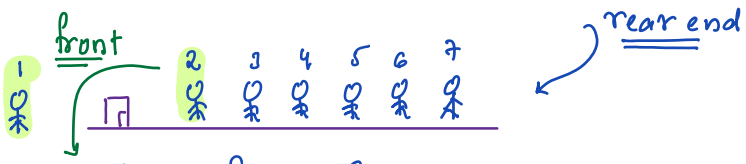
Todays Content:

: Queue basics ✓

: Problems

a) Revern first k ele in Que ✓
b) Implement Que using stacks ✓
c) k^{th} number using only 1 & 2 ✓
d) k^{th} palindrome using only 1 & 2 digits ✓

Queue:



FIFO: First in First out

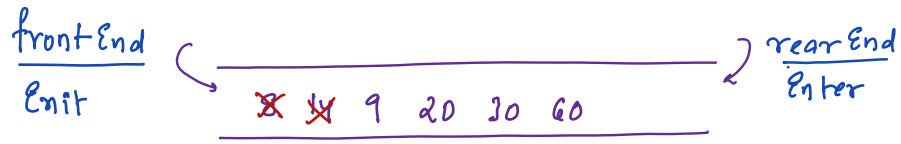
Functions:

- Enqueue(x): x will enter at rear end
- deque(): We deque from front end
- front(): Element at front end
- rear(): Element at rear end

Each function call takes: O(1) time

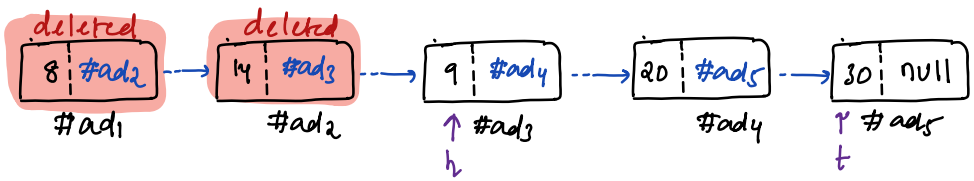
Examples:

8 ✓ 14 ✓ 9 ✓ 20 ✓ ↑ 8 30 front() 14 ↑ 14 rear() 30 60 ↑ 9



Implementation: LinkedList

8 ✓ 14 ✓ 9 ✓ 20 ✓ ↑ ✓ 30 ✓ front() 14 ↑ rear() 30 60 ↑



```
class Node {
    int data
    Node nent;
    Node(int n) {
        data = n
        nent = null
    }
}
```

```
Node h = null, int s = 0
Node t = null,

--- enque(int n) {
    Node nn = new Node(n)
    s = s+1
    if(h == null) {
        h = nn, t = nn;
    }
    else {
        tail.nent = nn
        tail = nn
    }
}
```

```
--- deque() {
    if(h == null) {
        #empty queue
        return;
    }
    Node t = h
    h = h.nent
    t.nent = null
    free(t) // de-allocate memory
    s = s-1;
}
```

```
int size() {
    return s
}
```

```
.... front() {
    if(h == null) {
        #empty queue
        return;
    }
    return h.data
}
```

```
.... rear() {
    if(h == null) {
        #empty queue
        return;
    }
    return t.data
}
```

Library :

→ Please refer Queue library in your language of choice

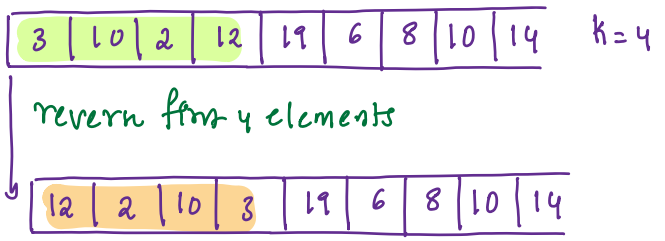
→ Queue<int> q : ⟶ [q.enque(n) q.deque() q.front() q.rear() q.size() A single operation takes O(1)]

Queue ↓ declaring

int ↓ Type of data we store in queue

Q) Given a Queue, Reverse its first k elements from front in given Queue using 1 Stack

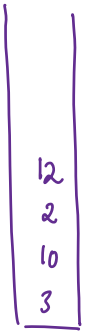
Eg1:



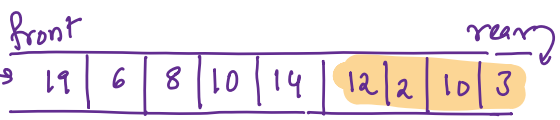
Ideal: k=4

Step1: deque k ele from queue & push in stack

↳ TC: O(k)

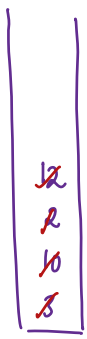


Step2: :

