Todays Content:

1. Queues Basics

Problems:

a. Reverse first k elements of Q }

b. Implement Queue using stacks }

c. k^{th} number using 1 & 2 }

d. k^{th} palindrome using only 1 & 2 digits }

Queue: 1 2 3 4 5 6

obs:

FIFO, first in first out.

Entry at ent happens at diff side

Functions: Exit side: Front Entry side: rear()

Enque(x) : x inside queue rear end:

deque() : delete from front end

front() : return ele at front end

size() : return no: of ele in queue

Examples

8 14 9 20 ↑ 30 front() ↑ front() front() ↑ 60 size()

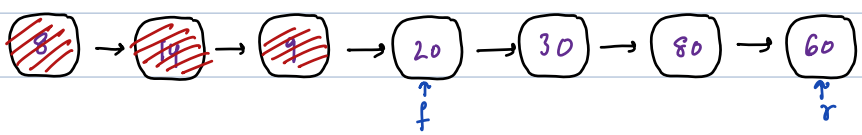
8 14 9 14 9 3

front → ~~8~~ ~~14~~ ~~9~~ 20 30 60 ↩ rear

Implementation: Linked:

8 14 9 20 ↑ 30 front() ↑ 80 60 ↑

return 14



```
class Node{
    int data;
    Node nent;
    Node(int x){
        data=x
    } nent = NULL
}

Node f= null, r=null;
int c=0
void enque(int x){
    Node nn = new Node(x);
    c=c+1;
    if(f==null){ //Empty
        f=nn; r=nn;
    }
    else{ r.nent=nn;
    } r=nn;
}

void deque(){
    if(f==null){ return _ }
    f=f.nent
    c=c-1;
}

int size(){
    return c;
}

int front(){
    if(f==null){return _}
    return f.data();
}
```

Library:

Queue<Type> que = new LinkedList<>(); : JAVA

Functions:

- enque() = que.add(x) : add ele at rear()
- deque() = que.poll() : Remove & return element at front
- front() = que.peek() : Return element at front
- size() = que.size() : Return size()

Note: We can use loops on stack & Queue

We can only access front ele in Queue

We can only acess peek ele in Stack

18. Given an Queue, Reverse its first k elements from front in given Queue using 1 stack.

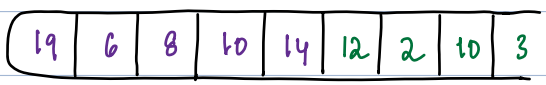
Eg1: 

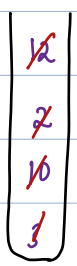
k = 4

Ideal:

k = 4 

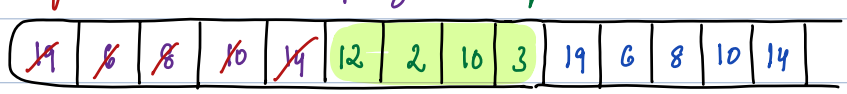
Step1: Deque k element from queue
push them in stack
↳ Iterations : 2k





Step2: Pop k elements from stack
enque them in queue.
↳ Iterations : 2k

Step3: Deque N-k elemens & again enque them.



↳ Iterations : 2(N-k)

Overall TC: $2N+2k = O(N+k)$ Overall SC : $O(k) \rightarrow O(1)$

28. Implement Queue using Stacks

Queue operations:

a. Enque( ):
b. Deque( ):
c. front( ):

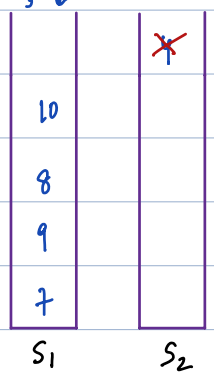
} Every queue function should be implemented using Stack functions only {

Stack

a. push( ):
b. pop( ):
c. peek( ):

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

5*



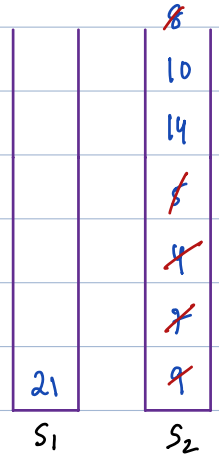
Idea1:

enque(x): push in S_1 : O(1)

deque( ): pop & push all elements from $S_1 \rightarrow S_2$ } : O(N)
pop in S_2
pop & push all elements from $S_2 \rightarrow S_1$ }

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

5* 4* 7* 9X 8



Idea2:

enque(x): push in S_1 : O(1)

deque( ): if(S_2 is Empty) { : Worst case: O(N)
pop & push all elements from $S_1 \rightarrow S_2$ }
pop in S_2

