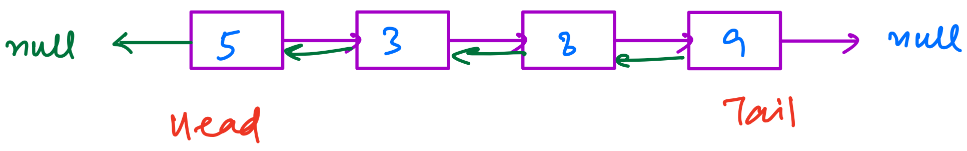
Linked List : Problems & Doubly Linked List

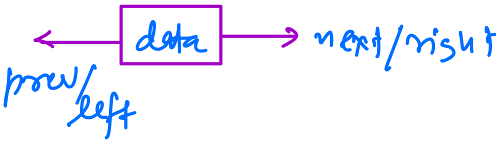
Agenda

- doubly linked list
- LRU cache
- Check if LL is palindrome

Doubly LL ?

It has prev & next pointers.





```
class Node {
    int data;
    Node next, prev;
    Node(x) {
        data = x
        next = prev = null
    }
}
```

Spotify playlist

Add song → Insert new song in playlist. If playlist is empty, it becomes "current song"

Play next song → move to next song & display its details

Play previous song → move to previous song & display its details

Current song → details of current song being played.

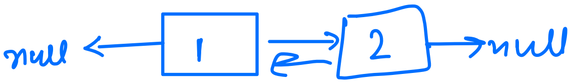
I/O :

Add song (Id : 1 , Name : " Yesterday Blues")

Add Song (Id : 2 , Name : " Imagine Dragons")

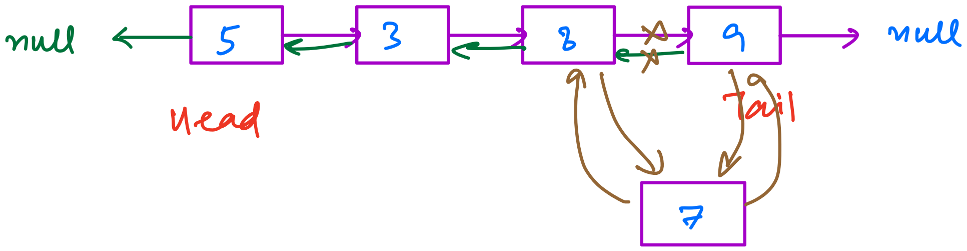
Play next song

⋮



Question

Insert node just before tail in doubly LL.



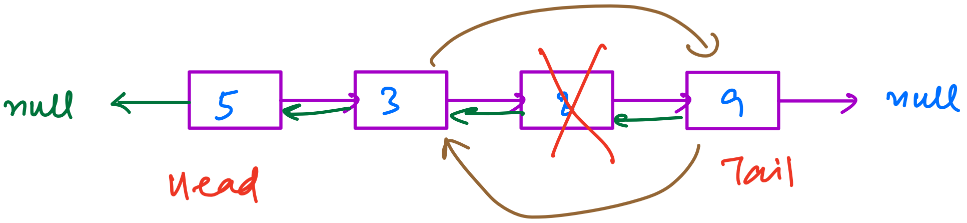
```
def insert_back( Node head, Node tail, Node newNode){

    newNode.next = tail
    new Node.prev = tail.prev
    tail.prev.next = new Node
    tail.prev = newNode

}
```

TC = O(1)

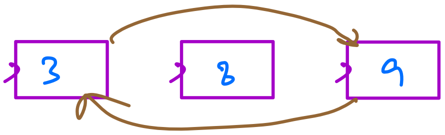
Question

Delete a given node from DLL.

