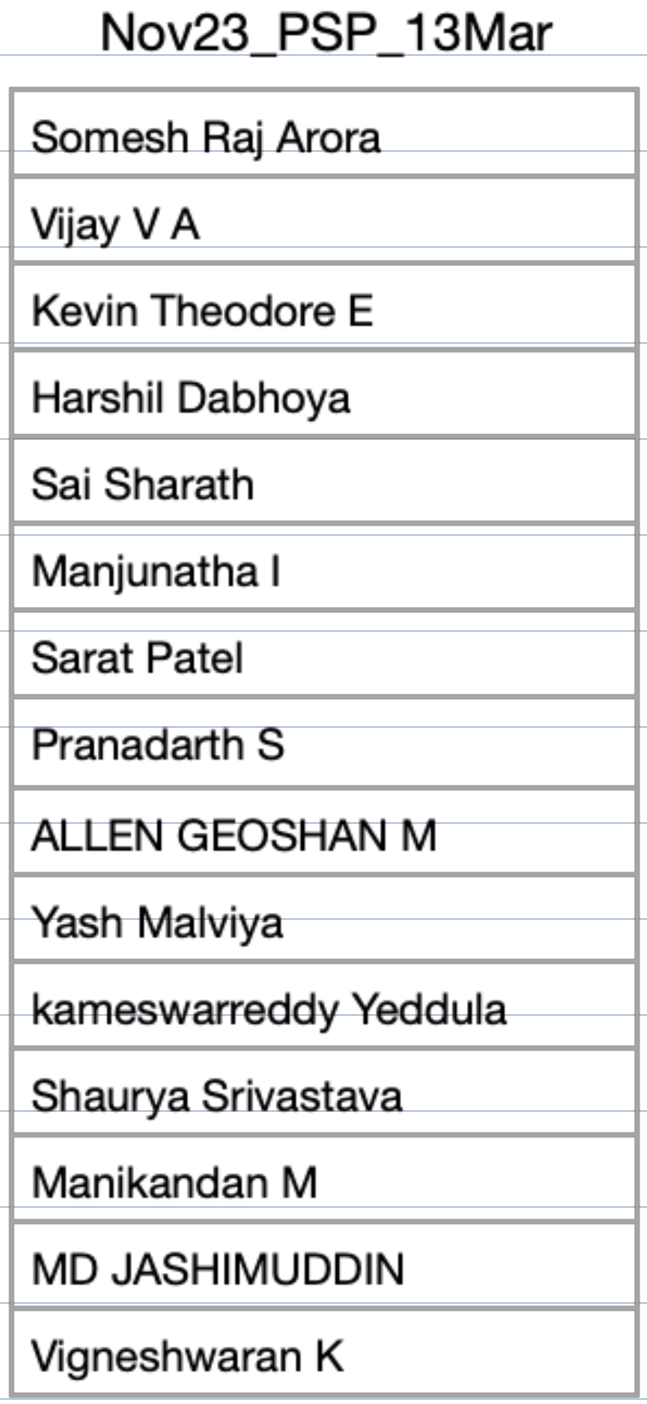
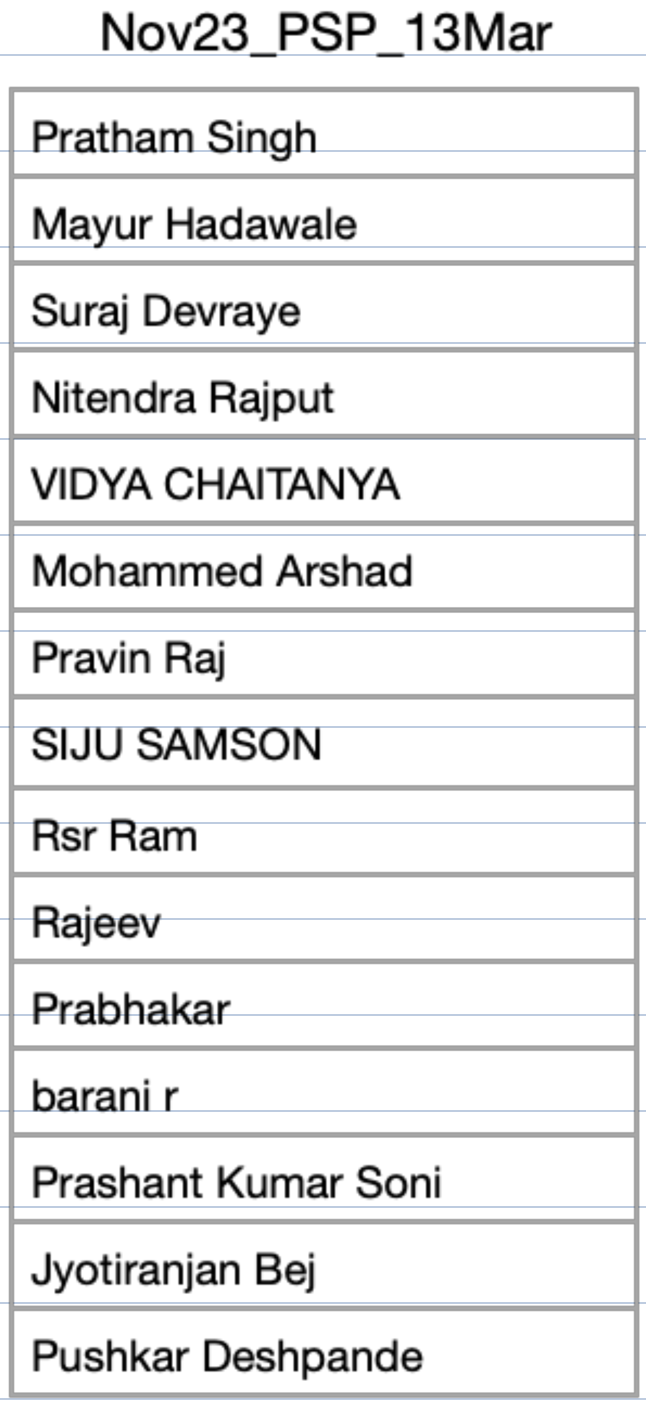
Average PSP : 55.71 → 60



