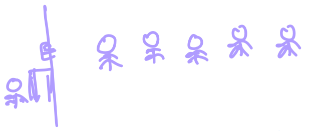
Today's content.

- Introduction to queues
- Queue Implementation using LL
- Reverse first k elements in queue
- Queue Implementation using stacks
- Generate k^{th} number in series using 1 4 2.

Queue Introduction



FIFO → First in first out.

Operations:

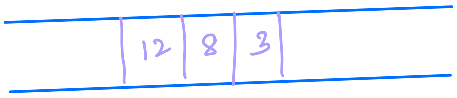
put(x) → enque(x) → insert x at the end of 'q'.

get() → deque() → remove first element from 'q'.

myQue[0] → front() → first ele

my Que[-1] → rear() → last ele .

eq (3), eq(7), eq(12), dq, dq, eq(8), eq(3).



Queue in python.

module → queue .

myQueue = queue.Queue().

methods : put(ele) , get() ,

myQueue[0], myQueue[-1].

Q1: Implement a custom queue class using linked list.

```
class Queue:

    def __init__(self):
        self.front = None
        self.rear = None.
        self.size = 0.

    def enqueue(self, data):
            new_node = Node(data)
            self.size += 1
            if self.rear is None:
                self.rear = new_node
                self.front = new_node.
            else
                self.rear.next_node = new_node.
                self.rear = new_node.

    def deque(self):
        if (size == 0)
            return None
        size = size - 1
        temp = self.front.
        self.front = temp.next.
        if (self.front is None)      / needed if there's only
            self.rear = None.            one element.
        return temp.data.
```

```
class Node:
    def __init__(self, data, next)
    {
          self.data = data.
          self.next = next
     }
```

Q2: Given a queue, Reverse its first k elements using 1 stack.

Ex1 :

3	10	2	12	19	6	8	10	14

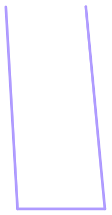
, K=4

↓

12	2	10	3	19	6	8	10	14

Idea: 1) Deque first k elements and insert to stack.

19	6	8	10	14



(k)

2) Insert all the element from stack to queue.

19	6	8	10	14	12	2	10	3

(k).

3) Deque() and enque() first (n-k) element.

12	2	10	3	19	6	8	10	14

(n-k) + (n-k).

TC: O(n).

SC: O(K).

38. Implement queue using stacks.

Queue operations

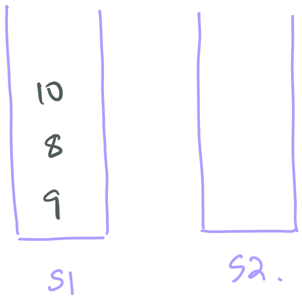
i) Enqueue(ele)
ii) Deque()
iii) front()
iv) rear()

use only stacks.
Expected TC : O(1) in avg case.

Data:

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

deq() ↓ 5, deq() ↓ 4.



Idea:

enque(x) : push it to S1. // O(1).

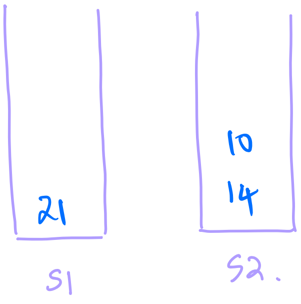
deque() : (i) move all elements from S1->S2.
(ii) Delete top from S2.
(iii) move all elements from S2->S1.

TC: O(n).

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

deq() ↓ 5, deq() ↓ 4., deq() ↓ 7, deq() ↓ 9., deq() ↓ 8

TC: O(n). // worst case
TC: O(1) in avg case



Steps: rear → variable which gets updated to enqued val.

enque(x) : push it to S1. // O(1).

deque() : (i) if(S2.size()==0)
move all elements from S1->S2.
(ii) Delete top from S2.

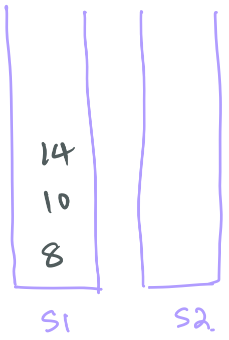
top() : top of S2 if S2 is not empty.
otherwise move S1 → S2 ⇒ return top[S2].

Amortized complexity: → Average amount of iterations for a single operation.

No. of operations for **deque:**

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

$$\underset{5}{\text{deq()}} \quad 8 \quad 10 \quad \underset{4.}{\text{deq()}} \quad \underset{7}{\text{deq()}} \quad 14 \quad \underset{9.}{\text{deq()}} \quad \underset{8}{\text{deq()}}$$



1st deq → $\underbrace{\text{All elements from S1→S2.}}_{\text{4 operations}}$ + $\underbrace{\text{delete 5}}_{\text{1 operation.}}$

2nd deq → delete 4. // 1 operation.

3rd deq → delete 7. // 1 operation.

4th deq → delete 9 // 1 operation.

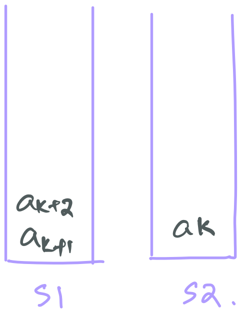
4 deq method calls = 8 operations.

On an average, for single deq = 2 operations → **Avg TC: O(1).**

10:26:00
10:36:00
34.

Generalization:

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



1st deq() → $\underbrace{\text{move from S1→S2}}_{\text{k operations}}$ + $\underbrace{\text{delete top.}}_{\text{1 operations.}}$

2nd deq() → delete top. → 1 operations.

3rd deq() → delete top. → 1 operations.

⋮

kth deq() → delete top. → 1 operations.

Total operations for k deq()calls → 2k ⇒ 1 deq() call → 2 operations. (Average).

Q4: Generate k^{th} number in series using digits 1 and 2 only.

k = 5 ; series : [1, 2, 11, 12, 21]

k = 7 ; series : [1, 2, 11, 12, 21, 22, 111]

k = 10.

1 digit numbers : [1, 2].

2 digit nos. : [11, 12, 21, 22]

3 digit nos : [111, 112, 121, 122, 211, 212, 221, 222].

~~1~~	~~2~~	~~11~~	~~12~~	21	22	111	112	121	122	- - - -

When to queues (prompting).

(i) Generate no's.

(ii) Level by level traversal.

Steps:

1) Initialize queue [1 | 2]

2) For (k-1) times you can deque & append 2 eles.

→ m < enque(m1)
enque(m2)

→ deque() → "x".

enque(x+"1")

enque(x+"2").

3) Ans is at front of queue.

TC: [O(1) + O(1)] (k-1) => O(k).

SC: O(k).

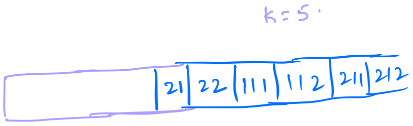
Code:

```
def   kthNumber( k)
{                                          import module queue.
     myQueue = queue.Queue().

     myQueue.put("1")
     myQueue.put("2")
     for  i  in  range(1 ,k)
          x =  myQueue.get()  // remove ele from queue.
          myQueue.put( x+"1")
          myQueue.put (x+"2")

     return  myQueue[0]

}
```



i= 1 , 2, 3 , 4

x = 1, 2, 11, 12