for deletion of the elements.
- ➢In simple words, it is a linear list in which elements can be added or removed from either ends but not in the middle.

Structure of Dequeue

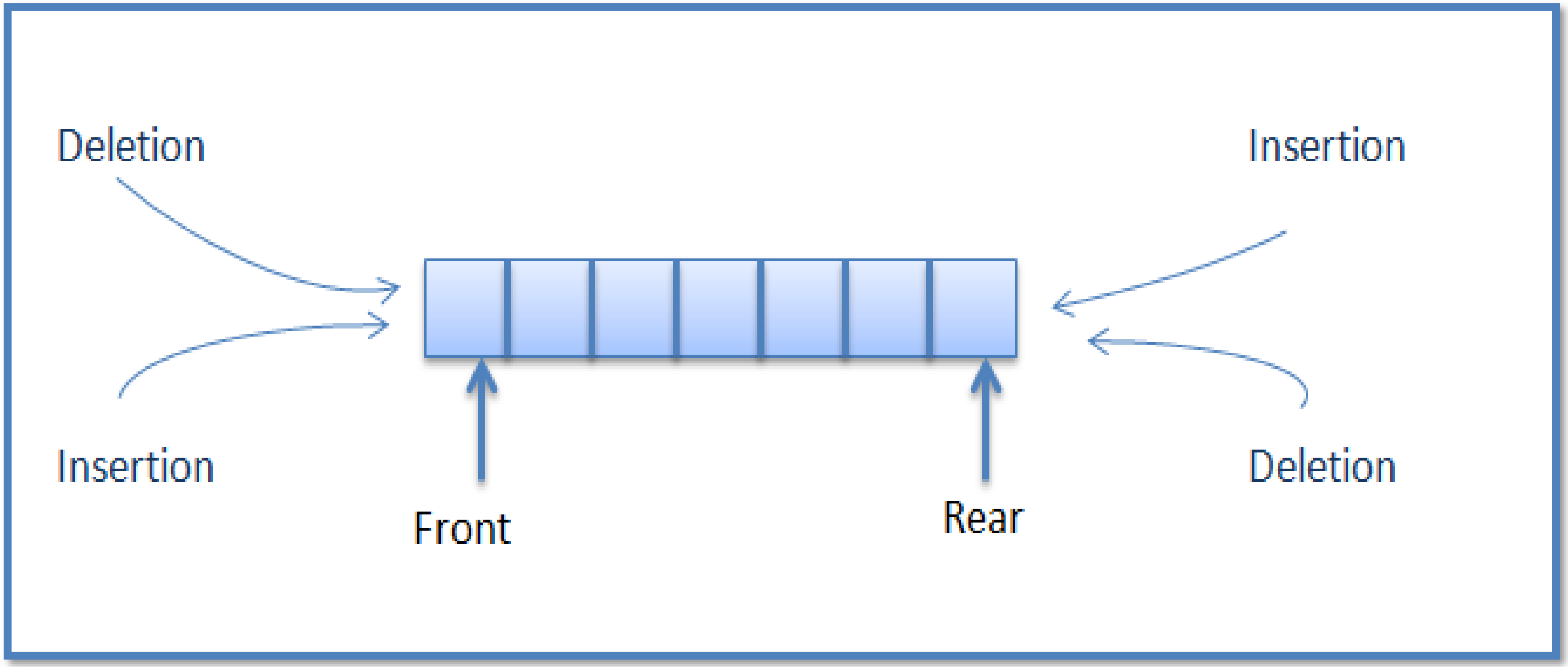
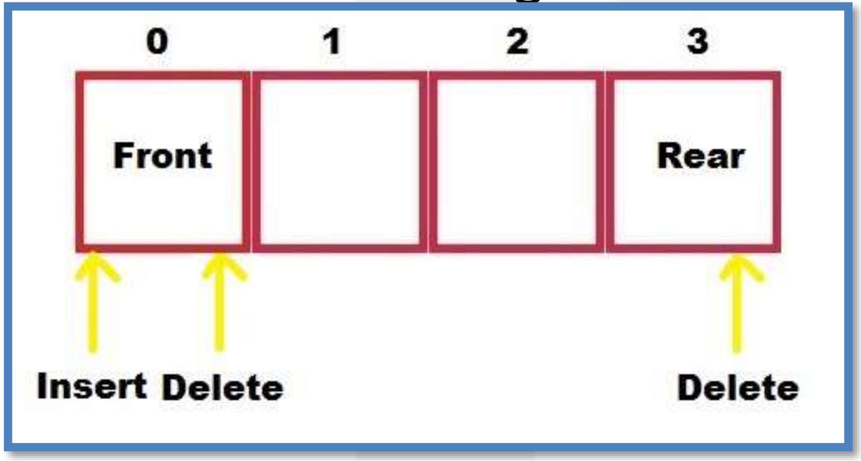