Queue, Just for Understanding:

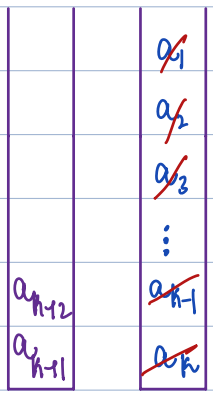
5 4 7 9 8 10 14

front rear

Generalized:

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

a_1* a_2*



Operations

1st deq: pop (1) & push (1) all ele from $S_1 \to S_2$: $2k$

pop from S_2: 1

2nd deq: pop from S_2: 1

3rd deq: pop from S_2: 1

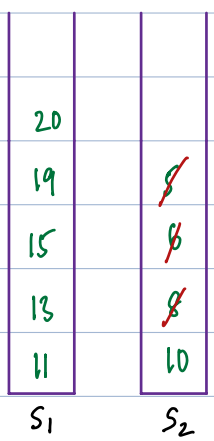
kth deq: pop from S_2: 1

Total k dequeue(): $2k + k = 3k$

avg 1 dequeue(): ≈ 3 operations = O(1) if consider avg

Note: When worst case scenario happens very rarely, we go avg case & write O() notation.

	5	6	8	10	11	13	15	19	20	*
enq:	1	1	1	1	1	1	1	1	1	
Transfer:	2	2	2	2	2	2	2	2	2	
deq:	1									



38. Generate k^{th} number in series, each number in should only contain digits 1, 2

Series: Increasing order

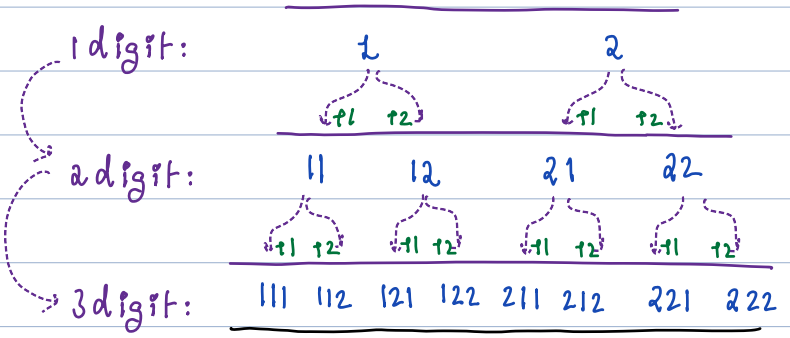
Note: Return k^{th} number in String format

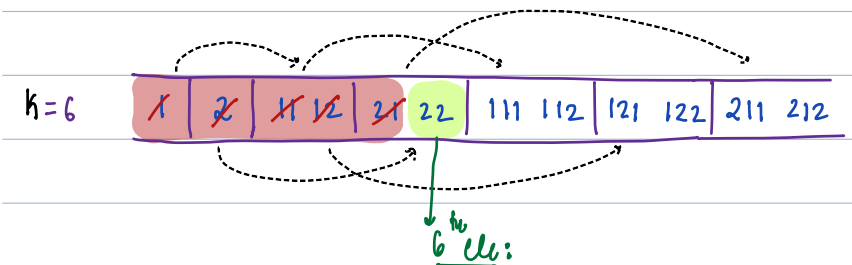
k=5: 1 2 11 12 21 = ans

k=7: 1 2 11 12 21 22 111 = ans

Idea: Generating numbers digit by digit ⇒ Queue very useful!!

First all 1 digit numbers → 2 digit numbers → 3rd digit...





Idea: Queue<String> que; Use String Builder

Enque "1" & "2" in que;

for(int i=1; i<=(k-1); i++){ TC: O(k) SC: O(k)

String f = que.front() } -1

que.deque();