Nov23_PSP_13Mar
Somesh Raj Arora
Vijay V A
Kevin Theodore E
Harshil Dabhoya
Sai Sharath
Manjunatha I
Sarat Patel
Pranadarth S
ALLEN GEOSHAN M
Yash Malviya
kameswarreddy Yeddula
Shaurya Srivastava
Manikandan M
MD JASHIMUDDIN
Vigneshwaran K



Nov23_PSP_13Mar
Pratham Singh
Mayur Hadawale
Suraj Devraye
Nitendra Rajput
VIDYA CHAITANYA
Mohammed Arshad
Pravin Raj
SIJU SAMSON
Rsr Ram
Rajeev
Prabhakar
barani r
Prashant Kumar Soni
Jyotiranjan Bej
Pushkar Deshpande

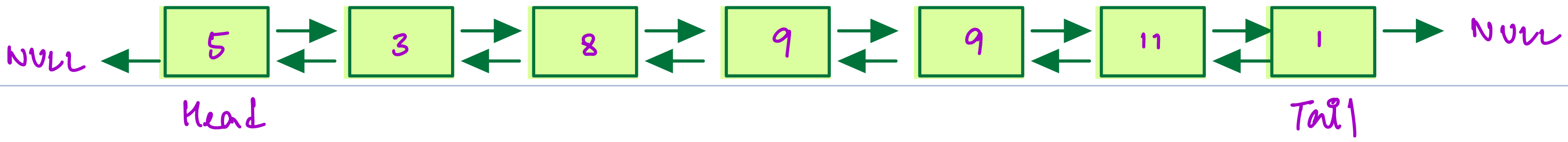
Agenda

Introduction to Doubly linked list

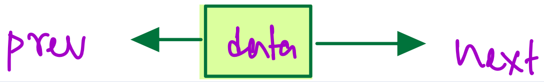
least Recently used

Deep copy of linked list

Doubly linked list

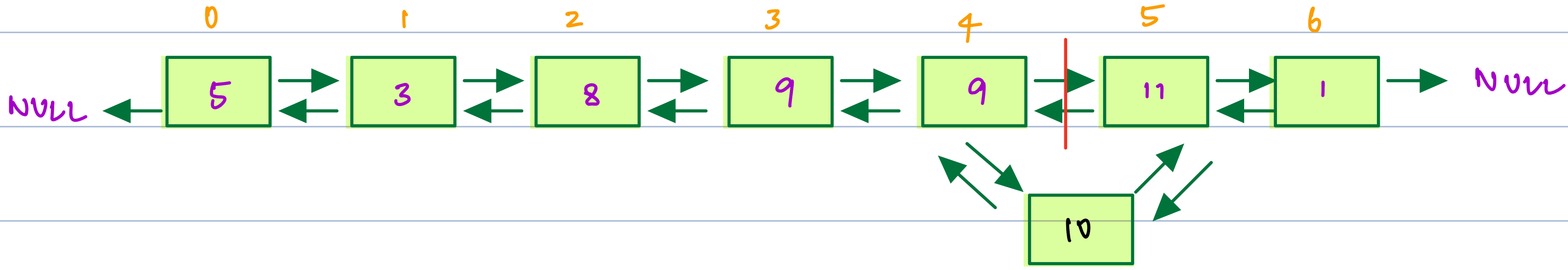


Same as single linked list, except we store the previous pointer. This enables us to move from tail to head



