DEQUE

> The first step to achieving your goal, is to take a moment to respect your goal. Know what it means to you to achieve it.
>
> DWAYNE JOHNSON

Good Morning

Content

01. First non repeating character in a stream
02. Deque Intro
03. Maximum in window of size K

01. Given a string. Find the **first non repeating character**.

Note :- All the character will be lowercase characters

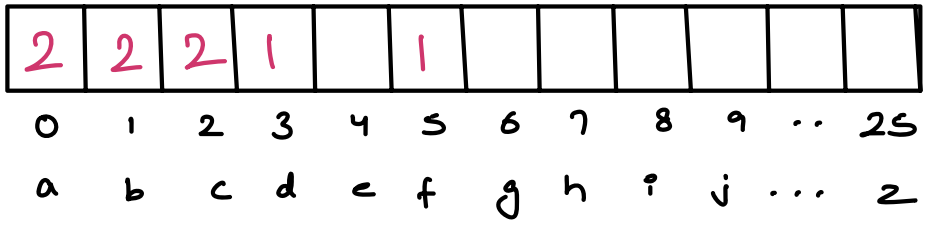
Eg :- e c h e d f c → h

a b c d a c b → d

a x a x a → #

Idea 1 → Create a freq arr & iterate on the string to get your ans.

Eg :- b c f d c a a b



Step 1 → Iterate over string & populate the freq arr. TC - O(n)

Step 2 → Iterate over string again & look for the char freq == 1. TC = O(n)

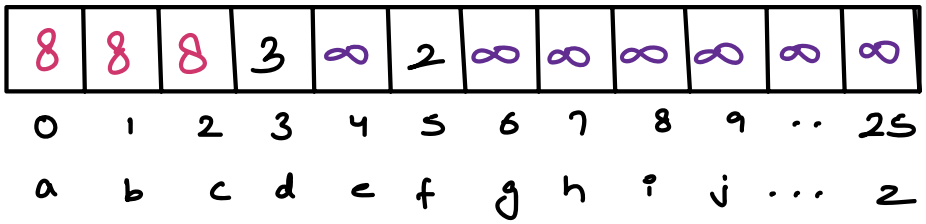
TC = O(n+n) ≅ O(n) SC : O(K)

Try to optimise this

Approach →

Eg:-

0	1	2	3	4	5	6	7
b	c	f	d	c	a	a	b



∞ = Not seen

[0-n-1] = if seen only time

-1 = if seen > once

Steps 1 → Iterate over string & populate the index arr with indices. TC=O(n)

Steps 2 → Iterate on index arr & look for the min index. TC = O(k)

TC = O(n+k) SC: O(k)

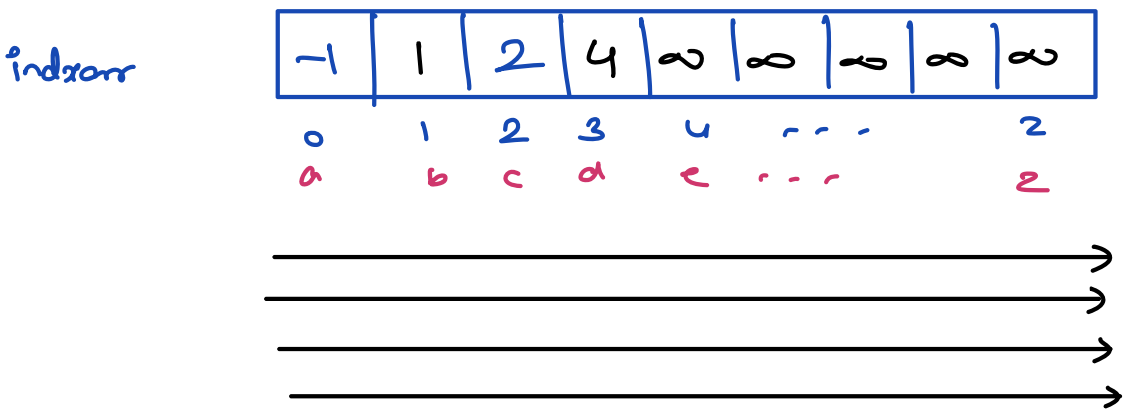
```
char fnrc(String s)
    int [] idxarr = new int[26]
     Arrays.fill(idxarr, Integer.MAX_VALUE);
    for(i=0; i<s.length(); i++){
        char ch = s.charAt(i);
        if(idxarr[ch-'a'] > 0 && idxarr[ch-'a']<n)
            idxarr[ch-'a'] = -1
        }
        else if(idx[ch-'a'] == Integer.MAX_VALUE)
            idxarr[ch-'a'] = i
        }
    }

    int minidx = -1
    int min = Integer.MAX_VALUE;

    for(i=0; i<26; i++){
        if(idxarr[i] != -1){
            if(idxarr[i] < min){
                min = idxarr[i];
                minidx = i
            }
        }
    }
    return (char)(minidx+'a')
```

02. Given a stream of characters & after adding every character, find the first non-repeating character so for.

Stream : a b c a d e d a b e c g
0 1 2 3 4 5 6

FNRC : a a a b b b b b c c #g

Idea : → Use the index array



TC = O(n * k)

