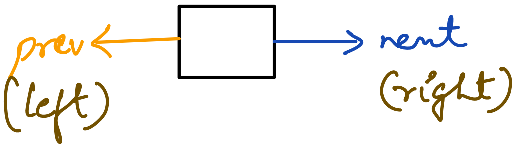
Welcome ☺

Always be extra careful about NPE

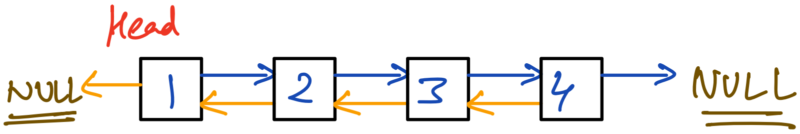
Agenda: DLL

2 ques^ns

Doubly Linked List DLL

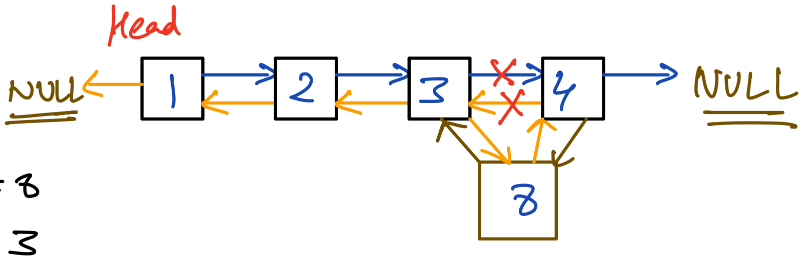


eg:



Q Given a DLL. Insert a node with data X at pos^n K

$0 \le K \le N$



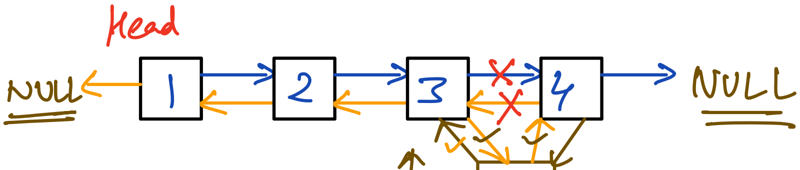
eg: x = 8

k = 3

Code

```
nn = newNode(8)
1) // Empty list
 if( head == NULL) return nn
2) // Update Head // K = 0
 if ( k==0)
 {
       nn.next = Head
       Head.prev = nn
       Head = nn
       return Head
 }
```

T.C o(1)

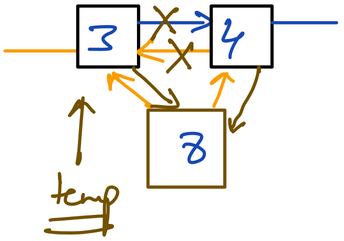


O(K)

```
temp = Head.
for ( i -> 1 to K-1) // traversal
{
    temp = temp.next
}
```

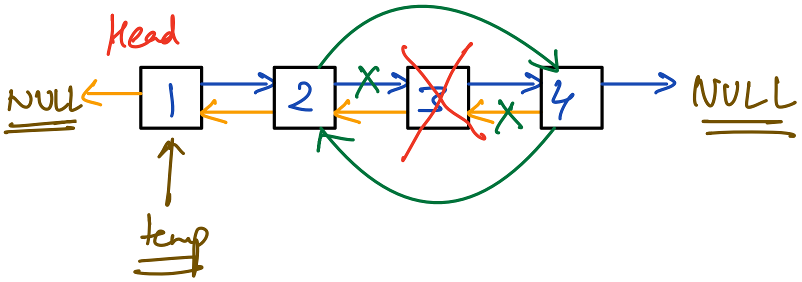
O(1)

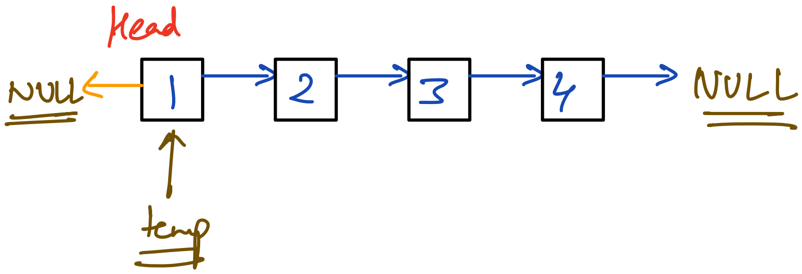
```
nn.next = temp.next
nn.prev = temp
if(!temp.next)     temp.next.prev = nn
temp.next = nn.
return Head ;
```



T.C -> O(K)
S.C -> O(1)

Q2 Delete the first occurrence of data X in DLL.
If not X not present -> no update.