Question 1

Given a doubly linked list. A node is to be inserted with data x at index k, where 0 <= k <= N



example :

x = 10 k = 5

Steps to execute

① Create a new node with data = x

```
Node nx = new Node (x);
```

2 Handle all edge cases.

```
// Given linked list is empty
if (head == null) return nx;

// we are trying to insert at k=0
if (k==0) {
        head. prev = nx;
        nx. next = head;
        return nx;
}
```

3 Iterate k-1 times

```
temp = head;
for (i=1; i<=k-1; i++) temp= temp. nxt
// update the 4 pointers
  nx. next = temp. next
  nx. prev = temp;
  if (temp. next != null)
        temp. next . prev = nx
  temp. next = nx;
  return head
```

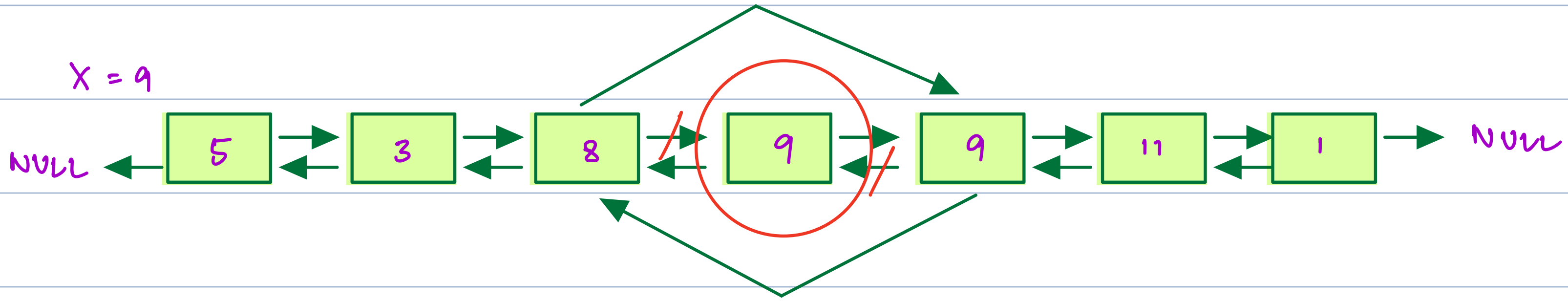
Time and space complexity

T.C = O(n)   S.C = O(1)

Question

Given a doubly linked list, delete the first occurrence of x. If it is not present, no update.

X = 9



If you are already given the address of node to delete. List of operations are

```
temp. prev. next = temp. next;
temp. next. prev = temp. prev;
```

Steps to execute:

① Search for first occurrence of x

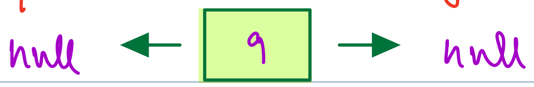
```
temp = head;
```

```
while (temp != null) {
    if (temp.data == x) break;
    temp = temp.next;
}
```

② Handle edge case

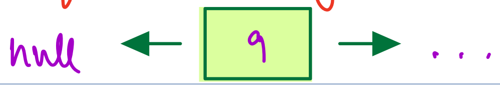
```
// Data does not exist
if (temp == null) { return head }
```

```
// temp is the only node available
```



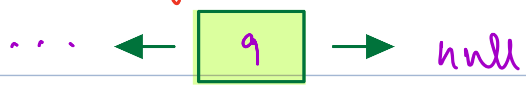
```
if (temp.prev == null && temp.next == null)
        return null
```

```
// temp is my head node
```



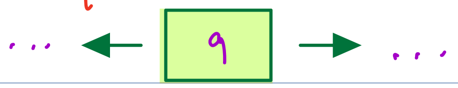
```
if (temp.prev == null){
        temp.next.prev = NULL;
        return temp.next;
}
```

// temp is my last node



```
if (temp.next == null) {
    temp.prev.next = null
    return head;
}
```