SC = O(k)
↳ 26

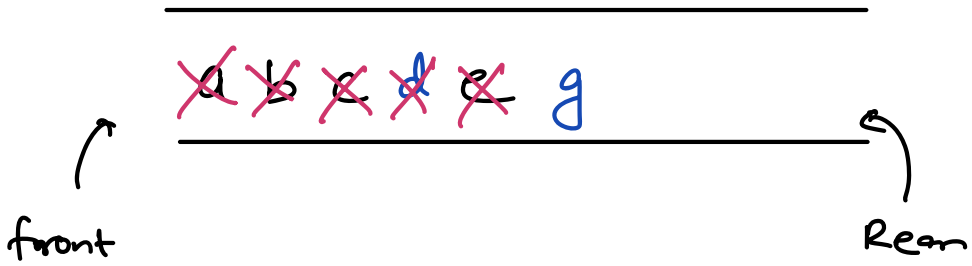
Stream : a b c b a d e d a b e c g

FNRC : a a a

Queue

Stream : a b b c a d e d a b e c g

FNRC : a a a a c c c c c c c # g



HashMap

a = x 2 3

b = x 2 3

c = x 2

d = x 2

e = x 2

g = 1

Step 1 → Update the freq in HM

Step 2 → if freq of curr ch == 1 → insert it in Queue

Step 3 → Remove all the front ch that has freq > 1

Step 4 → Front ele has to be printed

Idea: New character (x)

01. Update the freq of x in hm

02. if (hm.get(x) == 1) ⟶ insert it in queue

02. while (q.size() > 0 && hm.get(q.peek()) > 1)

    {
        q.remove();
    }

04. if (q.size() == 0) → print #

    else print (q.peek());

}

TC = O(n)

SC = O(n)

8:07 → 8:17 AM

Deque : Double ended queue

→ Addition or removal can be done from both the ends

Functionality TC: O(1)

01	addfirst ( )	addLast ( )
02.	removefirst ( )	removelast ( )
03.	getfirst ( )	getlast ( )

→ A deque provides the functionality of both queue & stack

* Using LinkedList

01.	insert at start	→	O(1)
02	insert at back	→	O(1)
03.	Delete from strart	→	O(1)
04.	Delete from end	→	O(n)

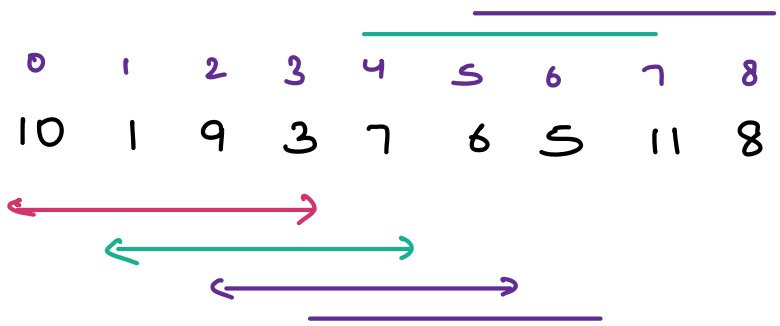
Doubly Linkedlist Implementation ↓ { TODO }

Inbuilt Deque

```
Deque<Integer> dq = new ArrayDeque<>();
```

01. Given ar[N] & K, find max element in every window of size K

Ex1: ar[] = { 10 1 9 3 7 6 5 11 8 }



K = 4

Output : 10 9 9 7 11 11

Idea 1 → For every subarray of len = k, iterate & find max

TC = (n-k+1) * k

SC: O(1)

$= k \simeq n/2 = (n - \frac{n}{2} + 1) * \frac{n}{2}$

$= (\frac{n}{2} + 1) * \frac{n}{2}$

$\simeq O(n^2)$

Idea 2 → Sliding Window → Not going to work

	0	1	2	3	4	5	6	7	8
ar[] = {	10	1	9	3	7	6	5	11	8 }

K = 4

max = 10

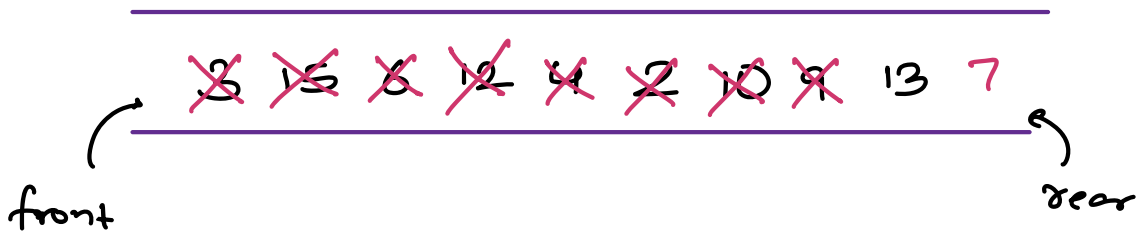
max = 9

Ex: ar =

3	15	6	12	4	2	10	9	13	7	2	5	3
0	1	2	3	4	5	6	7	8	9	10	11	12

K = 4

Deque



Print → 15, 15, 12, 12, 10, 13, 13

Obs

rear ele < curr ele → remove it

rear ele > curr ele → insert the curr ele

Max of window = front of deque

arr[] = { 10 6 10 3 9 11 }

(0 1 2 3 4 5)

K = 4

~~10~~ ~~6~~ ~~10~~ ~~3~~ ~~9~~ 11

print → 10 , 10 , 11

```
void submax ( int [] arr, int k)
    Deque<Integer> dq = new ArrayDeque<>();

    for ( i=0; i<k ; i++) {
        while (dq.size()>0 && arr[i] > dq.getlast())
            dq.removeLast();
        }
        dq.insertlast (arr[i]);
    }

    print (dq.getfirst());
```

```
for ( i=k; i<n; i++ )
    while (dq.size()>0 && arr[i] > dq.getlast())
        dq.removeLast();
    dq.insertlast(arr[i]);
    if (dq.getfirst() == arr[i-k])
        dq.removeFirst();
    print (dq.getfirst());
}
```