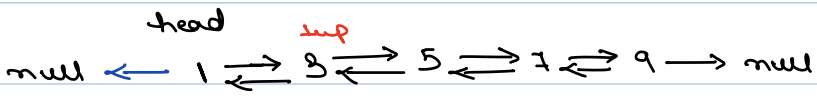
Today's content :-

- What is doubly linked list?
- How is doubly linked list different from singly linked list?
- 4 coding problems related to doubly linked list

← Doubly LL →

A doubly linked list is a type of data structure used in computer science and programming to store and organize a collection of elements, such as nodes. It is similar to a singly linked list but with an additional feature: each node in a doubly linked list contains pointers or references to both the next and the previous nodes in the list.




```
class Node {
    int data;
    Node next;
    Node prev;
}
```

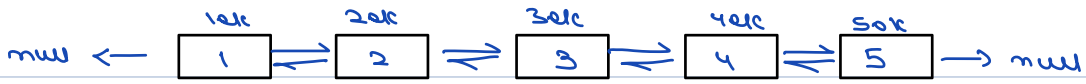
3

x = temp.next

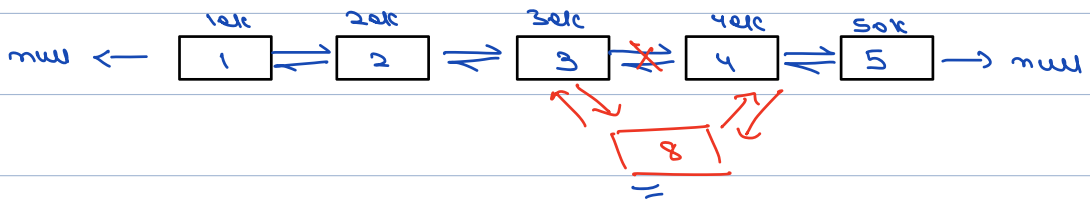
y = temp.prev

Ques

A doubly linked list is given. A node is to be inserted with data `X` at position `K`. The range of `K` is between 0 and `N` where `N` is the length of the doubly Linked list.



Ex → x = 8, k = 3



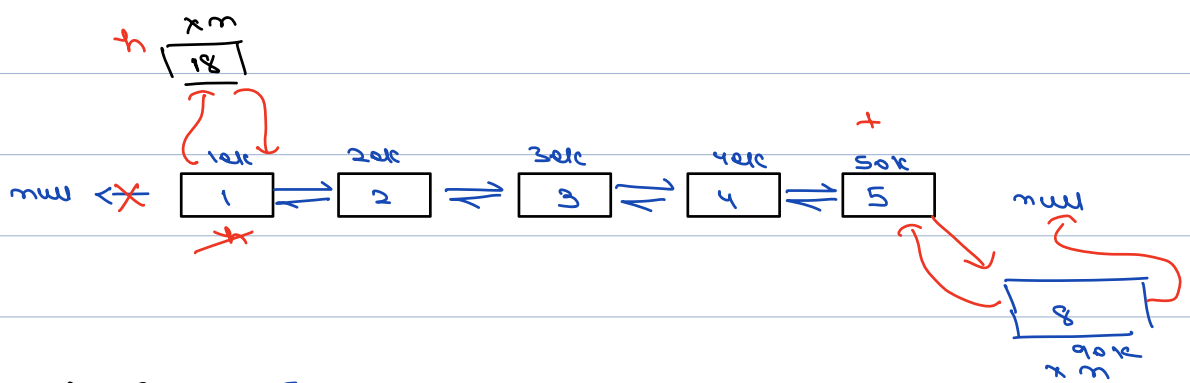
data = 8
next = 4ak
prev = 3ak

```
xn = new Node(x)
if (head == null) { return xn }

if (x == 0) {
    xn.next = head;
    head.prev = xn;
    head = xn;
    return head;
}
```

T.C → O(n)

