Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Data Structure

- organising data inside memory for efficient processing along with operations like add, delete, search, etc which can be performed on data.
- eg stack - push/pop/peek

- data structures are used to achieve
  - Abstraction
    Abstract Data Type (ADT)
  - Reusability
  - Efficiency
    time & space required to execute

Types of data structures

Linear data structures
(basic)

- data is organised sequentially/ linearly

- data can be accessed sequentially

e.g. array, structure/class, stack, queue, linked list

Non linear data structures
(Advanced)

- data is organised in multiple levels (hierarchy)

- data can not be accessed sequentially

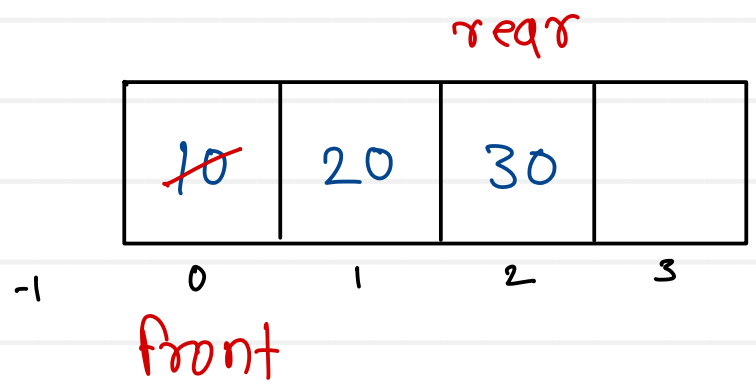
e.g. tree, graph, heap

Hash table

Linear queue

- linear data structure which has two ends - front and rear
- Data is inserted from rear end and removed from front end
- Queue works on the principle of “First In First Out” / “FIFO”

capacity = 4



Operations:

1. Add / push / insert / enqueue:
   a. reposition rear (inc)
   b. add value on rear index

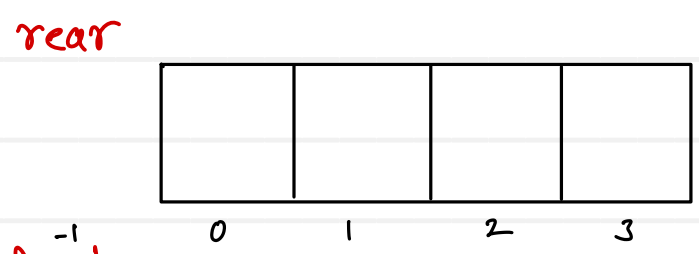
offer()

2. Delete / pop / remove / dequeue:
   a. reposition front (inc)

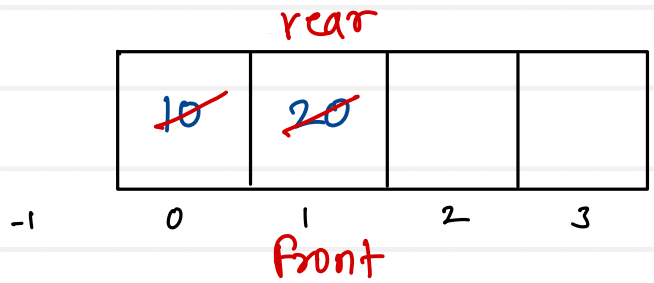
poll()

