Queues

Do something today that your future self will thank you for.

Good Evening

Today's content

01. Introduction to Queues
02. Implementation of Queues
    - → using Arrays
    - → using LinkedList
    - → using stacks
03. Generate K^{+} no. using 1 & 2
04. Max of every window (Deque)

Sliding Window Maximum (VVVI)

Queue → linear data structure where elements get inserted from one end & get removed from the other end.

→ First In First Out (principle)

* Some real World Examples

01. Message queue
02. Customer service
03. Ticket counter
04. Printer
05 Toll plaza

* Common operation in Queue → TC: O(1)

01. Enqueue → Add an element inside the queue. (q.add(x))
02. Deque() → Remove the first element & return it. (q.remove())
03. Peek() → peek() operation allows you to look at the

first element in queue. (q.peek())

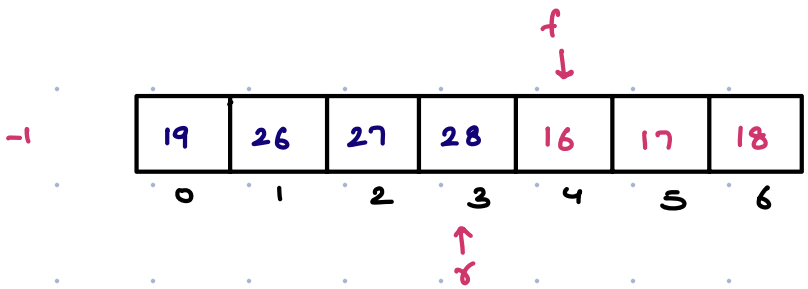
04. size() → return the no. of element present at queue. (q.size())

* Implementation of Queue

01. Array

f = -1 → idx of last ele which just got removed

r = -1 → idx of last ele which just got inserted



Add(5)	remove()	Add(16)	Add(26)	peek()
Add(6)	remove()	Add(17)	Add(27) ←	peek()
Add(7)	remove()	Add(18)	Add(28) ←	
Add(9)	remove();	Add(19)	remove() → 16	

if (f == r) → Queue is empty

└→ Queue is full

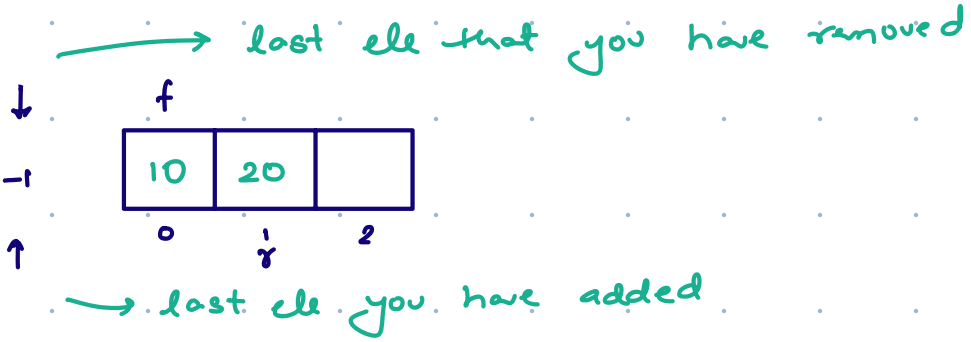
```
void add(x){
    if (sz==n) return;
    r= (r+1) %n
    A[r] = x;
    sz++;
}
```

```
int size(){
    return sz;
}
```

```
int remove(){
    if (sz==0) return -1;
    f = (f+1) % n
    int val = A[f]
    sz--;
    return val;
}
```

```
int peek(){
    if (sz==0) return -1;
    int idx = (f+1) % N
    return A[idx];
}
```

```
boolean isEmpty(){
    if (sz==0) return true;
    else return false;
}
```



Add(10)

peek() ⟶ 10

Add(20)

remove()

* Implement Queue using LinkedList

↳ FIFO

Add (5) ✓

Add (6) ✓

Add (7) ✓

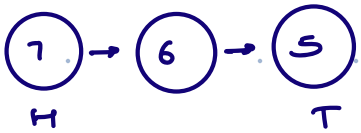
remove ()

remove ()

Add(8)