1. Node reference is given
2. Given node will not be head/tail
3. DLL is not null

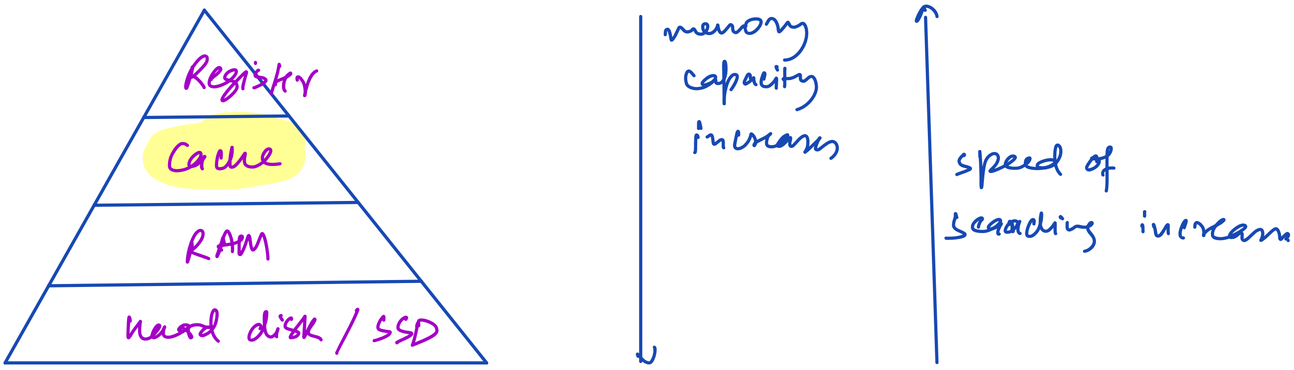


```
def remove ( Node x) {

    p = x.prev
    n = x.next
    p.next = n
    n.prev = p
    x.prev = x.next = null
    free (x)  // or garbage collected in Java

}
```



TC = O(1)

Memory Hierarchy



LRU cache : Least Recently Used

its principle is more recently accessed data will more likely to be accessed in future.

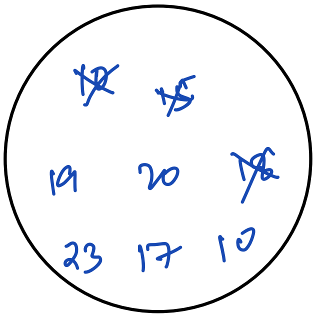
Question : LRU cache

Given a running stream of integers & a fixed memory of size M.

Maintain the latest M elements in memory. In case memory is full, delete the least recent item.

eg → 10✓ 15✓ 19✓ 20✓ 18✓ 23✓ 20✓ 19✓ 17✓ 17✓ 10✓

M=5



Cache

old → new

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

Once memory is full

if intake x

x is not present

1. Delete least recent item.

2. Insert x as most recent item.

x is present

1. Delete x from its position

2. Insert x as most recent item.

Requirements :

<data, location of data>

1. Search ∀ intake x → ~~Hashset~~ / Hashmap

2. Maintain order of recency → ~~Array~~ , ~~Stack~~ , ~~Queue~~ ,

Linked list

delete curr node ⇒

doubly LL

Code

```
Hashmap < Integer, Node >  hm = new Hashmap<>();

Head = Tail = NULL

for ( ∀ input : X ) {

    if( hm. contains (X) ) {

        temp = hm.get (X)   // node in DLL   → O(1)

        Head = deleteNode ( head, temp)   → O(1)

        Tail = insert Last Node ( Tail, temp) → O(1)

    }

    else {
```

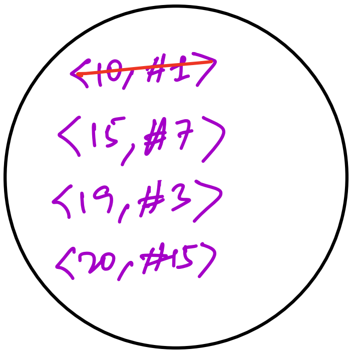
```
    if ( hm.size() == m ) {   // evict least recently used
        hm.remove (head.data) → O(1)
        head = deleteHead ( head ) → O(1)
    }
    newNode = new Node (X)
    hm.put ( X , newNode ) → O(1)
    Tail = insertLastNode ( Tail, newNode ) → O(1)
}
```

time complexity → O(1) per operation

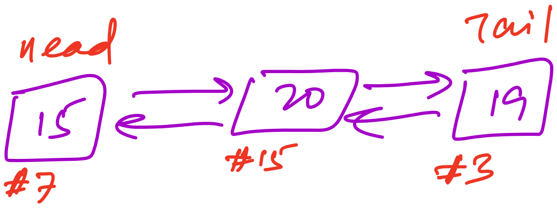
eg → 10✓ 15✓ 19✓ 20✓ 19✓ 23 20 19 17 17 10

m=3

head = Tail = NULL



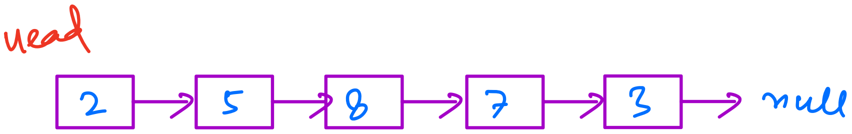
Hash map



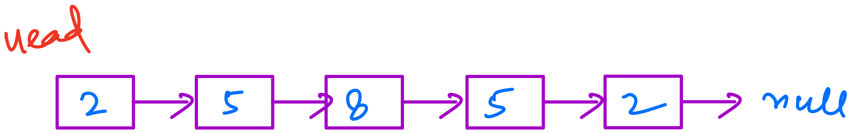
Question

Given a LL, check if it is palindrome.