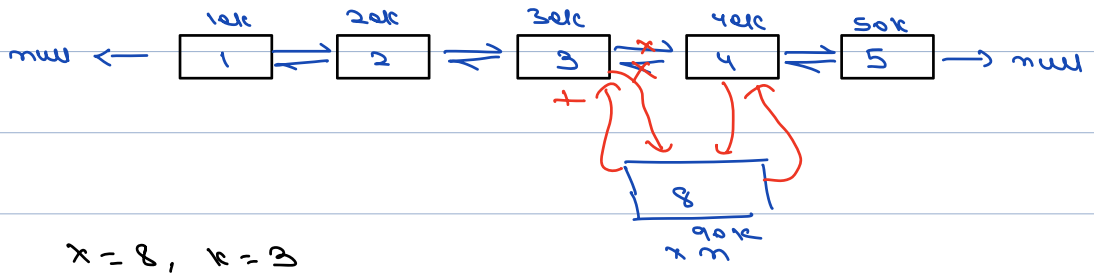
S.C → O(1)

```
temp = head;
for (i = 1; i < x; i++);
    temp = temp.next

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



x = 8, k = 5

(0 based)

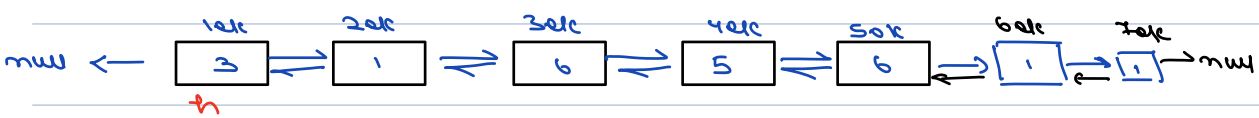


x = 8, k = 3

(0 based)

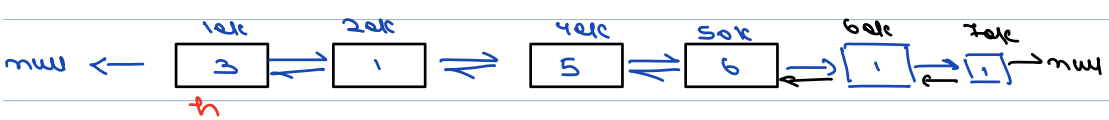
Ques

we have been given a doubly linked list of length N, we have to delete the first occurrence of data X from the given doubly linked list. If element X is not present, don't do anything.

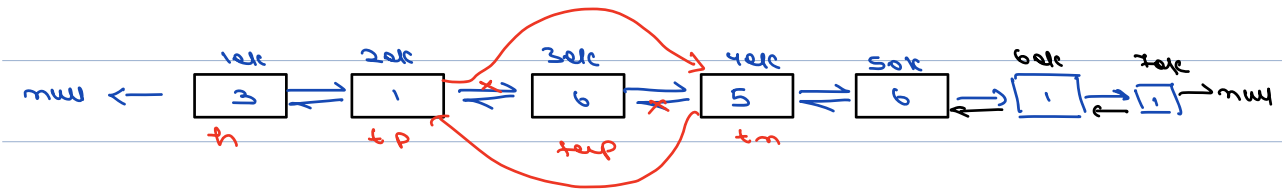


x = 6.

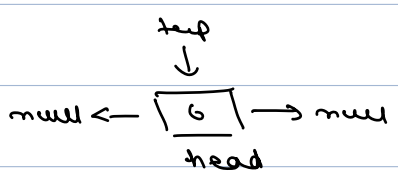
Ans.



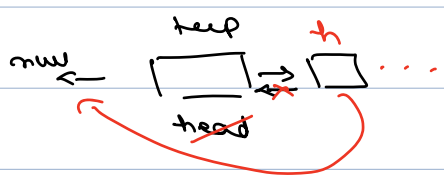
Soln :- x = 6 .



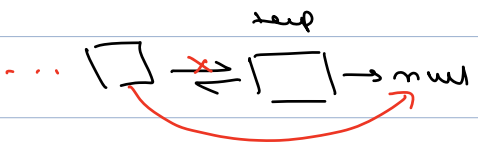