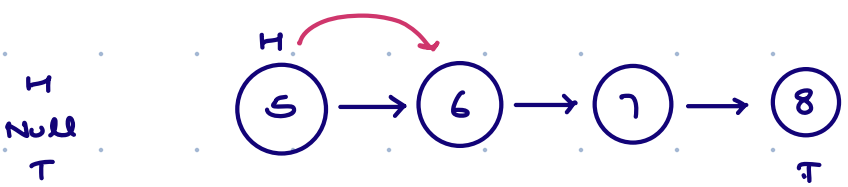
✗

* Insert at head → O(1)
* Remove at tail → O(n)



* Insert at Tail → O(1)

Remove at Head → O(1)



Tail.next = new Node (8)

Tail = Tail.next

What will be the state of the queue after these operations enqueue(3), enqueue(7), enqueue(12), dqueue(), dqueue(), enqueue(8), enqueue(3)

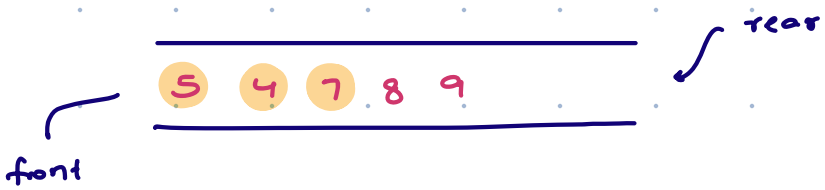
3 7 12 8 3 ✓

Output = 12 8 3

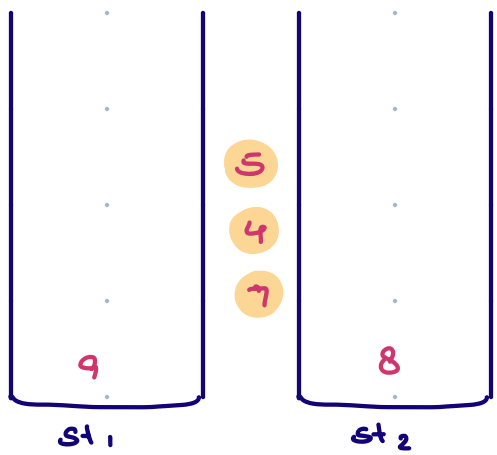
What will be the state of the queue after these operations enqueue(4), dqueue(), enqueue(9), enqueue(3), enqueue(7), enqueue(11), enqueue(20), dqueue()

Output = 3 7 11 20

* Implementation using Stacks



01. Add(5) → st.push(5)
02. Add(4) → st.push(4)
03. Add(7) →
04. Add(8) →
05. remove() ⇐
06. remove()
07. Add(9)
08. remove()



Add(x) → st1.push(x) → TC: O(1)

Remove() → if (st2.size() == 0) → TC: O(1)

Yes → Move all ele from st1 to st2 → st2.pop();

No → st2.pop();

size() → st1.size() + st2.size();

01. Add(5)

02. Add(4)

03. Add(7)

04. Add(8)

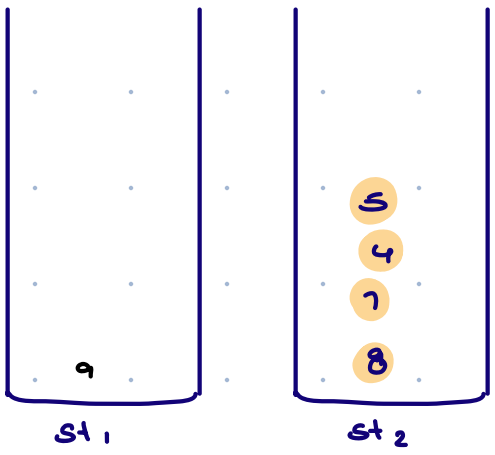
05 remove() ✔

06. Add(9)

07 remove()

08 remove()

09 remove()



1st rem = Move all ele from st1 to st2 & st2.pop()

8 ops + 1 ops

2nd rem = st2.pop() = 1 ops

3rd rem = st2.pop() = 1 ops

4th rem = st2.pop() = 1 ops

4 rem = 12 ops

1 rem = $\frac{12}{4}$ = 3 ops ≈ O(1)

10 : 33 pm → 10 : 43 pm

N^th perfect no.

Generate K^{th} perfect no. in series using only 1 & 2 as digits.