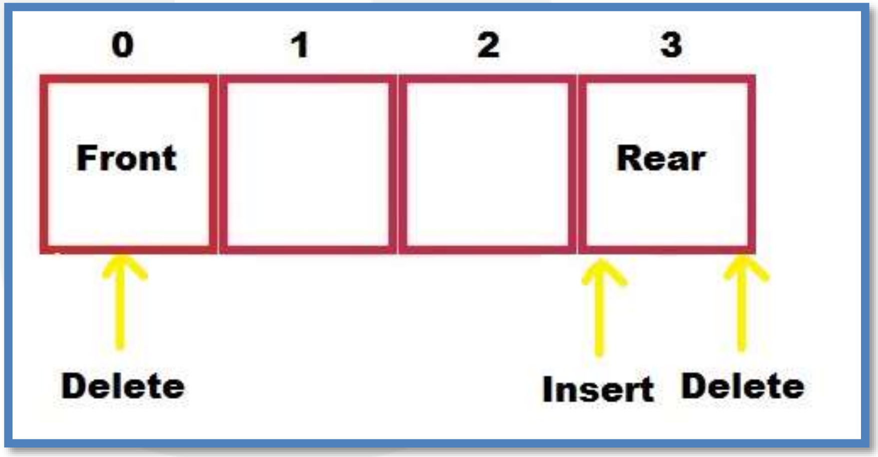
Image source : Google

Classification of Dequeue

➢Input Restricted:

▪When we use only one end for inserting an element and both the ends for deleting an element.