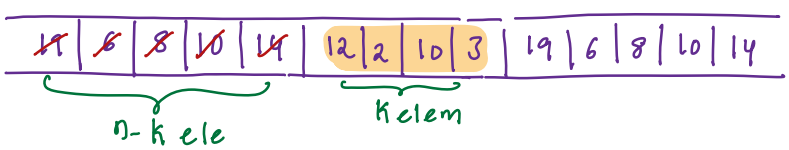


pop k ele from stack & equeue in que

↳ TC: O(k)



Step3:



n represents: Total elements in queue

deque & enque n-k elements from the same queue

↳ TC: O(N-k)

Overall TC: O(N+k) // at max k ≈ N

↳ O(N+N) = O(N)

Code: TODO

28] Implement Queue using Stacks

Queue operations

a) Enque() :
b) Deque() :
c) front() :
d) rear() :

} Every queue function should be implemented using stack functions only.

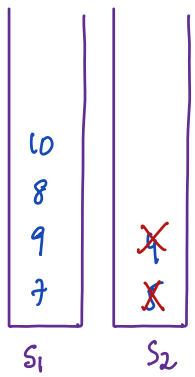
Stack

- push()
- pop()
- top()

Data:

5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 21

5 ✓ 4 ✓ 7 ✓ 9 ✓ deq() 5 8 10 deq() 4 deq() ..... deq() 21



Ideal:

enque(x): push ele in S_1 ⇒ TC: O(1)

deque(): Transfer all ele from $S_1 \rightarrow S_2$ TC: O(N) [for a single deque]

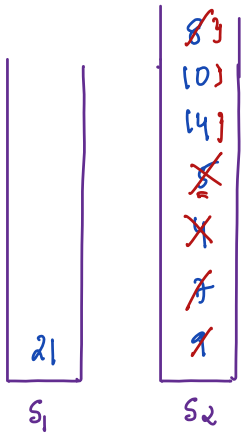
S_2.pop // delete top from S_2

Transfer all ele from $S_2 \rightarrow S_1$

Idea:

5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 21

5 4 7 9 8 10 14



Idea 2:

enque(x): push ele in S_1 → TC: O(1)

deque(): if(S_1.size() + S_2.size() == 0) {
    // Empty cannot delete return
}

if(S_2.size() == 0) → Worst TC: O(N) → avg: O(1)

Transfer all ele from $S_1 \rightarrow S_2$

S_2.pop() // delete top from S_2

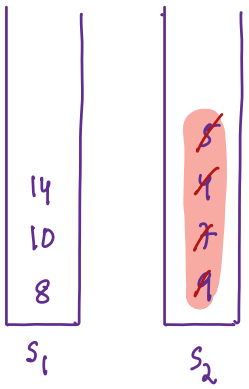
deeper analysis?

// Just for understanding

~~5~~ ~~4~~ ~~7~~ ~~9~~ ~~8~~ 10 14

5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 1

Single Transfer: pop from S_1 & push to S_2: 2 operations



1st deq: Transfer 4 ele from $S_1 \to S_2$ + S_2.pop
(Transfer: 8 ope; pop: 1 ope)

2nd deq: S_2.pop() : 1 ope

3rd deq: S_2.pop() : 1 ope

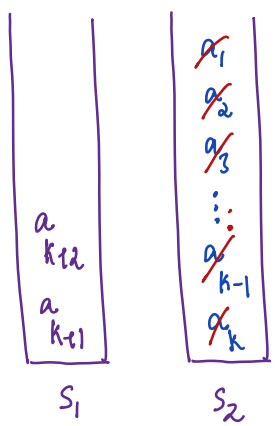
4th deq: S_2.pop() : 1 ope

4 deq = Total operations = 12 oper

avg 1 deq = 3 operations = O(1)

Generalized:

a_1 a_2 a_3 ... a_{k-1} a_k deq() a_{k+1} a_{k+2} deq() ... deq()
(k-1 dele)



1st deq: Transfer k ele from $S_1 \to S_2$ + S_2.pop
(Transfer: 2k op; pop: 1 ope)

2nd deq: S_2.pop() 1 ope

3rd deq: S_2.pop() 1 ope

k deq: S_2.pop() 1 ope

k dedeq: Total operations = 2k + k = 3k

avg 1 deq = 3 ope = TC: O(1)

Total: N = 4N

Total operations performed on a single ele: n = {4 operations} = O(1)

1: enque 3: deque()
TC: O(1)

