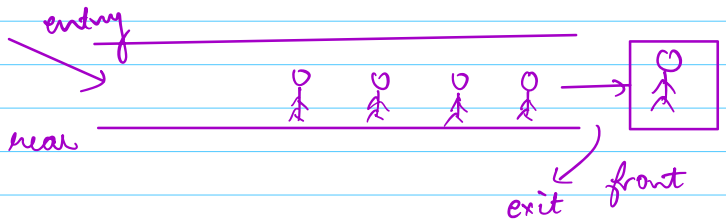
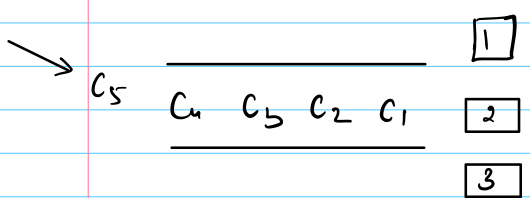
16/1/2024

Queue



customer care

First In First Out



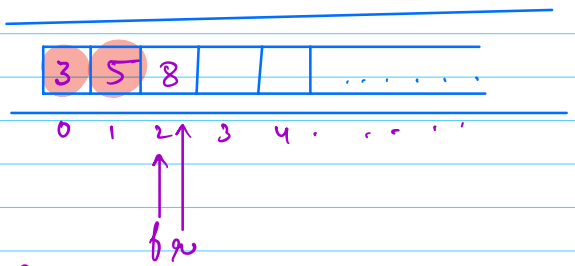
Operations

1) Enqueue (x) ⟶ Insert x from rear end.
2) Deque () ⟶ remove data from front end
3) isEmpty () ⟶ checks if Q is empty
4) Front() ⟶ get data at front end
5) Rear () ⟶ get data at rear end.

} TC: O(1)

Q Implement queue using array.

enqueue(3) ✓
enqueue(5) ✓
enqueue(8) ✓
dequeue() → 3
isEmpty() → false
front() → 5
dequeue() → 5
rear() → 8



Queue → from index f to r (subarray)

f = ~~-1~~ 0
r = ~~-1~~ 0

```
void enqueue(x){
    r++;
    A[r] = x
}
```

// overflow → 1) Use dynamic array.
2) Do not insert more than size

```
int dequeue(){
    if(isEmpty())
        return -1;
    f++;
    return A[f-1];
}
```

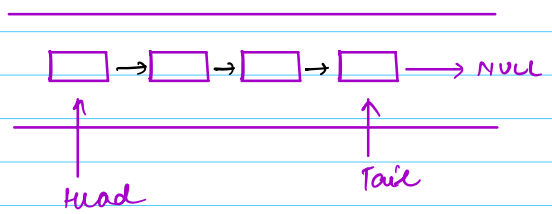
```
boolean isEmpty(){
    return f>r;
}
```

```
int front(){
    if(isEmpty())
        return -1;
    return A[f];
}
```

```
int rear(){
    if(isEmpty())
        return -1;
    return A[r];
}
```

✓ O(1)

Q Implement queue using linked list.



TC:

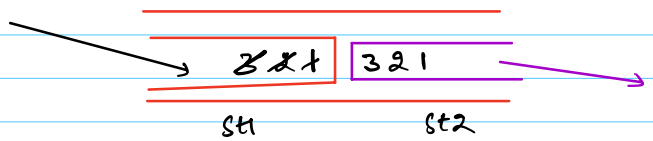
	Insertion	Removal
Head	O(1)	O(1)
Tail	O(1)	O(N)

enqueue(x) → insert at tail
dequeue( ) → removal of head
isEmpty( ) → head == null
front( ) → head.data
rear( ) → tail.data

Q Implement queue using 2 stacks

enqueue(x)
dequeue()
isEmpty()
} O(1)

→ push(x)
→ pop()
→ isEmpty()



enqueue(1)
enqueue(2)
enqueue(3)
dequeue()
enqueue(4)
enqueue(5)

```
void enqueue(x){
    st1.push(x);
}

void move(){
    while(!st1.isEmpty()){
        st2.push(st1.pop());
    }
}

int dequeue(){
    if(isEmpty())
        return -1;

    if(st2.isEmpty())
        move();

    return st2.pop();
}
```

```
boolean isEmpty(){
    return st1.isEmpty()
        && st2.isEmpty();
}
```