// temp in middle



```
else {
    temp.prev.next = temp.next;
    temp.next.prev = temp.prev;
    return head;
}
```

T.C = O(n)   S.C = O(1)

Question 3

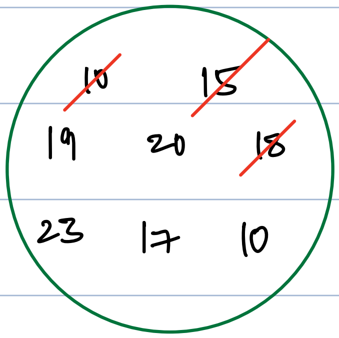
Given a running stream of integers and a fixed memory of size M. Maintain the latest M elements in memory. In case memory is full, delete the least recent element

LRU

example

10	15	19	20	18	23	20	19	17	17	10	....

, M=5



old -- new

~~10~~ ~~15~~ ~~19~~ ~~20~~ ~~18~~ 23 20 19 ~~17~~ 17 10

once the memory is full, for all intake x

x is not present

① Delete the least recent

② Add x as my most

If x is already present

① Delete x from its position

② Insert x as most recent

recent

Operations we are performing

① For all intake of x, search if it is already part of memory

Hashset / Hashmap

② Maintain order of recency to insert and delete

Stacks | Queue | Arrays | Linked List

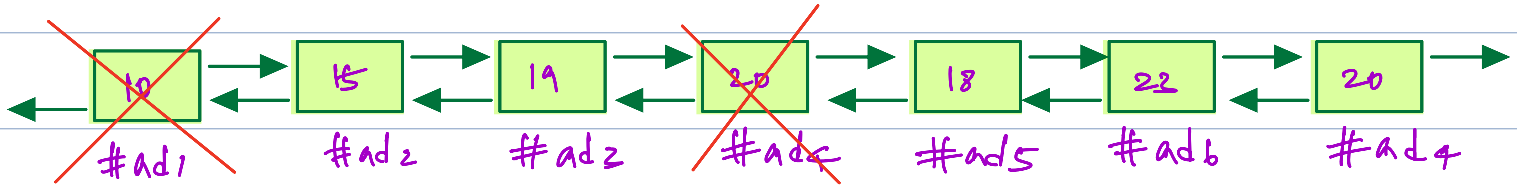
Dry Run

10	15	19	20	18	23	20	19	17	

M=5

Data	addr
~~10~~	~~#ad1~~
15	#ad2
19	#ad3
20	#ad4
18	#ad5
23	#ad6

Data	addr



#pseudo code

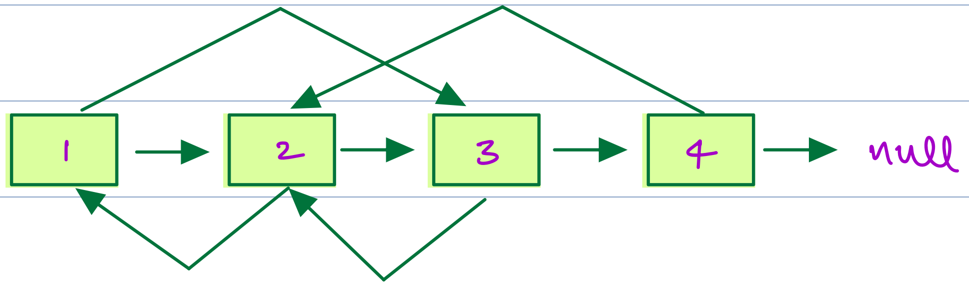
```
Hashmap <Interger, Node> hm = new HM<>();
// empty LL
  Head = null; Tail = null;

  // read data from stream
    for (x : input) {
        if (hm.containskey (x)) {
            Node xn = hm.get (x);
            Head = deleteNode (Head, xn);
            insert In Tail (Tail, xn);
        }
        else {
            if (hm.size() == m) {
                hm.remove (Head.data);
                Head = delete Head (Head);
            }
            Node nNode = new Node (x);
            hm.put (x, nNode);
            insert Last Node (Tail, nNode);
        }
    }
```

T.C = O(n)   S.C = O(n)

Question 4

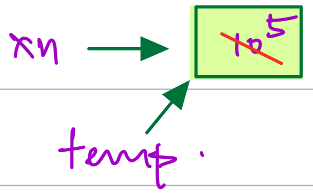
Given a linked list with next & random pointer create a deep copy of the linked list



Shallow copy

Node xn = new Node (10);

Node temp = xn;



