Linked List 3

~~~~~~~~

Agenda

~~~~~~~~

1. Revision Quiz
2. Introduction to Doubly LinkedList
3. Real Life problem
4. Operations on DLL :
    * Insert Node Before Tail
    * Delete Given Node from LinkedList
5. LRU cache
6. Clone a LinkedList

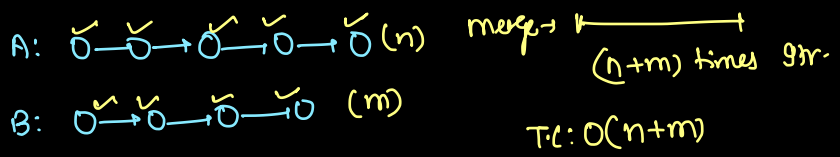
Hello Everyone
Very Special Good Evening
to all of you 😊😊😊
We will start discussion
from 09:06 PM

~~~~~~~~~~~~~

Revision Quiz

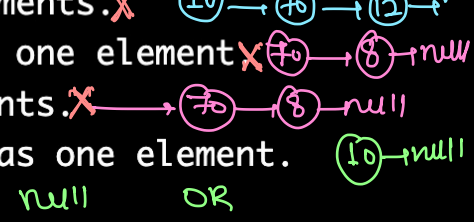
~~~~~~~~~~~~~

Quiz 1:
Can we do Binary Search in a sorted Linked List?
[ ] Yes
[✓] No
[ ] Sometimes only

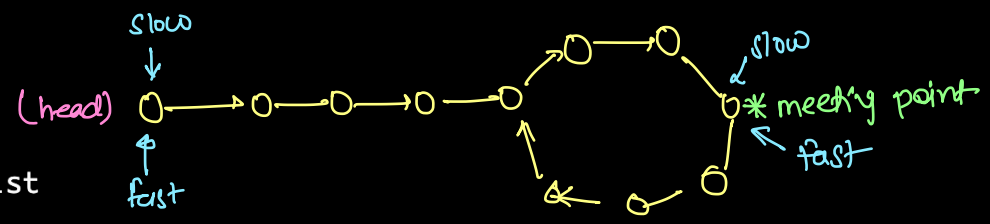


Quiz 2:
What is the time complexity to merge two sorted linked lists into a single sorted linked list?
[✓] O(n + m)
[ ] O(n log(n))
[ ] O(n * m)
[ ] O(m log(m))

Quiz 3:
What is the base case in the merge sort algorithm for linked lists?
[ ] When the list has three elements.
[ ] When the list has more than one element.
[ ] When the list has two elements.
[✓] When the list is empty or has one element.



Quiz 4:
In the problem of detecting a cycle in a linked list, what do the slow and fast pointers initially point to?
[ ] Tail of the list
[x] Head of the list
[ ] Middle of the list
[ ] Random nodes in the list



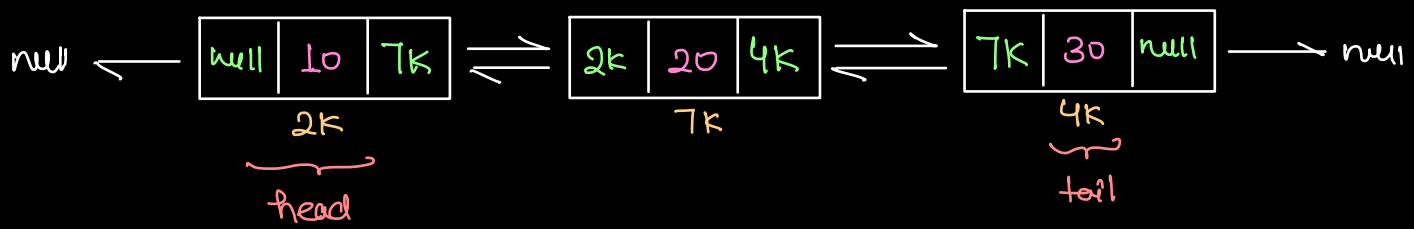
Quiz 5:
How is the starting point of the cycle found once a cycle is detected in the linked list?
[x] Reset the slow pointer to head and move both pointers one step at a time.
[ ] Continue moving fast pointer until it reaches the head. ✗
[ ] Move slow pointer two steps at a time. ✗
[ ] Move fast pointer to the start of the cycle. ✗

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Introduction to Doubly LinkedList

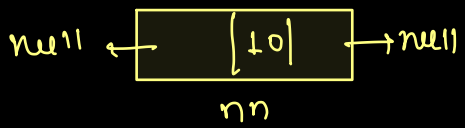
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

What is Doubly LinkedList?




```
class Node {
    int data;
    Node prev;
    Node next;
    public Node(int data) {
        this.data = data;
        this.next = this.prev = null;
    }
}
```

```
Node nn = new Node(10);
```



~~~~~~~~~~~~~~~~~~

Real Life problem : Spotify's Music Manager

~~~~~~~~~~~~~~~~~~

Scenerio

Spotify wants to enhance its user experience by allowing users to navigate through their music playlist seamlessly using "next" and "previous" song functionalities.