```
temp = head;
// Searching.
while (temp != null) {
    if (temp.data == x) {
        break;
    }
    temp = temp.next;
}
if (temp == null) {
    return head;
}
if (temp.prev == null && temp.next == null) {
    head = null;
    return head;
}
```




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



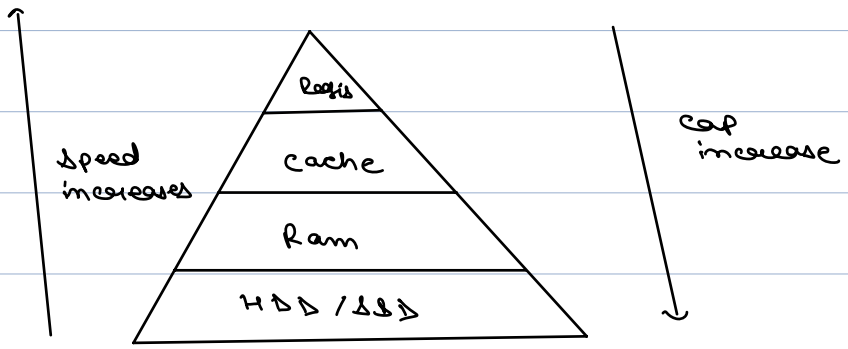
```
else if (temp.next == null) {
    temp.prev.next = null;
    return head;
}
else {
    tp = temp.prev;
    tn = temp.next
    tp.next = tn
    tn.prev = tp
    return head;
}
```



T.C → O(n)

S.C → O(1).

LRU Cache → least recently used.

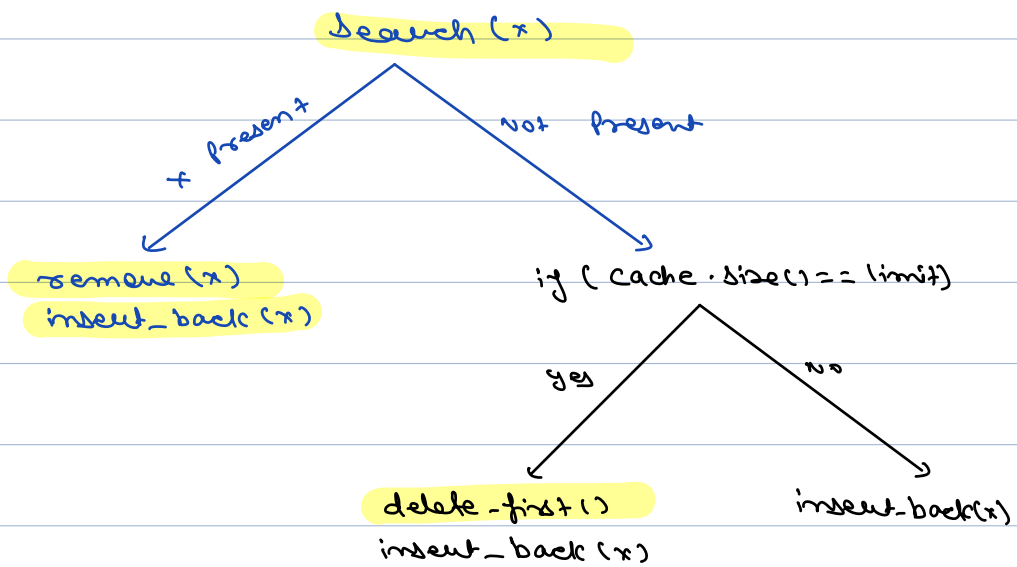


Data 7 3 2 9 6 10 14 2 15 10 8 11 10 19

