Queue Data Structure (LL implementation) & Applications

Operations → enqueue(x) size()
dequeue() isEmpty()
getFront()
getRear()

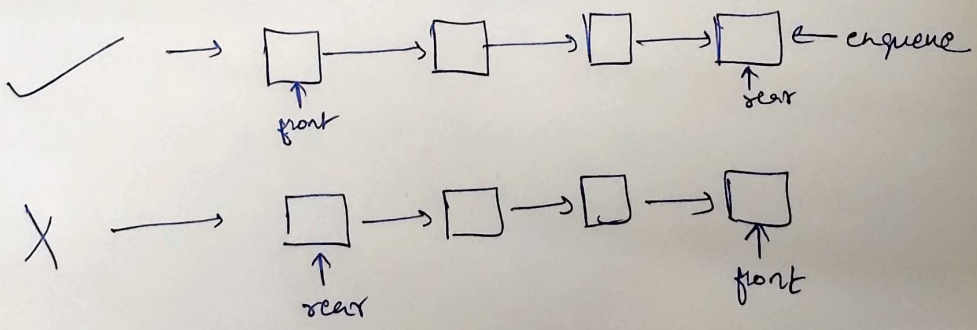
Application

→ 1) Single source multiple consumers.
2) In OS and computer Architecture - (FCFS, Spooling, etc).
3) In Routers / Switches and Mail in Computer Networks.
4) Synchorizing devices.

LL implementation

→ Array implementation discussed with Stacks - Basic.

→ for O(1) for both enqueue and dequeue will shoulds include tail pointer.



Code

```
class Node{
    int data;
    Node next;
    Node(int x){ data = x; next = null; }
}
```

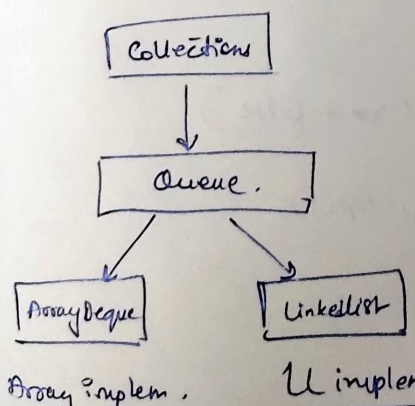
```
class Queue
{   Node front, rear;
    Queue() { front = rear = null; }

    void enque(int x)
    {   Node temp = new Node(x);
        if (front == null)
        { front = rear = temp; return; }
        rear.next = temp;
        rear = temp;
    }

    void deque()
    {   if(front == null) return;
        front = front.next;
        if(front == null) { rear = null; }
    }
```

size → inside enque → size++ ;
dequeue → size-- ;

In Java



Array implem.

LL implem.

* Stack is class
* Queue is interface.

Queue<Integer> q1 = new ArrayDeque<Integer>();

offer → enqueue
poll → dequeue.

peek
size
isEmpty.

Alt with exception are add remove element.

enq & deque in both O(1).

Stack using queue.

Stack can be implemented using 2 queues. (or (1queue + Recursion)

3 q_1 : { 10, 5, 15 } ⟶ q_1 : { 20, 10, 5, 15 }
q_2 : { } q_2 : { }

We can see 20 is added in front instead of rear like a stack & will be called first with pop().

How → Step 1 → q_1 : { }
q_2 : { 10, 5, 15 }

Step 2 → q_1 : { 20 }
q_2 : { 15, 5, 10 }

Step 3 → q_1 : { 20, 10, 5, 15 }
q_2 : { }

```
Struct Stack {
        queue<int> q1, q2;
        int top() { return q1.front(); }
        int size() { return q1.size(); }
        int pop() { return q1.front(); }

        int push(int x) {
            while (q1.empty() == false)
            {
                q2.push(q1.top());
                q1.pop();
            }
            q1.push(x);
            while (q2.empty == false)
            {
                q1.push(q2.top());
                q2.pop();
            }
        }
```

Reversing a Queue

→ Stack used to reverse.

* Iterative.

```
Void reverse (Queue <Integer> q)
{  Stack <int> s;
   while (q.empty() == false)
   {  s.push(q.top());
      q.top();
   }
   while (s.empty() == false
   {  q.push(s.top());
      s.pop();
   }
```

top → remove in Java. then q.top not needed in next line.

s.push(q.remove); is enough

Recursion

* Void Reverse (Queue <Integer> s)

```
{  if(q.empty()){return;}
   int x = s.top();
   q.pop();
   reverse(q);
   q.push(x);
}
```

x = q.pop() → only one line in Java.