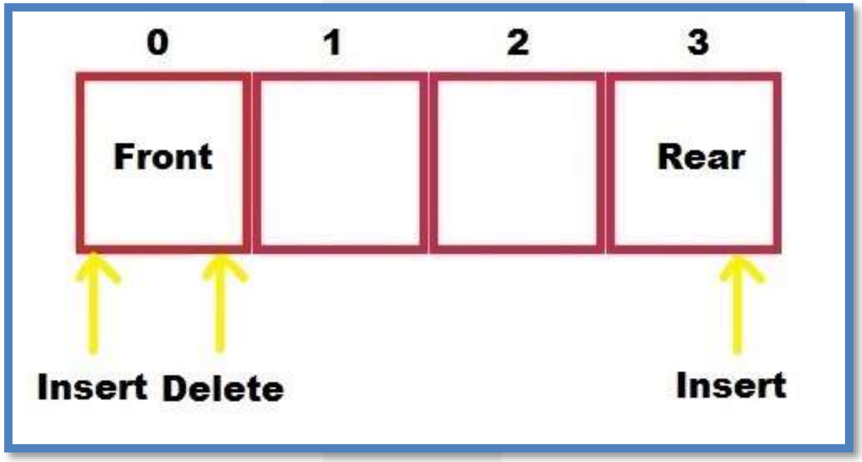


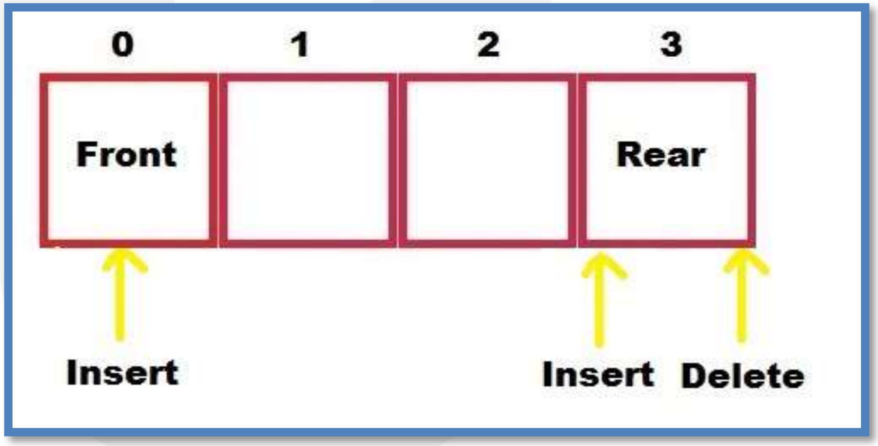


Classification of Dequeue

➢Output Restricted

▪When we use only one end for deleting an element and both ends for insertion of an element.





Methods to implement Dequeue

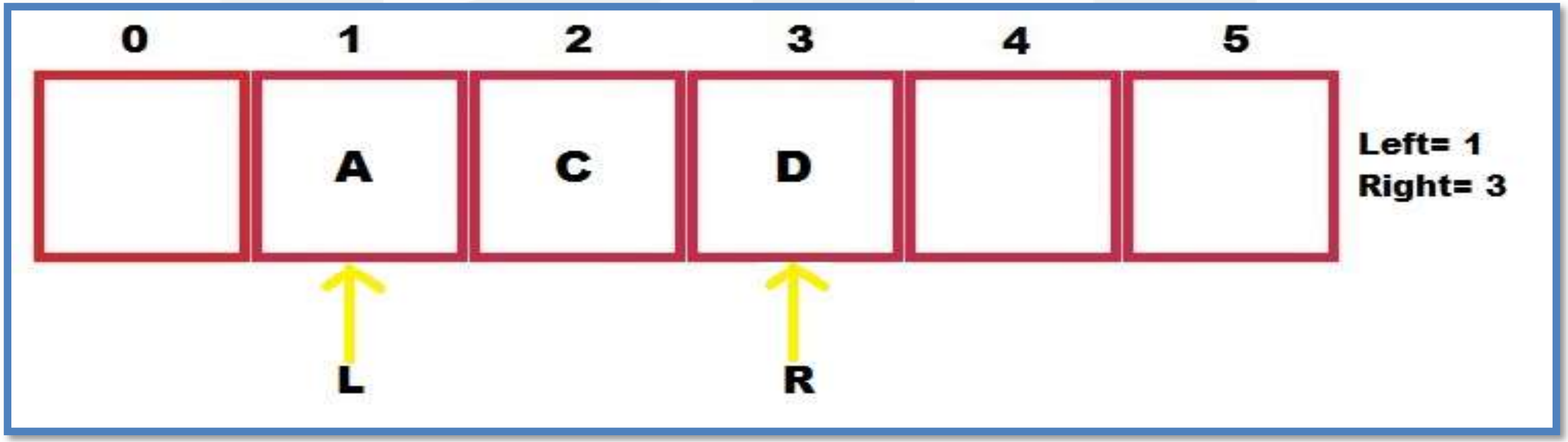
- Using circular array
- Using singly linked list
- Using doubly linked list
- Using singly circular linked list
- Using doubly circular linked list

Operations on dequeue

- initialize():Make the empty queue.
- empty(): To check whether queue is empty or not.
- full(): To check whether queue is full or not.
- enqueueF(): To insert an element to the front end of the queue.
- enqueueR(): To insert an element to the rear end of the queue.
- dequeueF(): Delete the front element.
- dequeueR(): Delete the rear element.
- print(): Print elements of queue.

Example of Dequeue

- ➢ Consider the following dequeue of characters where dequeue is a circular array which is allocated 6 memory cells.
- ➢Left=1 and Right =3.
- ➢Dequeue: _,A,C,D,_,_.

