Agenda

1) First non-repeating character **
2) Intro to deque (Doubly ended queue)
3) Sliding window maximum **

Q.1 Given a string A, denoting stream of lowercase alphabets. Find first non-repeating char, each time a char is coming in A string stream.

↳ first non-repeating char till ith index

A = a b a b c

ans: a a b # c

Expected TC: O(n)

A = a b c a c e b

ans: a a a b b b e

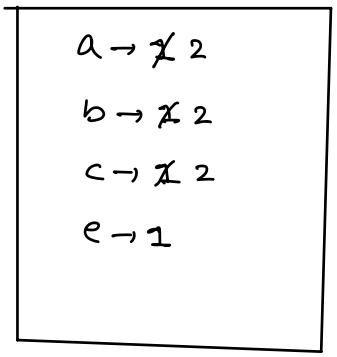
A = a b c a c b d K a d

ans: a a a b b #d d d K

A = a b c a c e b

ans: a a a b b b e

q: 



map

```
HashMap<Character, Integer> map = new HashMap<>();
Queue<Character> q = new ArrayDeque<>();
StringBuilder ans = new StringBuilder();
for(int i=0; i<A.length(); i++) {
    char ch = A.charAt(i);
    if(ch is coming first time) {
        map.put(ch, 1);
        q.add(ch);
    }
    else {
        map.put(ch, updated_freq);
    }
    // find first non-repeating char till now
    while(q.size()>0 && map.get(q.peek())>1) {
        q.remove();
    }
    if q.size()==0, there is no first non-repeating
    char so add # in ans.
    else add q.peek() in ans because it is the
    first non-repeating char till now.
}
```

at max N addition in q

at max N removal from q

itr: 2N

TC: O(N)

SC due to map: O(26)

SC due to q: O(26)

SC: O(1)

```
for (int i=0; i<A.length(); i++) {
    char ch = A.charAt(i);
    if(ch is coming first time) {
        map.put(ch, 1);
        q.add(ch);
    }
    else {
        map.put(ch, updated_freq);
    }
    // find first non-repeating char till now
    while(q.size()>0 && map.get(q.peek())>1) {
        q.remove();
    }
    if q.size()==0, there is no first non-repeating
    char so add # in ans.
    else add q.peek() in ans because it is the
    first non-repeating char till now.
}
```

A = a b c a c b d k a d

ans: a a a b b # d d d k

a	b	c	d	k

a → ~~1~~ ~~2~~ 3

b → ~~1~~ 2

c → ~~1~~ 2

d → ~~1~~ 2

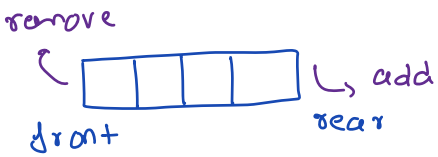
k → 1

map

(char vs int)

Intro to Deque

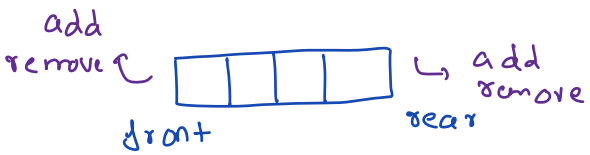
Doubly ended queue (Deque)



normal queue

add happens at last

removal happens from front



deque

add and removal can happen from both end in just O(1).

{ with the help of DLL we can create Deque }

Create and use Deque in Java

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

```
//functions:
dq.addLast(x)          or   dq.add(x)
dq.addFirst(x)
dq.removeLast()
dq.removeFirst()       or   dq.remove()
dq.getLast()
dq.getFirst()
```

TC: O(1)

Q.2 **Sliding window Maximum** { Hard }

Given an array of int values A[] and K, find max of every subarray of length K in A[].

How many subarrays of K length in an array of n length

= $n-k+1$

	0	1	2	3	4	5	6	7	8	
A =	10	2	9	3	1	6	5	11	8	K = 3
ans:	10	9	9	6	6	11	11			

	0	1	2	3	4	5	6	7	8	9	
A =	3	15	16	12	4	2	10	9	13	7	K = 4
ans:	16	16	16	12	10	13	13				

Brute force

Go on every subarray and length K and travel to find its max.