amortized TC

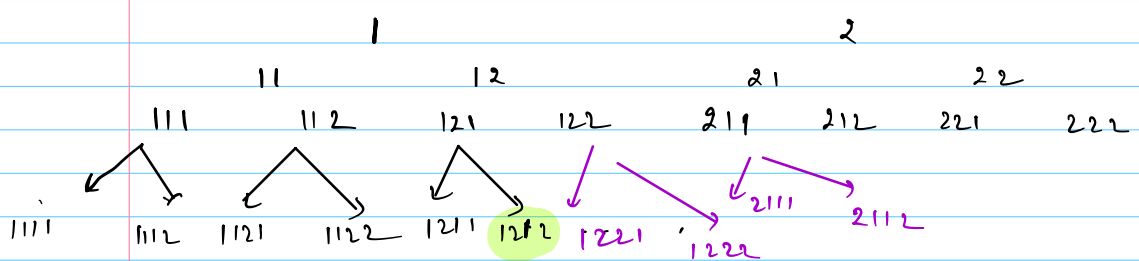
If TC of move() = O(K) → next K dequeue() will have TC: O(1)

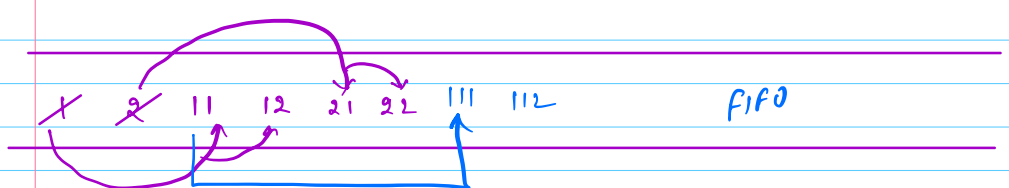
↳ O(2) ≃ O(1)

Meet at 8:30 am IST

Q Given an integer N, find N^{th} number that can be formed by digits 1 & 2 only.

N	1	2	3	4	5	6	7	8	. . . .
No	1	2	11	12	21	22	111	112	121	122





$x \longrightarrow x*10+1$

$x*10+2$

```
int getNthNo (int N) {

    if (N <= 2) return N;
    q.enqueue(1); q.enqueue(2)

    i = 3
    while (i <= N) {
        x = q.dequeue();
        a = x*10+1;
        b = x*10+2;
        if (i == N) return a;
        if (i+1 == N) return b;

        q.enqueue(a);
        q.enqueue(b);

        i+=2;
    }

    return -1;
}
```

N = 10

~~1, 2, 11, 12~~, 21, 22 111 112

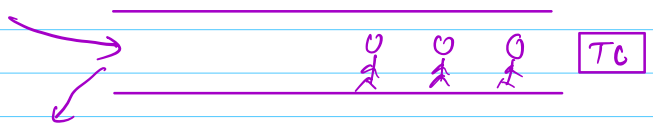
i	x	a	b
~~3~~	~~2~~ 11 12	~~11~~ ~~21~~ ~~111~~ 121	~~12~~ ~~22~~ 112 122
~~5~~			
~~7~~			
9			

TC: O(N)

SC: O(N)

Doubly ended Queue (deck)

1> enqueueFront(x)
2> enqueueRear(x)
3> dequeueFront()
4> dequeueRear()
5> isEmpty()



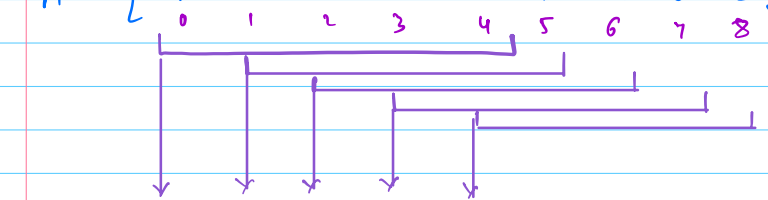
Support undo also
Queue + stack

TC: O(1) of operations.

Q Given an integer array & an integer k.
Find the max element of subarray of size k.

sliding window.

A = [1 8 5 6 7 4 2 0 3] k = 5
0 1 2 3 4 5 6 7 8



O/P → 8 8 7 7 7

A = [1 8 5 6 7 4 2 0 3] k = 4
0 1 2 3 4 5 6 7 8

~~0 1 2 3 4 5 6 7 8~~

l, r
~~2~~ ~~5~~
~~3~~ ~~6~~
~~4~~ ~~7~~
5 8

store ~~data~~ index

TC: O(N)
SC: O(K)

removeInvalidIndex ( ) {

}

Sliding window + dqueue

Queue<Integer> q = new Deque();

Deque<Integer> q.

Coding thru interfaces