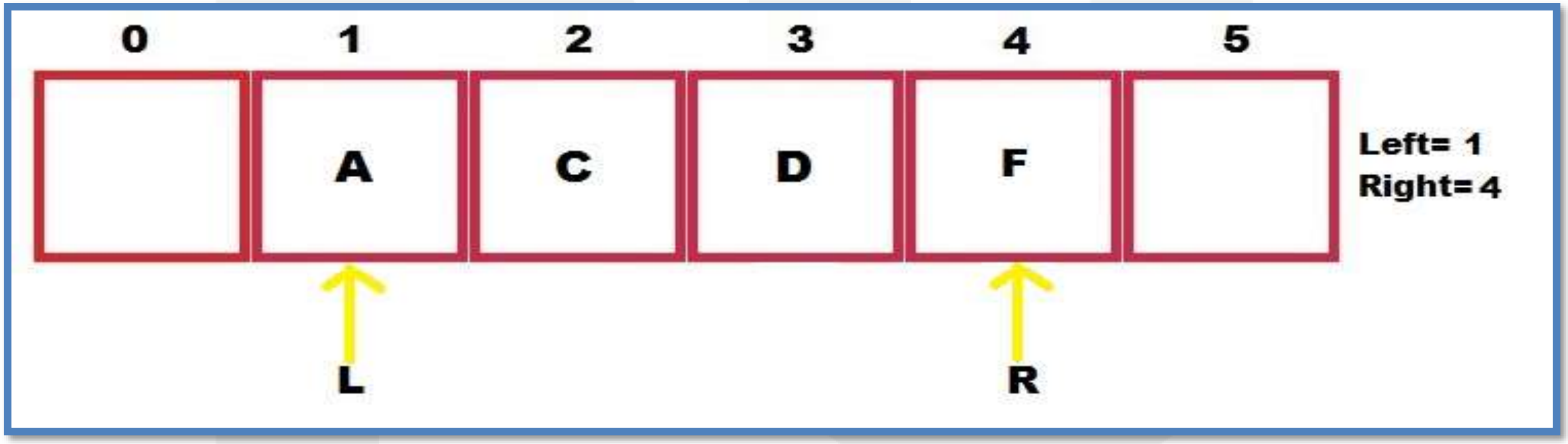


0	1	2	3	4	5
	A	C	D		
	L		R		

Left= 1
Right= 3

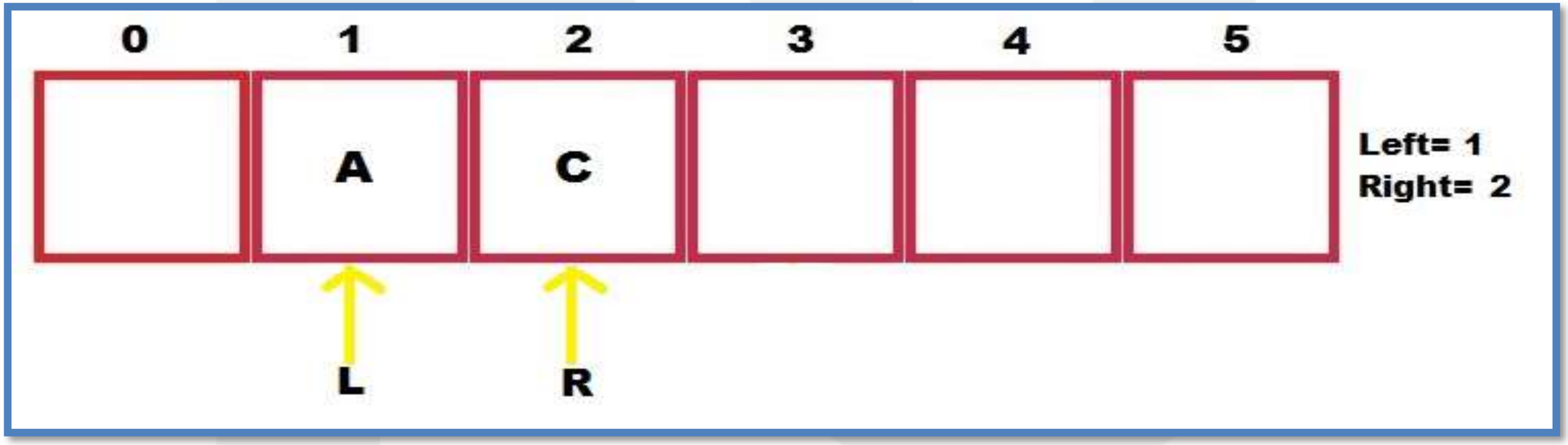
Example of Dequeue

- F is added to right of dequeue