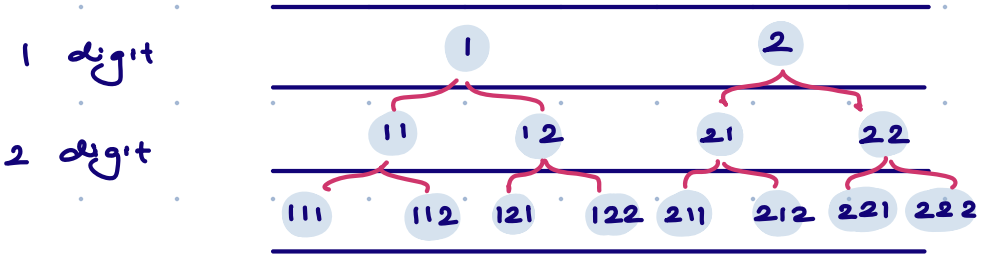
Note → The series should be in increasing order

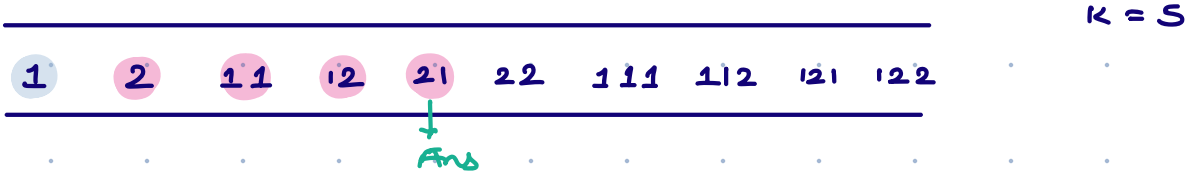
	1	2	11	12	21	22	111	112	121	122
K =	1	2	3	4	5	6	7	8	9	10

Brute force → Start iterating from 1 & keep increasing the count whenever we get the number that is only made up of 1, 2 or its combination.

`if (count == K) return no.`

Idea 2 Data in form of levels / data in digits is following some pattern





```
int solve (N)
{
    if (N ≤ 2) return N

    Queue <Integers> q ;

    q.add(1)
    q.add(2)

    i = 3

    while (i ≤ N) {
        int rem = q.remove();

        a = rem * 10 + 1
        b = rem * 10 + 2

        if (i == N) return a;
        if (i+1 == N) return b;

        q.add(a)
        q.add(b)
        i = i+2 ✓
    }
}
```

N = 1 Ans = 1

N = 2 Ans = 2

N = 7

q = ~~1~~ ~~2~~ 11 12 21 22

queue

1	2	11	12	21	22

a = 111

b = 112

TC : O(n)

SC : O(n)

* Doubly Ended queue (Deque)

→ Data structure where we can go & access elements from both sides.

→ Add elements from front & last

→ Remove elements from front & last.

* Deque can act both as stack & Queue.

Functionalities

01. addfirst()
02. removefirst()
03. getfirst()

TC: O(1)

04. addatlast()
05. Remove last()
06. getlast()

Stack

* **Sliding Window Maximum**

Q Given an integer array A & a window of size K. Find the maximum of all windows.

A[] = {10 1 9 3 7 6 5 11 8} K = 4

Ans[] = {10 9 9 7 11 11}

A[] = {1 4 3 2 5} K = 3

Ans[] = {4 4 5}

Brute force → Generate all subarrays of size K & iterate on the subarray to get the maximum.

TC: $O(n-k+1) * k \simeq O(n^2)$

$K = n/2$

A[] = { 3 15 6 12 4 2 10 9 13 2 }

3 15 6 12 4 2 10 9 13 2

Ans = 15 15 12 12 10 13 13

01. Remove from last → help us to maintain only probable maximum in DS

02 Access the front ele

03 Remove ele from front → help you in dropping previous window ele

04. Add ele from last

```
void maxwindow (int [] A, int k)
    Deque<I> dq = new ArrayDeque();
    // Insert first window
    for (i=0; i<k; i++){
        while (dq.size()>0 && A[i] > dq.getlast())
            dq.removelast();
        dq.addlast(A[i]);
    }
```

```
print ( dq.getfirst());

// For all other window

s = 1
e = k

while ( e < n ) {
    // remove A[s-1] if present
    if ( A[s-1] == dq.getfirst()) {
        dq.removefirst();
    }
    // add A[e]
    while ( dq.size() > 0 && A[e] > dq.getlast())
        dq.removelast();
    dq.addlast (A[e]);
    print ( dq.getfirst());
    s++;
    e++;
}
}
```

TC : O(N)

SC : O(N)