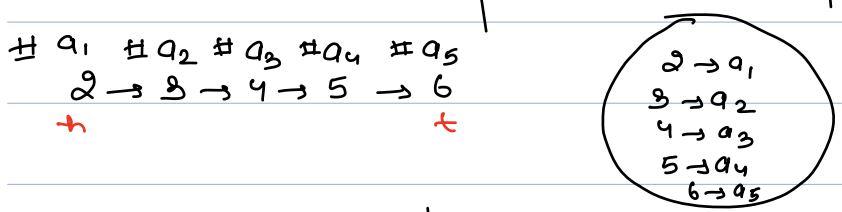
Cap = 5

~~7~~ ~~3~~ ~~2~~ ~~9~~ ~~6~~ ~~10~~ ~~14~~ ~~2~~ 15 ~~10~~ 8 11 10 19

Flow chart (x)



Operations :-	Dynamic Arr	LL
search (x)	O(n)	O(n)
remove (x)	O(n)	O(n)
insert_back (x)	O(1)	O(1) (if you maintain tail)
delete_first ()	O(n)	O(1)



LL + HashMap

Operations :-	Queue	LL + HashMap
search (x)	O(n)	O(1)
remove (x)	O(n)	O(n)
insert_back (x)	O(1)	O(1) (with tail)
delete_first ()	O(1)	O(1)

DLL + HashMap

Data 7 3 2 9 6 10 14 2 15 , 19 ,

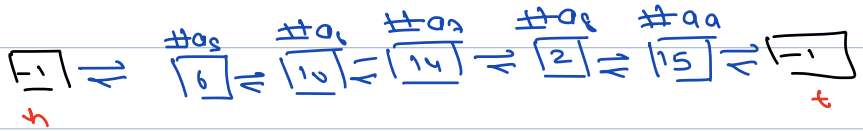
Cap = 5

~~7~~ ~~3~~ ~~2~~ ~~9~~ 6 10 14 2 15

HashMap

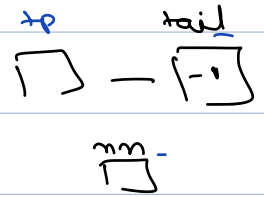
10 → a6
14 → a7
2 → a8
15 → a9
6 → a5

DLL

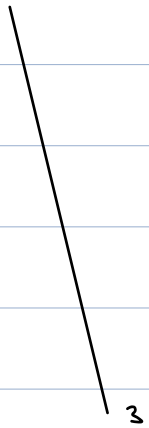



