Todays Content:

: Queue basics

: Problems

a) Reverse first k ele in Que

b) Implement Que using stacks

c) k^{th} number

→ Ver : Using digit 1/2

→ k^{th} palindrome number

Queue: FIFO, first in first out

↳ **Operations**:

enque(x) : Insert x at rear end

deque( ) : delete ele at front end

front( ) : Return ele at front end

rear( ) : Return ele at rear end

front end ↳

~~3~~	9	4	10	12	14

↲ rear end

rear( ) = 12

front( ) = 3

enque(14) =

deque( ) = We delete 3

1Q) Given a Queue, Reverse it's first k elements using 1 Stack

Ex1:

3	10	2	12	19	6	8	10	14

$k=4$

reverse 4 ele

12	2	10	3	19	6	8	10	14

Ideal:

3	10	2	12	19	6	8	10	14

$k=4$

Step1: Delete first k elements in Queue & Insert into stack

19	6	8	10	14

→ Stack

12
2
10
3

TC: $O(k)$

Step2: Pop elements from stack & Insert them in Queue

TC: $O(k)$

19	6	8	10	14	12	2	10	3

~~12~~
~~2~~
~~10~~
~~3~~

Step3: Deque( ) & Enque( ) for N-k times, N is total no: of ele

TC: $O(N-k)$

~~19~~	~~6~~	~~8~~	~~10~~	~~14~~	12	2	10	3	19	6	8	10	14

Overall TC: $O(N+k) \approx O(N)$

↳ At max $k=N \approx O(N+N) = O(N)$

Overall SC: $O(1)$

28) Implement Queue using Stacks

Queue operations

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

Every queue function should be implemented using stack operations

Expected TC for a Single Queue : O(1)

Stack

push()

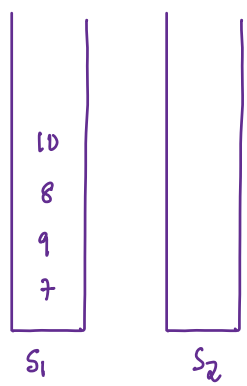
pop()

top()

Data:

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

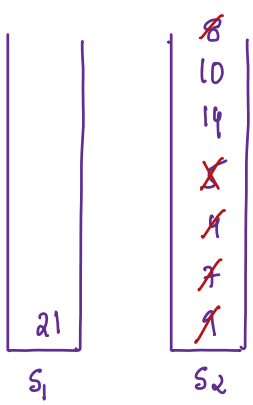
5 4 7 9 deq() 8 10 deq() deq() 14 deq() deq() 21



Idea1:

enque(x) : pushing it in S_1 TC: O(1)

deque() : → move all elements $S_1 \to S_2$
→ Delete top element from S_2
→ move all elements $S_2 \to S_1$ TC: O(N)



Idea2:

int r : rear of que

enque(x) : pushing it in S_1 TC: O(1)
: r = x //update rear

deque() : if(S_2.size() == 0){ TC: O(N), amortized : O(1)
} move all ele from $S_1 \to S_2$
→ Delete top element from S_2

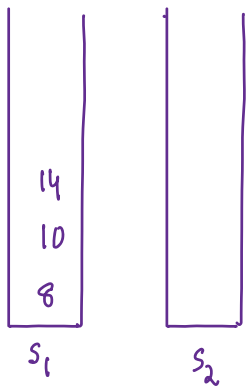
rear() : return r TC: O(1)

front() : if(S_2.size() == 0){ $S_1 \to S_2$ } S_2.top() ≈ amortized O(1)

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

5 4 7

4 7



1st deq(): All elements $S_1 \to S_2$ + Top of S_2
(4 operations) (1 operation)

2nd deq(): Top of S_2 : 1 operation

3rd deq(): Top of S_2 : 1 operation

4th deq(): Top of S_2 : 1 operation

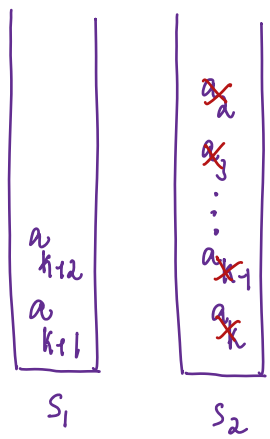
Total 4 deque = 8 operation

On a average = 2 operations
Single deque

Generalized :

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

a_1 $k-1$



1st deq : All elements $S_1 \to S_2$ + Top of S_2
(k time) (2)

2nd deq : Top of S_2 : 2

3rd deq : Top of S_2 : 2

⋮

k^{th} deq : Top of S_2 : 2

Total k deq : 2k operations

On a average + Single deque : O(1)

Note : If freq of costly operation is very very less, to have an indepth analysis in time we will go to amortized analysis

Amortized Analysis :

Calculate avg amount of iterations, for a single operation

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

k = 5 : series = { 1, 2, 11, 12, 21 } ans = 21

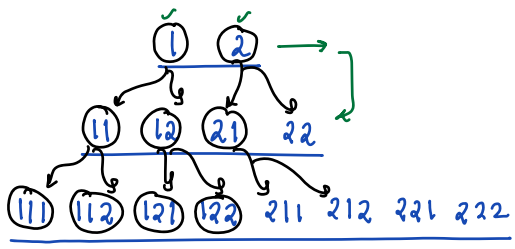
k = 7 : series = { 1, 2, 11, 12, 21, 22, 111 } ans = 111

k = 10 :

1 digit num =

2 digit num =

3 digit num =



~~1~~	~~2~~	~~11~~	~~12~~	~~21~~	22	111	112	121	122	211	212 ...

→ If we delete an element x :

: Insert $x + "1"$

: Insert $x + "2"$

Idea:

→ Que : [1 | 2]

→ for k-1 times delete an element & append 2 elements

→ Ans is at front of Queue

TC: $k \times \{ \underbrace{O(1)}_{\text{deque}} + \underbrace{O(2)}_{\text{2-enque}} \} = O(k)$

SC: $O(k)$

↳ for every iteration size inc by 1, after k iterations size become $O(k)$

```
int kth number( int k){
    Queue<String> q; // Refer in your language of choice
    q.enque("1") q.enque("2");
    i= 1; i< k; i+1){
        String s = q.front()
        q.deque()
        q.enque( s+"1");
        q.enque( s+"2");
    }
    return q.front()
}
```

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

Note: We only need to generate even length Palindrome

$k=5$

2st digit palin : 11 22

4 digit palin : 1111 1221 2112 2222

11 : same character & start & end

1111 2112

11 : same character in between

1111 1221

Trace:

11 22

1111 1221 2112 2222

11 11 11 11 22 11 . . . .

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

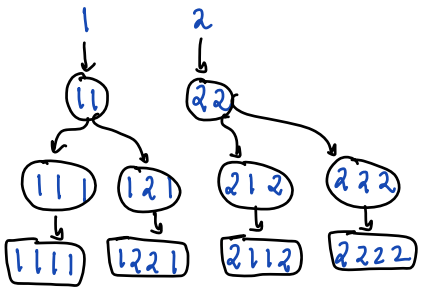
<u>k=10</u> :

1st digit pal :

2 digit pal :

3 digit pal :

4 dig pal :



5 dig pal : 11111 11211 12121 12221 ...

1) even length ⟶ odd length: add 1/2 in centre

2) odd length ⟶ even length: add middle char once again at centre

<u>odd length</u> : (x) <u>1</u> (x) (x) <u>2</u> (x)

↓

<u>even length</u> : (x) <u>1 1</u> (x) (x) <u>2 2</u> (x)