→ S_1.push(n) : 2 ope

→ { Transfer $S_1 \to S_2$ }: S_1.pop() & S_2.push(n): 2 ope

→ S_2.pop() : // n removed : 1 ope

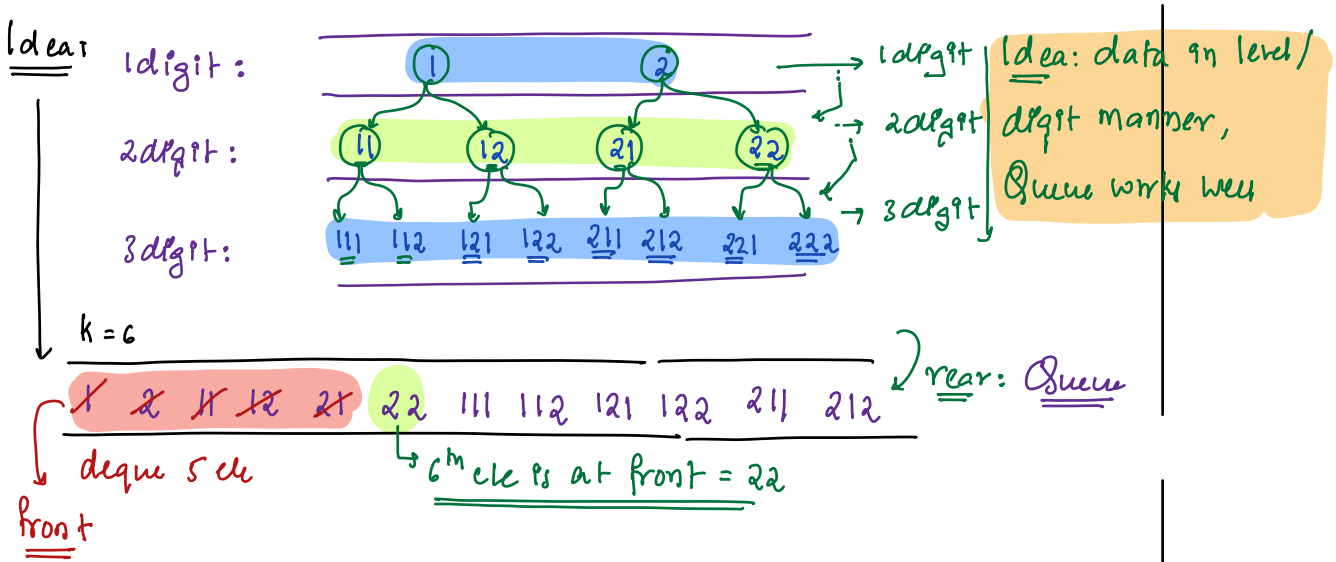
3Q) Generate k^{th} number in series using digits only 1, 2, k input

k = 5 : 1 2 11 12 21 → 5th number

k = 7 : 1 2 11 12 21 22 111 → 7th number

Idea: Start iterating from 1.... so on & for every number, check if number is made of digit 1 & 2, if it's inc count, when count == k return number.

Idea:



Idea: Declare Queue<String> q; // int is not used because it can give overflow

Initialize q with 1 & 2

Delete k-1 times from q:

```
String ele = q.front();
q.deque()
q.enque(ele + "1") // append 1 to ele
q.enque(ele + "2") // apped 2 to ele
```

return q.front()

```
string kth number(int k){   TC: O(k)  SC: O(k)
    Queue<String> q;
    q.enque("1")
    q.enque("2")
    i=1; i<k; i++){
        String ele = q.front();   } delete front element
        q.dequeue()               } -1
        q.enque(ele+"1") // append 1 to ele   } insert elements
        q.enque(ele+"2") // append 2 to ele   } +2
    }
    return q.front()
}
```

In each iteration size increases by 1

$=$

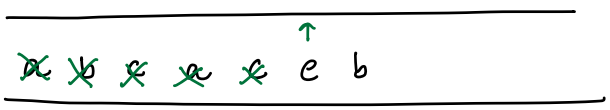
4Q) Generate k^{th} palindrome number using digits 1 & 2 : TODO Friday

Series: 1 2 11 22 111 121 212 222 .....

Sol) Given input of stream of characters for every new input character
Calculate 1st non-repeating character for entire data → medium

Eg: a b c c a e b e a g

ans: a a a a b b e null null g



Store freq of all characters

a: 2
b: 2
c: 2
e: 1

output: a a a b b b e

Idea: New character: x

: update freq in hashmap

: insert x in Q

: while(Q.size()>0 && hm[Q.front()]>1){

// Q.front() cannot be ans, delete

Q.deque()

}

if(Q.size()==0){ print null }

else{ print(Q.front()) }

Repeat above process for entire stream of characters

If say there are N characters TC: O(N) SC: O(N)

TODO: Try it hashset ⇒ {it's not possible}