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

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

$\approx O(n^2)$

```
          i   j
          0   1   2   3   4   5   6   7   8
A =      10   2   9   3   1   6   5  11   8        K = 3
```

max = 10 { single max var is not enough }

aquire j

release (i-1)

```
                                  i       j
          0   1   2   3   4   5   6   7   8
A =      10   2   9   3   1   6   5  11   8        K = 3
```

ans: 10 9 9 6 6 11 11

dq | 10 | 2 | 9 | 3 | 1 | 6 | 5 | 11 | 8 |

Keep values in dq. →

give impact of A[j]

remove impact of A[i-1]

```java
static void sliding_window_max(int[]arr,int k) {
    int n = arr.length;
    ArrayDeque<Integer>dq = new ArrayDeque<>();

    //calculate the ans of first window
    for(int i=0; i < k;i++) {
        while(dq.size() > 0 && dq.getLast() < arr[i]) {
            dq.removeLast();
        }
        dq.addLast(arr[i]);
    }

    System.out.println(dq.getFirst());


    //travel the rest of windows
    int i=1, j=k;
    while(j < n) {
        //give impact of arr[j]
        while(dq.size() > 0 && dq.getLast() < arr[j]) {
            dq.removeLast();
        }
        dq.addLast(arr[j]);


        //remove impact of arr[i-1]
        if(dq.getFirst() == arr[i-1]) {
            dq.removeFirst();
        }

        //now dq is storing the impact of current window
        System.out.println(dq.getFirst());

        i++;
        j++;
    }
}
```

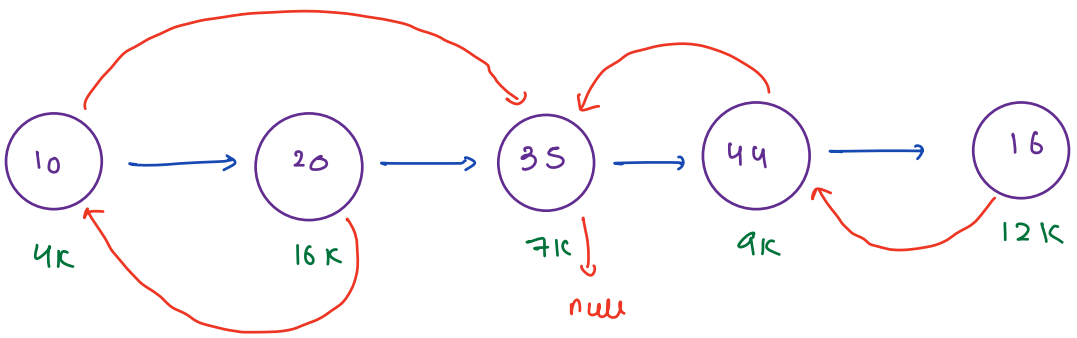
K = 4

					i			j
	0	1	2	3	4	5	6	7
A =	3	15	16	12	4	2	10	19
o/p:	16	16	16	12	19			

dq: ~~3~~ | ~~15~~ | ~~16~~ | ~~12~~ | ~~4~~ | ~~2~~ | ~~10~~ | 19

Doubts

```
RandomListNode {
        int label;
        RandomListNode next;
        RandomListNode random;
        ==
}
```



→ (next)

→ (random)