3. Peek:
   a. read data / value from front end
      ↓
      (front+1)

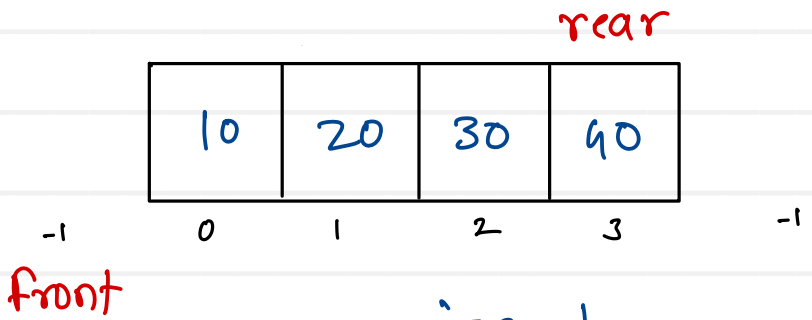
Linear queue - Conditions



Empty

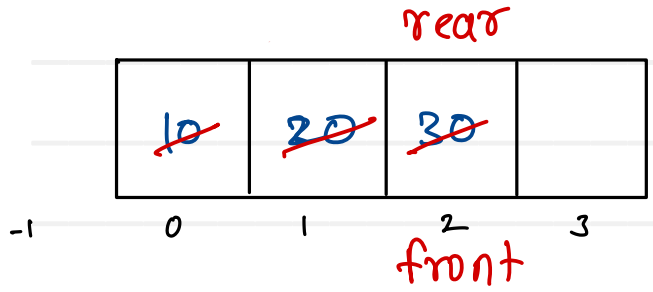


front == rear



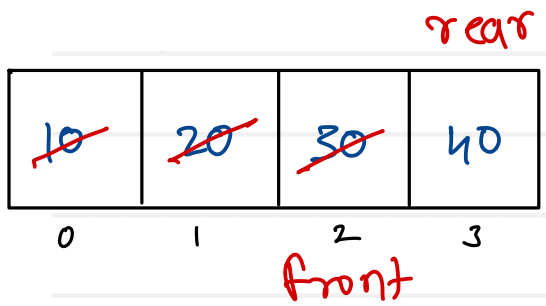
Full

rear == size - 1




```
pop():
  if(front==rear)
    front=rear=-1;
```

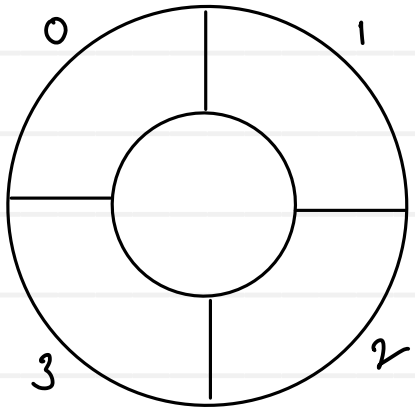
Disadvantage :



- if rear is on last index & few initial locations are vaccant, still we can not use them, this leads to poor memory utilization

Circular queue

capacity = 4



rear = (rear+1) % size

front = (front+1) % size

rear = front = -1

= (-1+1) % 4 = 0

= (0+1) % 4 = 1

= (1+1) % 4 = 2

= (2+1) % 4 = 3

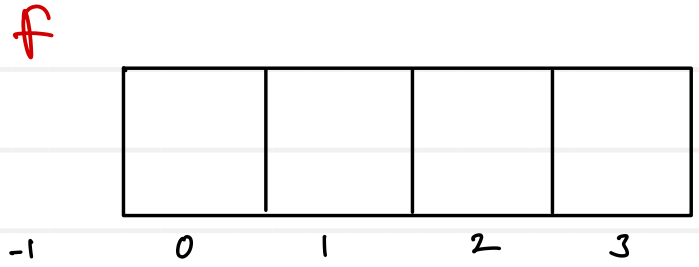
= (3+1) % 4 = 0

Operations :

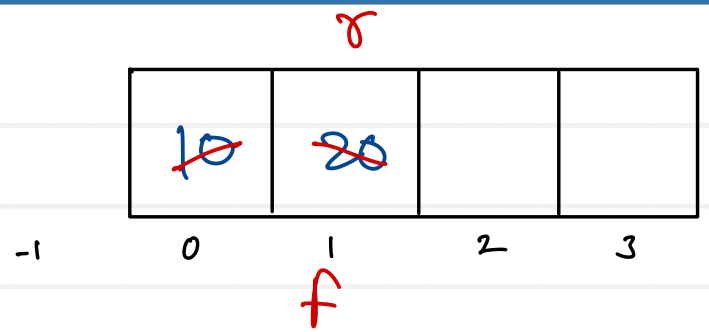
1. Push / enqueue / add / insert :
   a. reposition rear (inc)
   b. add value at rear index
2. Pop / dequeue / delete / remove :
   a. reposition front (inc)
3. Peek :
   a. read / return data of front+1 index