Code

```
temp = Head
while( temp ! = NULL) // Searching for X
{
    if( temp.data = = X )
        break;
    temp = temp.next
}
```

T.C -> O(N)

```
1) if ( temp == NULL)   return Head. // No update

2) if ( temp.next == NULL && temp.prev == NULL) // single Node.
          return NULL

   elseif ( temp.prev == NULL)  // delete Head Node.
   {
          temp.next.prev = NULL
          Head = temp.next
   }
   else if ( temp.next == NULL) // delete Tail Node
   {
          temp.prev.next = NULL
   }
   else
   {
            temp.prev.next = temp.next
   O(1)     temp.next.prev = temp.prev
   }

   return Head.
```

T.C → O(N)
S.C → O(1)

T.C to delete → O(1)

LRU Cache

Q Given a running stream of integers and a fixed memory size M → O(M)

∀ intake ⇒ maintain most recent M elements

eg: 10 15 13 20 18 23 20 19 17 17 10

m=5

~~10~~ ~~15~~ ~~13~~
~~20~~ ~~18~~ 23
20 19 ~~17~~ 17 10

Vintake ⇒ 1) If x is not present

1) If memory is full → Delete least recent item

2) Insert X as most recent item.

2) If X is already present

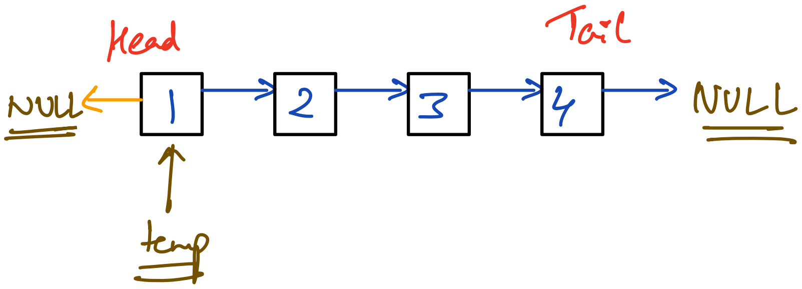
1) Remove X from its posⁿ

2) Insert X as most recent item.

1) Check → ~~Hashset~~ / Hashmap → (X, ~~Node of X~~ pointer to X)

2) Store elements in order of recency → ~~Array~~ / ~~Dynamic Arrays.~~ ~~L.L~~ / DLL

Code



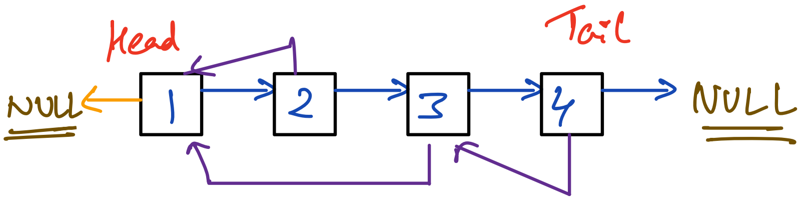
```
if ( hm.contains ( X ))
{
    // delete X from its posⁿ
    xn = hm.get ( X )
    deleteNode ( xn , head )
    // Insert it as last Node (MRU)
    Tail.next = xn
    xn.prev = Tail
    xn.next = NULL.
    Tail = Tail.next
}
else   // not present
{
    if ( hm.size( ) == M)   // full memory
    {
        hm.remove ( head.data)
        deleteNode ( head )
    }

    xn = newNode (X)
    hm.put ( X, xn)
    if ( head == NULL)
            head = tail = xn
    else // Insert as last node
    {
        Tail.next = xn
        xn.prev = Tail
        Tail = Tail.next
    }
}
```

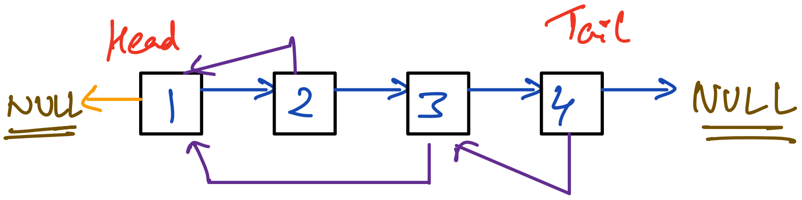
T.C per intake → $O(1)$

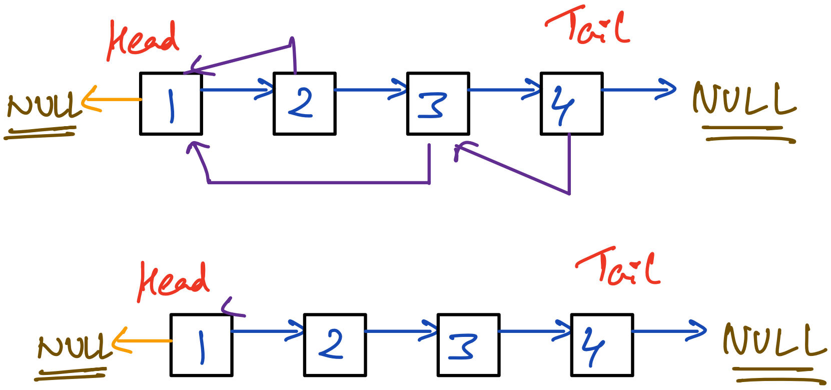
S.C → $O(2m)$

$\simeq O(m)$

Q Create a deep copy of DLL with random pointers.