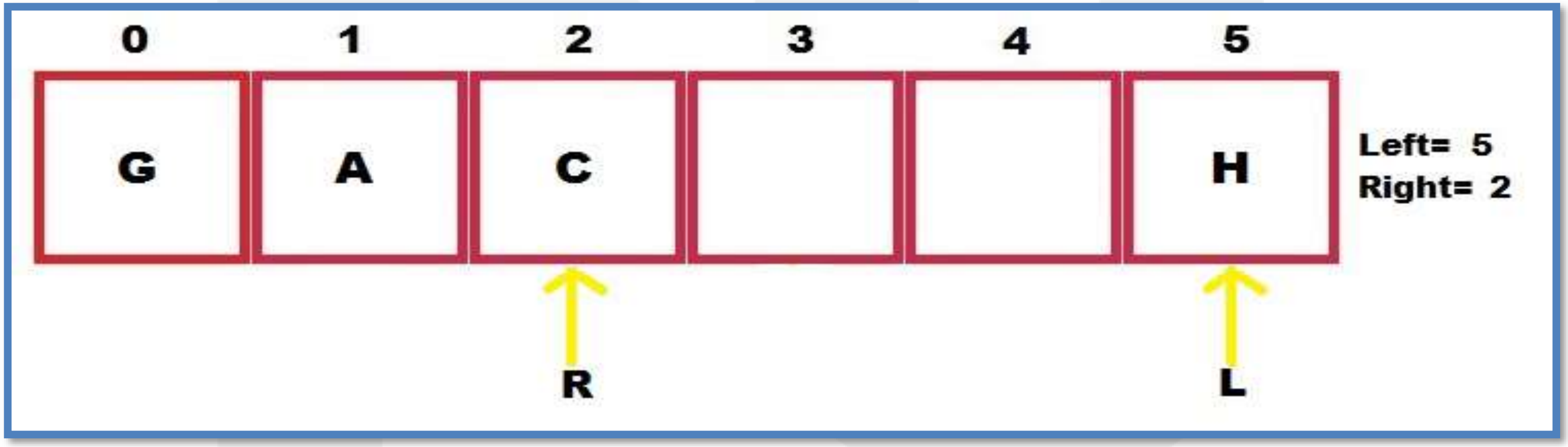
Example of Dequeue

- ➢2 letters are deleted from right.



Example of Dequeue

- G, H are added to left of dequeue.