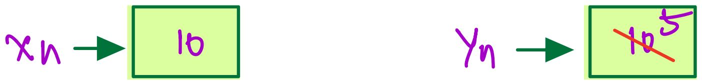
temp. data = 5

shallow copy is where you point to same address

Deep copy

Node xn = new Node (10);

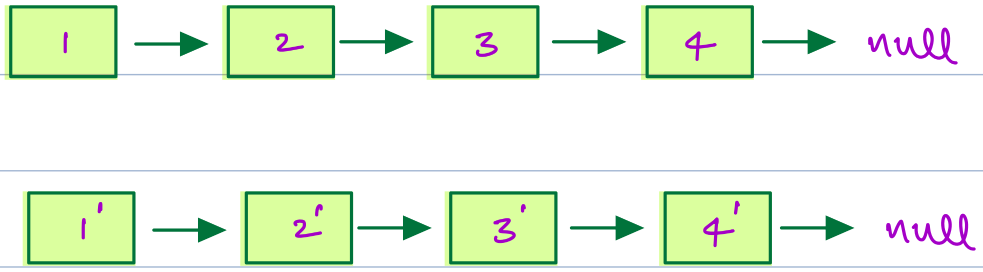
Node yn = new Node (x. data



Yn . data = 5

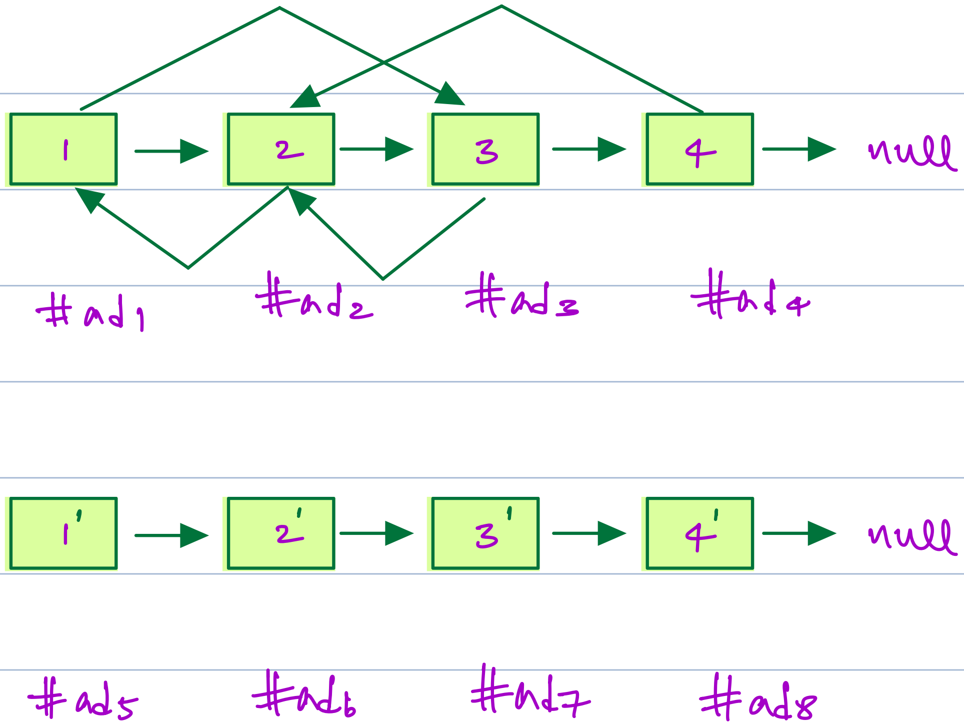
deep copy is where you point to different address .

Given a linked list can we create a deep copy



Steps to implement

1. check for edge conditions
2. Create your head outside the loop
3. Use a Hashmap to link <old Node : new Node>
4. Iterate till my original LL is not null and implement step 3



old addr	new addr
#ad1	#ad5
#ad2	#ad6
#ad3	#ad7
#ad4	#ad8

pseudo code

```
def    Create Deep Copy ( Node H1) {
|
|    Hashmap <Node, Node> hm = new
|                                  HM<>();
|    // check for edge cases
|    if (h1 == null) { return null }
|    // deep copy of head;
|    Node h2 = new Node (h1.data);
|    hm. put (h1 , h2);
|    Node t1 = h1;
|    Node t2 = h2;
```

```
        t1 = t1.next;
        while (t1 != null){
            Node t = new Node(t1.data);
            t2.next= t;
            t2 = t2.next;
            hm.put(t1, t);
            t1 = t1.next;
        }

        // create random mapping.
}
```