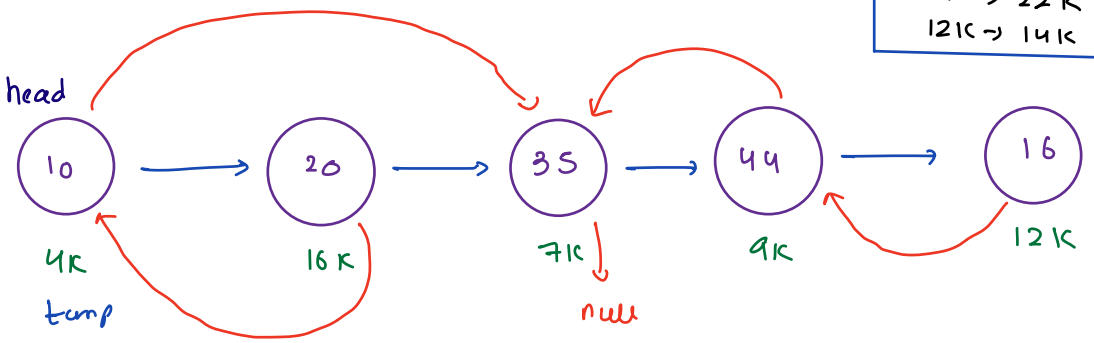
```
temp = 4K;
```

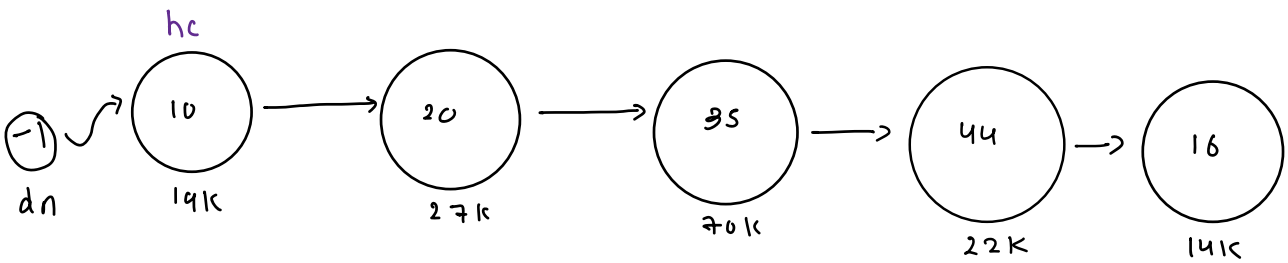
```
sopln(temp.next.random.label);
```

// just copy data and next

4K → 19K
18K → 27K
7K → 70K
9K → 22K
12K → 14K

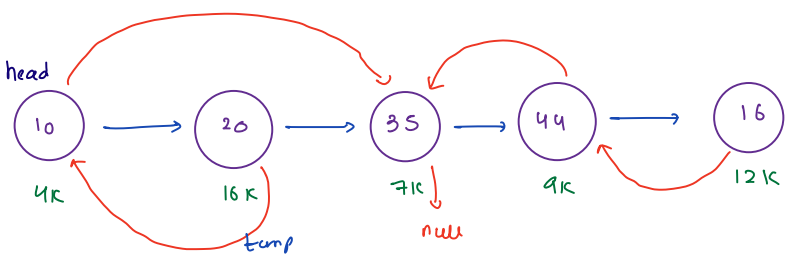
Old vs new

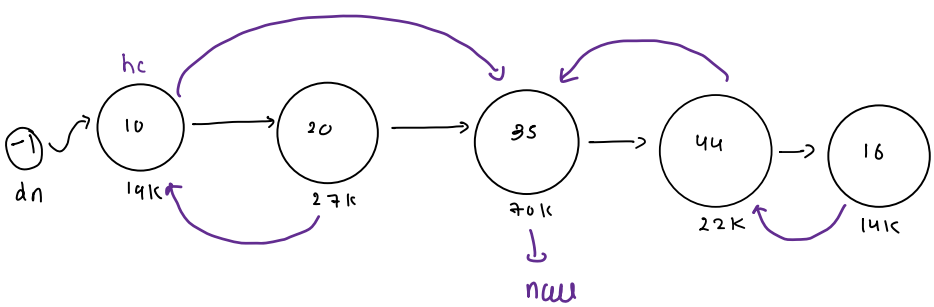




// to copy random pointer

4K → 19K
18K → 27K
7K → 70K
9K → 22K
12K → 14K





old = temp

nn = map.get(temp)

nn.random = map.get(old.random)

A = [1234, 2345, 4567, 5678]

1 → 1
2 → 2
3 → 2
4 → 3
5 → 3
6 → 2
7 → 2
8 → 1

```
for ( i => 0 to A.length-1) {
    int num = A[i];
    while (num > 0) {
        int d = num % 10;
        num = num / 10;
        if (map.containsKey(d) == false) {
            map.put(d,1);
        }
        else {
            map.put(d, updated-freq);
        }
    }
}
int ans = 0, max = 0;
// travel map
for( int key : map.keySet() ) {
    int val = map.get(key);
    if( val > max ) {
        ans = key;
        max = val;
    }
}
```

variable name

1 → 1
2 → 2
3 → 2
4 → 3
5 → 3
6 → 2
7 → 2
8 → 1

ans = ~~0~~ ~~1~~ ~~2~~ 4

max = ~~0~~ ~~1~~ ~~2~~ 3 ✓