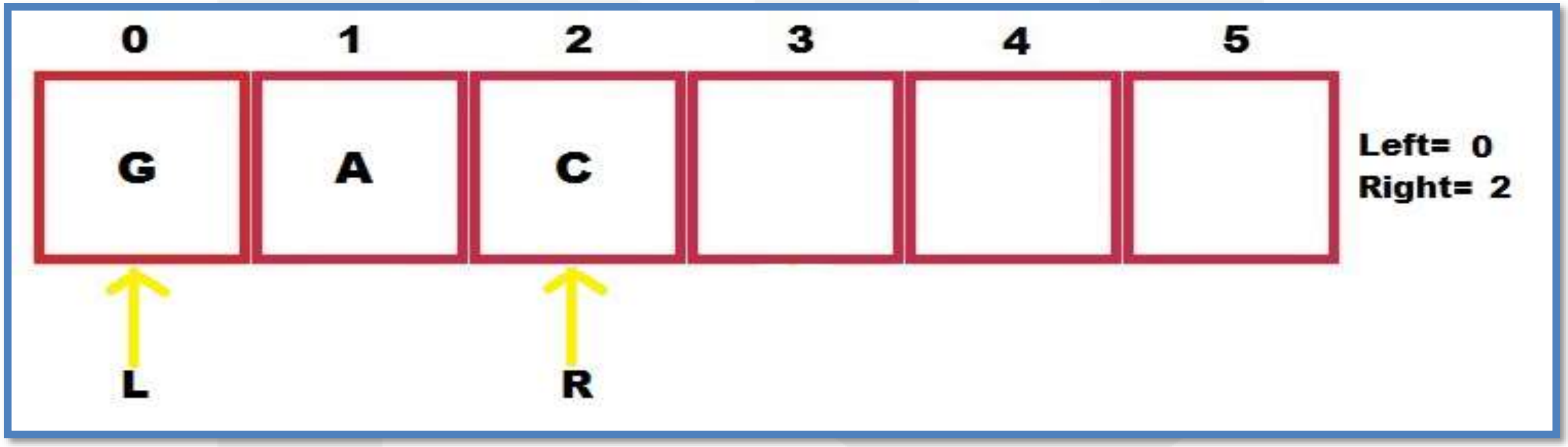


0	1	2	3	4	5
G	A	C			H

Left= 5
Right= 2

Example of Dequeue

- 1 letter is deleted from left

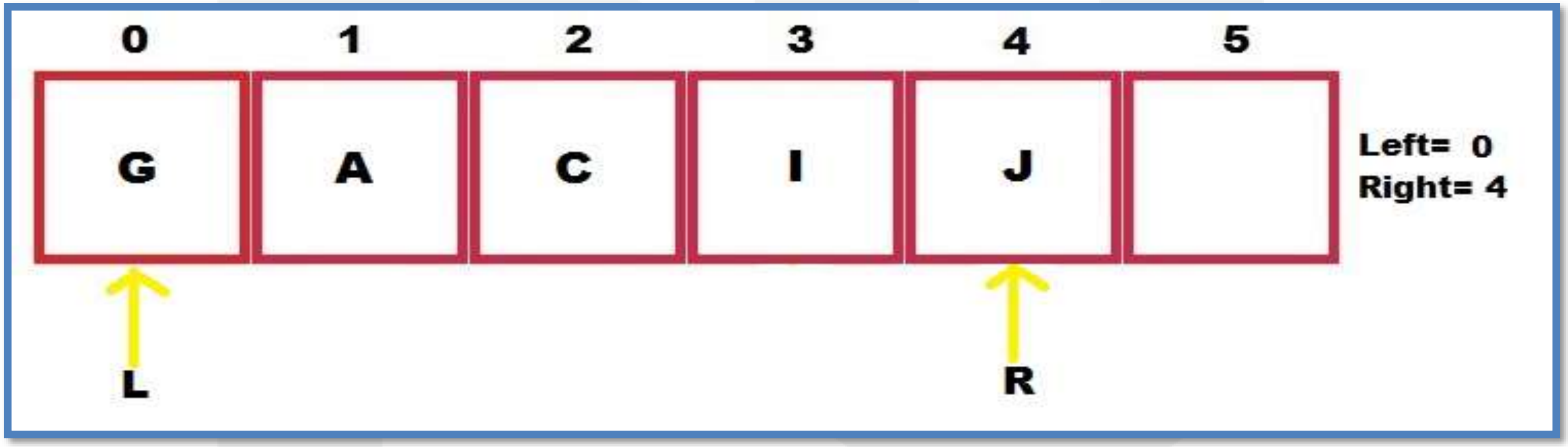


0	1	2	3	4	5
G	A	C			

Left= 0
Right= 2

Example of Dequeue

- ➢I, J are added to right of dequeue

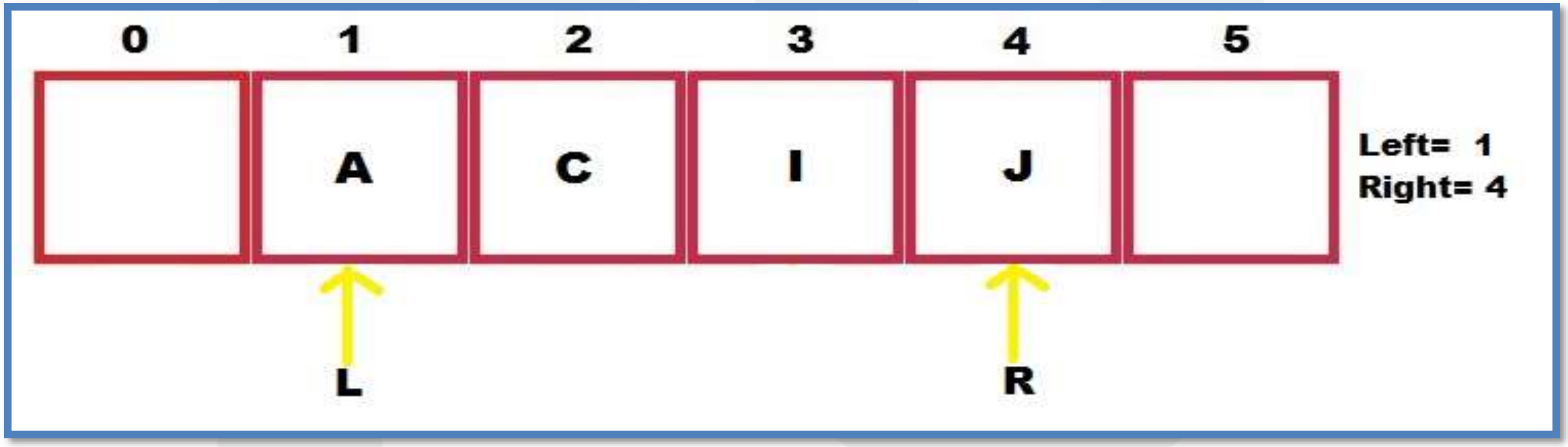


0	1	2	3	4	5
G	A	C	I	J	

Left= 0
Right= 4

Example of Dequeue

- 1 letter is deleted from left



Priority Queue

- It is a collection of elements such that each element has been assigned a priority and the order in which elements are processed and deleted follows following rules:
- An element with higher priority is processed before any element with lower priority.
- Two elements with same priority are processed on FCFS (First Come First Serve) basis.

Operations on priority queue

- initialize(): Make the empty queue.
- full(): Check if the queue is full or not.
- empty(): Check if the queue is empty or not.
- enqueue(): Insert an element as per its priority.
- dequeue(): Delete the element in front(as front element has the highest priority).
- print(): Print queue elements.

Types/Representation of priority queue

1) One way list representation of a priority:

- Each node in a list will contain 3 items of an information: INFO(), PRIORITY_NO() and a link.
- A node X precedes node Y in the list where:

  1. X has higher priority than Y.

  2. When both have same priority but X was added in the list before Y.

Algorithm

INSERT

- It adds an item with priority number ‘n’ to a priority queue which is maintained in memory as one-way list.