```
Node h = new Node(-1);
Node t = new Node(-1);
h.next = t
t.prev = h
HashMap<int, Node> hm;

- - - LRU (int x, int limit) {
    if (hm.search(x)) {
        Node t = hm[x];
        Delete Node(t);
```

```
        Node nn = new Node(x);
        insert_back(nn, tail);
        hm[x] = nn;
    }
    else {
        if(hm.size() == limit) {
            Node t = h.next;
            hm.delete(t.data);
            Delete Node(t);
        }
        Node nn = new Node(x);
        insert_back(nn, tail);
        hm.insert(x, nn);
    }
}

void DeleteNode(Node temp) {
    Node tp = temp.prev;
    Node tn = temp.next;
    tp.next = tn;
    tn.prev = tp;
}
```



```
void insert_back (node nn, node tail) {
    node tp = tail.prev;
    tp.next = nn;
    nn.prev = tp;
    nn.next = tail;
    tail.prev = nn;
}
```

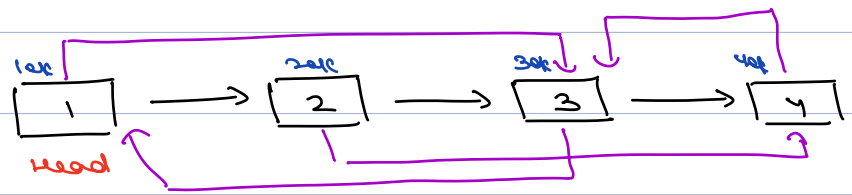


Ques Deep Copy of a LL.

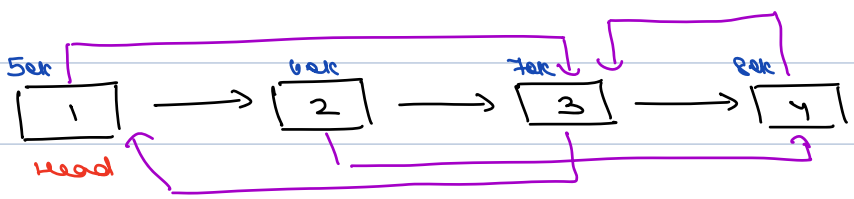
```
class node {
    int data;
    node next;
    node rand;
}
```

3

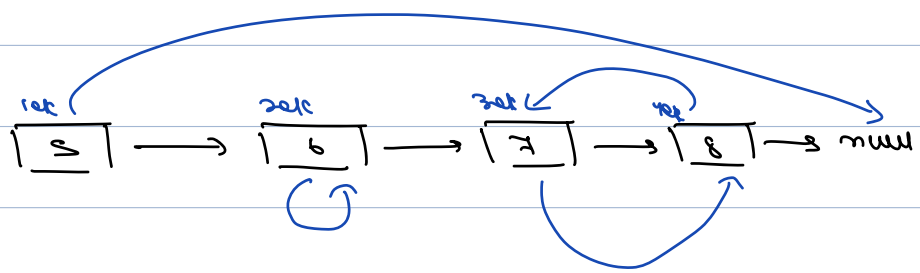