que.enque(f+"1"); } 2 ele

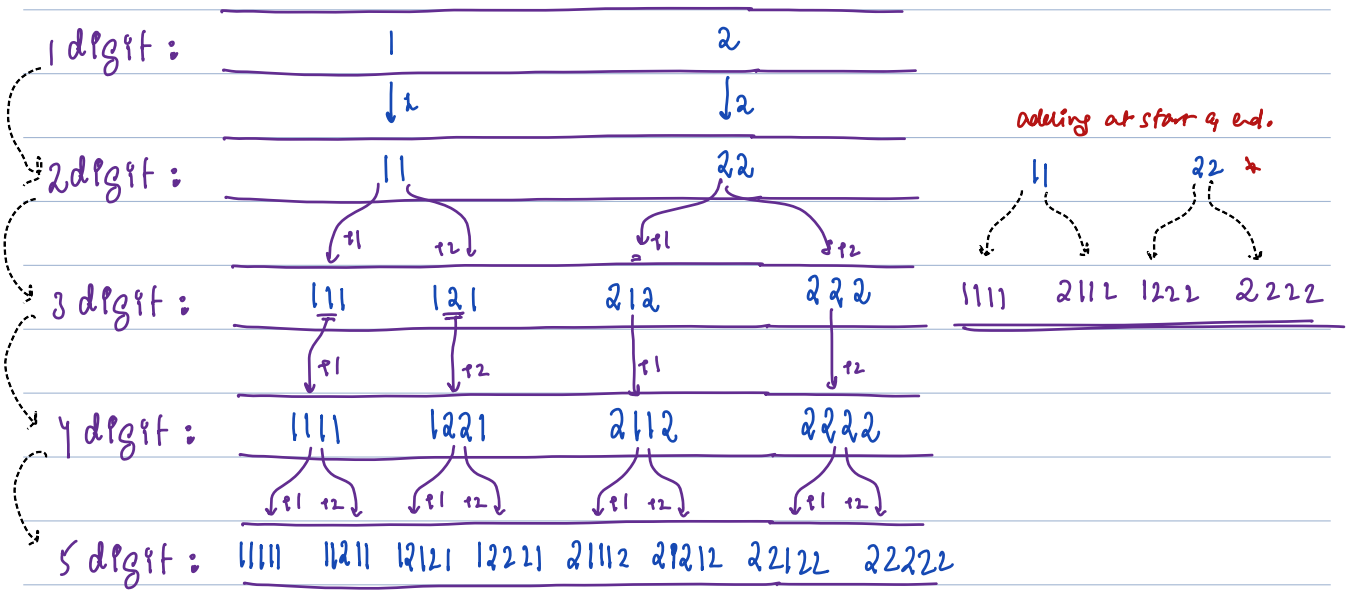
que.enque(f+"2"); } After each iterate size inc by 1

}

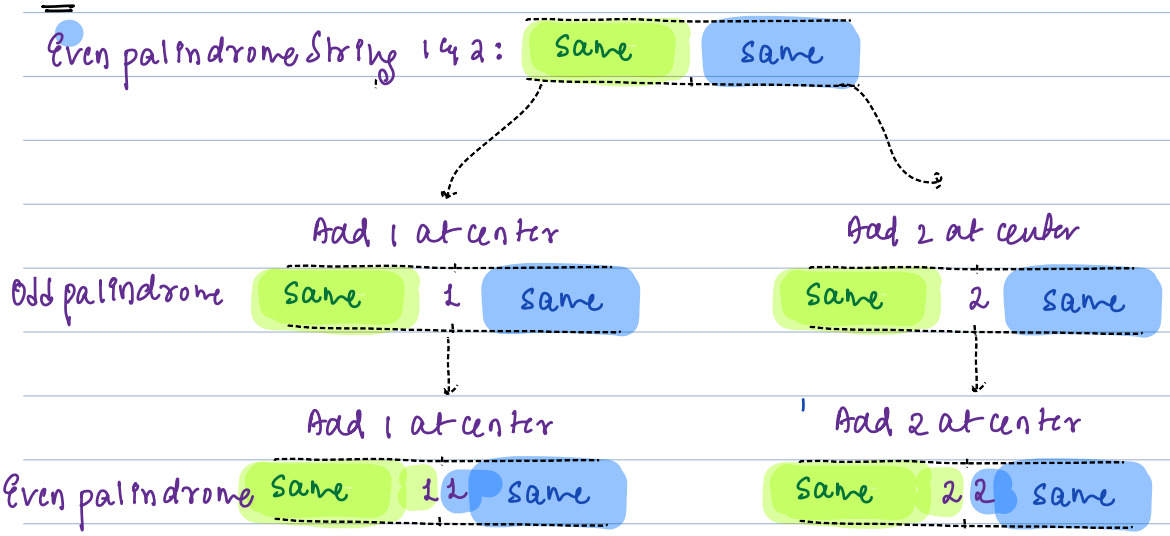
return que.front();

40. Generate k^{th} palindrome number using digits 1 & 2 : TODO

Series : 1 2 11 22 111 121 212



obs:



obs: 1. Even length palindrome : Add 1 or 2 at centre

2. Odd length palindromes : Add center character again at center.