- Traverse one- way list until you find a node X whose priority number exceeds ‘n’.
- Insert ITEM in front of node X.
- If no such element is found, insert ITEM as the last element of the list.

Algorithm

DELETE

- ➢ It deletes and processes the first element in a priority queue which appears in memory as one-way list.
- ▪ Set ITEM=INFO[START]
- ▪ Delete first node from the list
- ▪ Process ITEM
- ▪ Exit

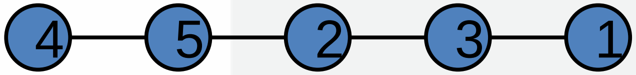
Types/Representation of priority queue

2) Array Representation of Priority Queue:

- Another way to maintain priority queue is to use a separate queue for each level of priority.
- Each such queue will appear in its own circular array and must have its own pair of pointers(rear and front).
- In each queue is allocated same amount of space, a 2D array of queue can be used instead of linear arrays.

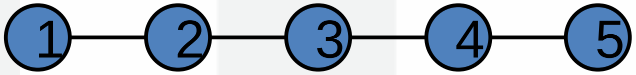
Implementation of priority queue

- Implementation with an unsorted list



- Performance:
  - insert takes ***O***(1) time since we can insert the item at the beginning or end of the sequence
  - remove take ***O***(***n***) time since we have to traverse the entire sequence to find the smallest key

- Implementation with a sorted list



- Performance:
  - insert takes ***O***(***n***) time since we have to find the place where to insert the item
  - remove and take ***O***(1) time, since the smallest key is at the beginning

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in