I/p

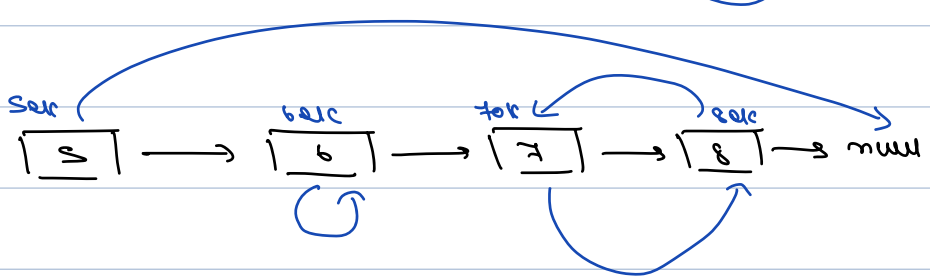


O/p

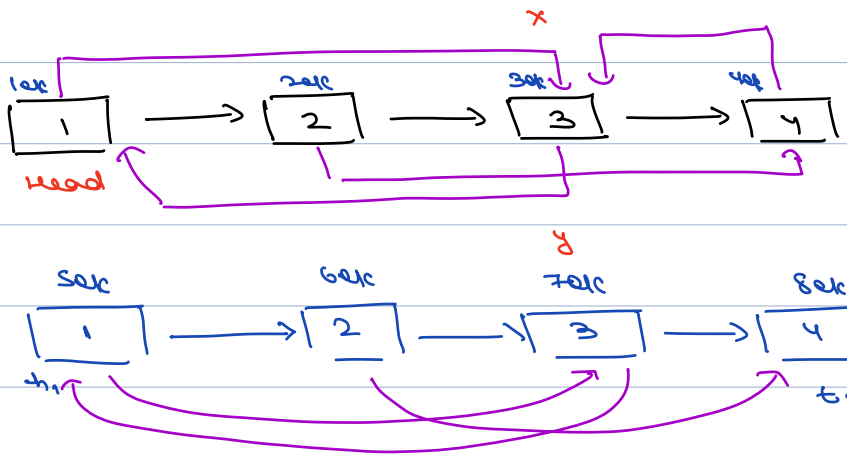


I/p





idea



while (x != null) {

temp = x.next;

y.next = hm.get(temp)

x = x.next;

y = y.next;

}

HashMap <Node, Node>

1 → 5

2 → 6

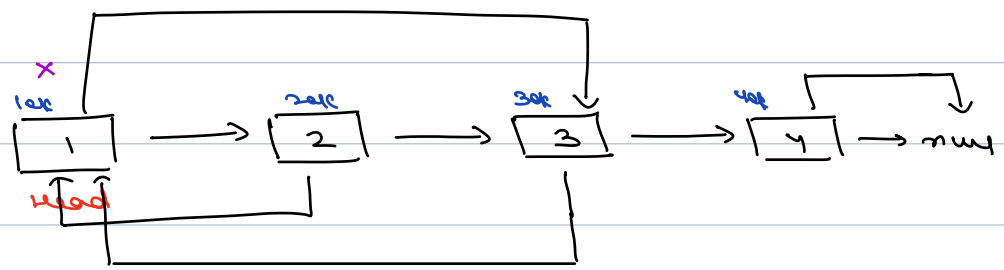
3 → 7

4 → 8

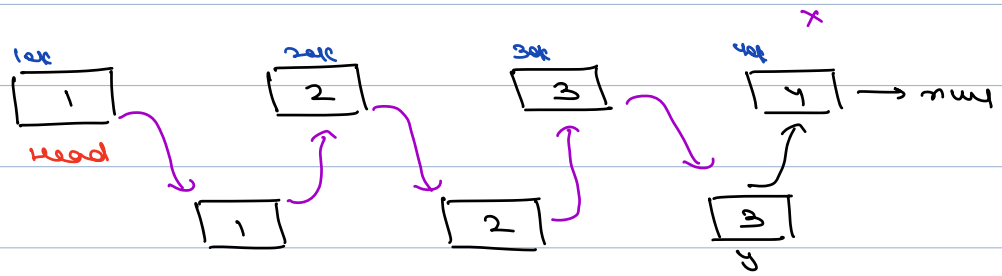
T.C → O(n)

S.C → O(n)

Constraint :- Do it in Constant Space :-



1)

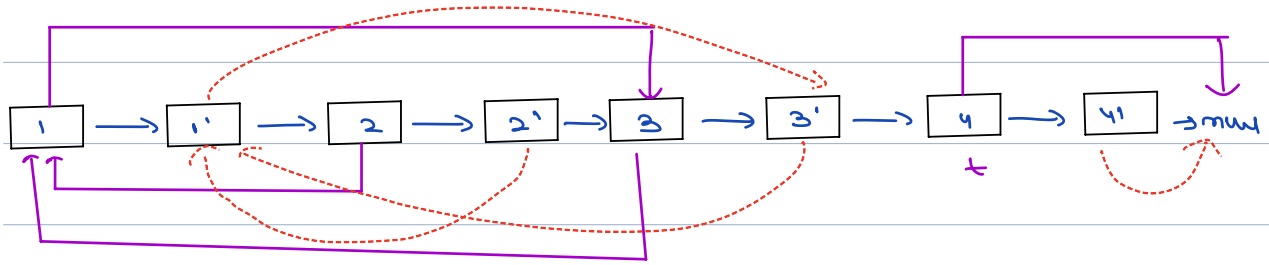


y = new Node(x.data) ✓

y.next = x.next; ✓

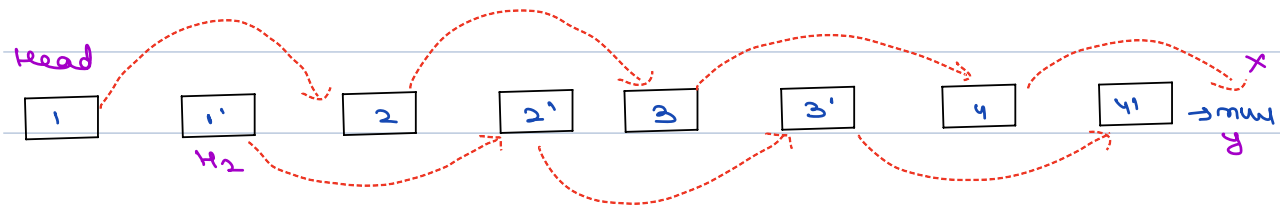
x.next = y ✓

x = x.next.next




```
temp = head';

while (temp != null) {
        if (temp.random == null) { temp.next.random = null }
        else { temp.next.random = temp.random.next }

        temp = temp.next.next
}
```

Step 3 detach




```
h2 = head.next
x = head';
y = head.next
while (x != null) {
        x.next = x.next.next
        if (y.next != null) {
                y.next = y.next.next;
        }
        x = x.next;
        y = y.next;
}

return h2';
```