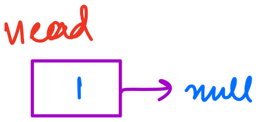
eg



ans = false



ans = true



ans = true

Idea 1 :

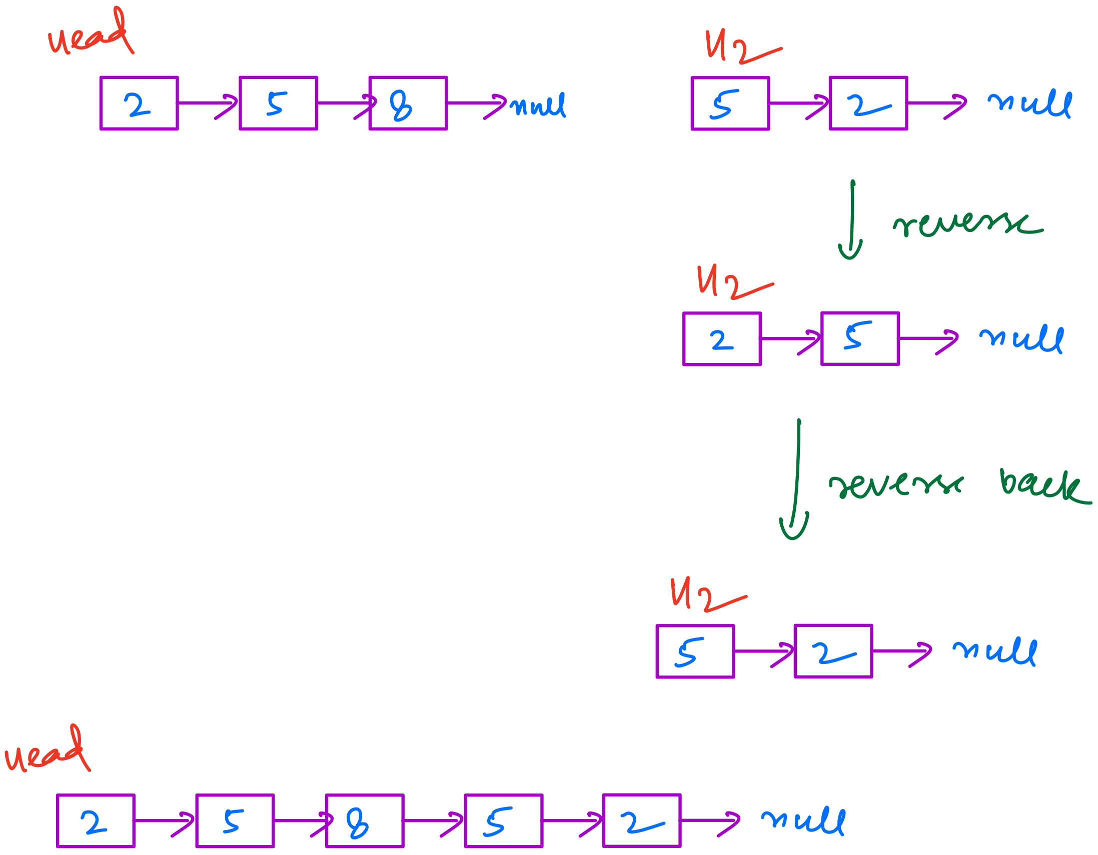
1. Create a copy of LL .
2. Reverse the copy
3. Compare both one by one

TC = O(N) SC = O(N)

Idea 2 :

1. Find middle element of LL → slow-fast pointer approach
2. Reverse the second half of LL
3. Compare first half & second half
4. Reverse back to original.

TC = O(N) SC = O(1)



head
2
5
8
null
h2
5
2
null
reverse
h2
2
5
null
reverse back
h2
5
2
null
head
2
5
8
5
2
null