Hash map

< old node , newNode >

< 1 , 1' >

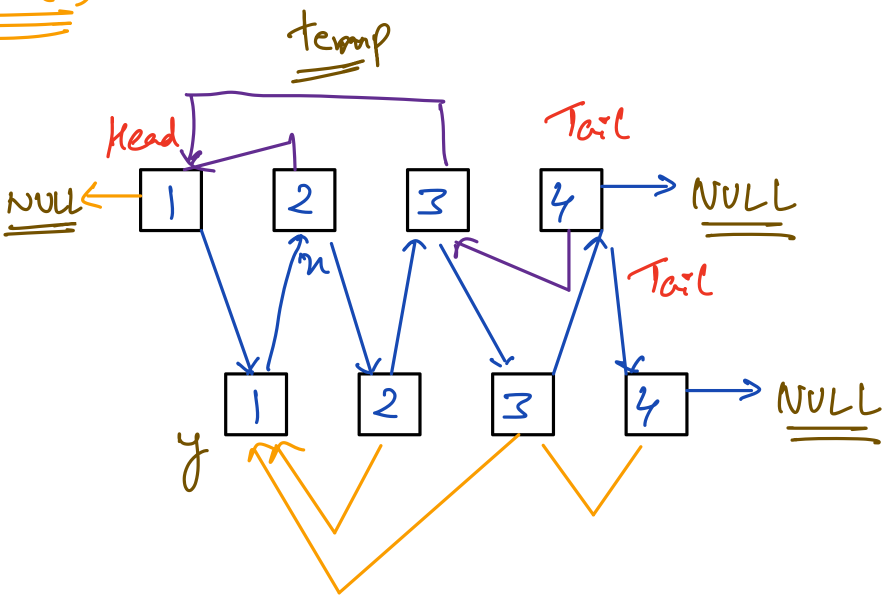
2 , 2'

3 , 3'

4 , 4'

S.C ⇒ O(N)

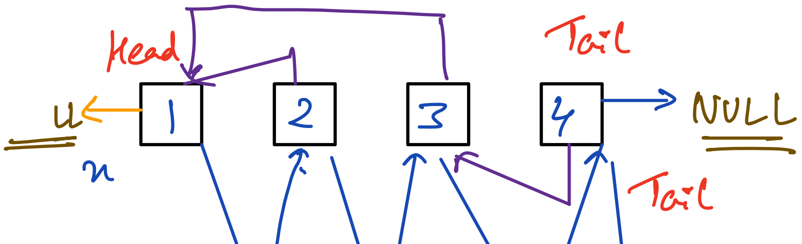
* S.C ⇒ O(1)



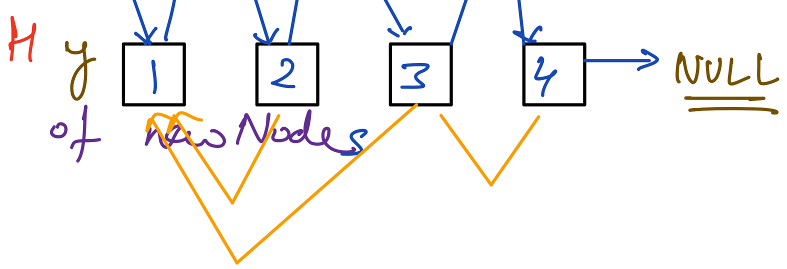
Code

1) Link two trees

```
while ( temp. != NULL)
{
    y = new Node ( temp .data)
    y.next = temp.next.
    temp.next = y
    temp = y.next
}
```



2) Populate random pointers of new Nodes



```
x = Head.
x = x.next ⟶ ignore this if x.random can never be NULL
while ( x != NULL)
{
    y = x.next
    y.random = x.random.next
    x = x.next.next
}
```

3) Soperate 2 LL

```
H = head.next
x = head
while ( x != NULL)
{
    y = x.next
    x.next = x.next.next
    if(!y.next)     y.next = y.next.next
    x = x.next
}
return H
```

T.C. → O(N)

S.C → O(1)