Circular queue - Conditions

Empty

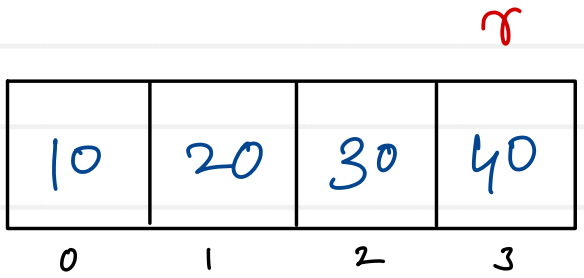


front == rear && rear == -1

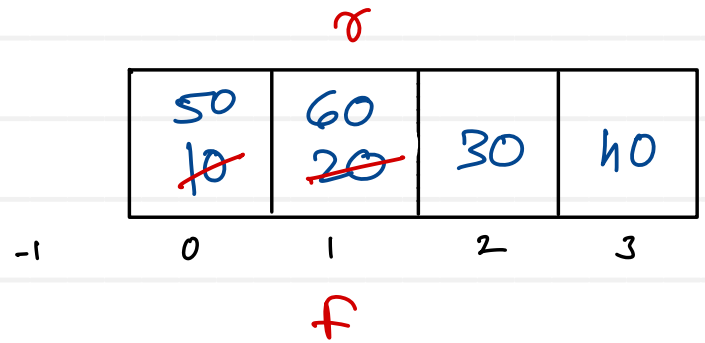



```
pop():
    front = (front+1) % size;
    if(front == rear)
        front = rear = -1;
```

Full



front == -1 && rear == size-1



front == rear && rear != -1

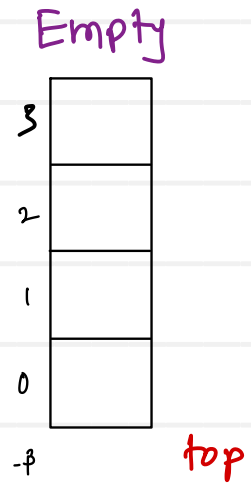
Stack

- Stack is a linear data structure which has only one end - top
- Data is inserted and removed from top end only.
- Stack works on principle of “Last In First Out” / “LIFO”

capacity = 4

(top always points to last inserted data)

1. push :
   a. reposition top (inc)
   b. add value at top index
2. pop :
   a. reposition top (dec)
3. peek :
   a. read data of top index



top == -1

top == size-1

Ascending stack

top = -1

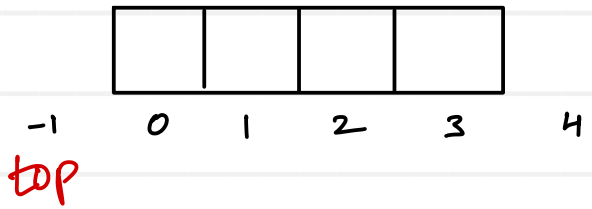
push : top++
arr[top] = value

pop : top--

peek : arr[top]

Empty : top == -1

Full : top == size-1



Descending stack

top = size

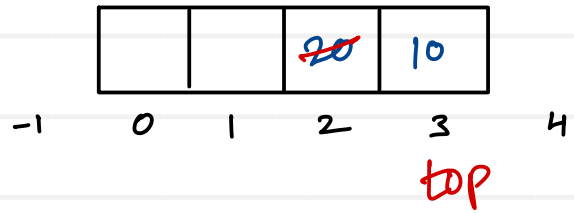
push : top--
arr[top] = value

pop : top++

peek : arr[top]

Empty : top == size

Full : top == 0



Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com