Problem

You are tasked to implement this feature using a doubly linked list where each node represents a song in the playlist.
The system should support the following operations:

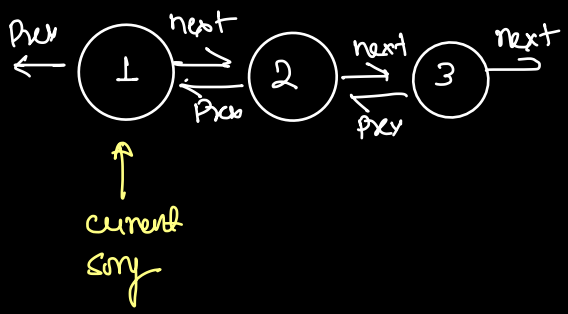
* Add Song: Insert a new song into the playlist. If the playlist is currently empty, this song becomes the "Current song".
* Play Next Song: Move to the next song in the playlist and display its details.
* Play Previous Song: Move to the previous song in the playlist and display its details.
* Current Song: Display the details of the current song being played.

Constraints:

Each song is uniquely identified by its song ID.
The playlist starts empty, and the first song added becomes the current song.
All operations are valid within the current state of the playlist (i.e., there won't be a request to play the next song if there is no next song, and no request to play the previous song if there is no previous song).

Input:

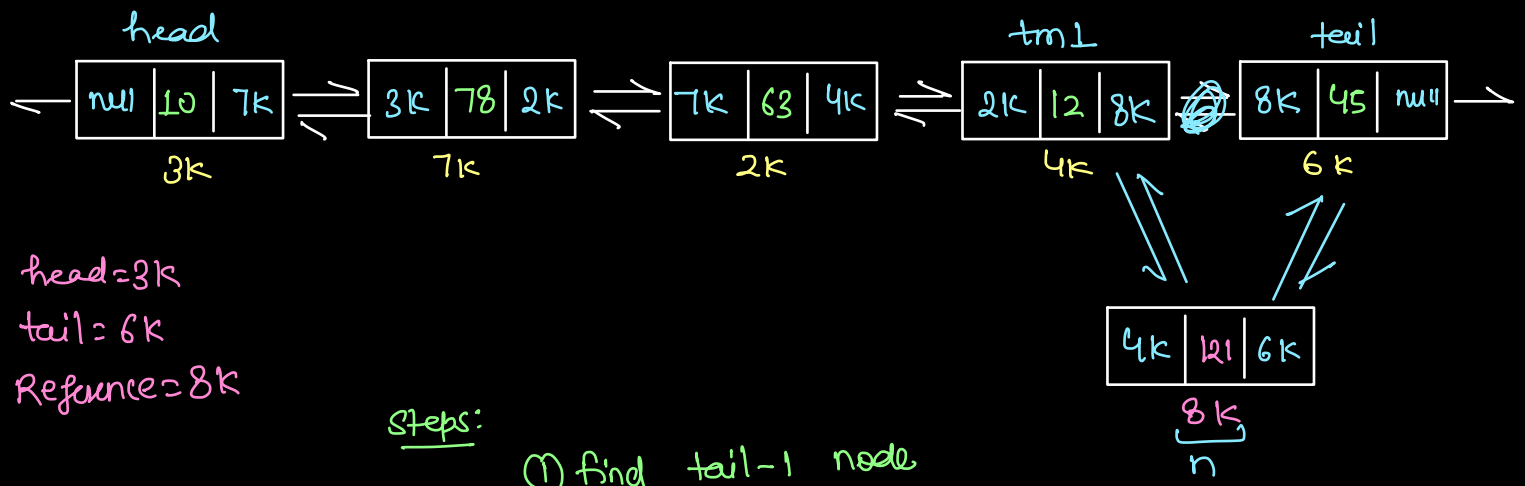
Add Song (ID: 1, Name: "Yesterday Blues")
Add Song (ID: 2, Name: "Imagine Dragons")
Play Next Song
Current Song Imagine Dragons.
Add Song (ID: 3, Name: "Hotel California")
Play Next Song
Current Song Hotel California
Play Previous Song
Play Previous Song
Current Song Yesterday Blues



Insert Node Before Tail

Given head and tail of DLL and reference of a node. Add that node just before the tail of Doubly LinkedList.

Note : The node whose reference is given is not already present in DLL.



head = 3K
tail = 6K
Reference = 8K

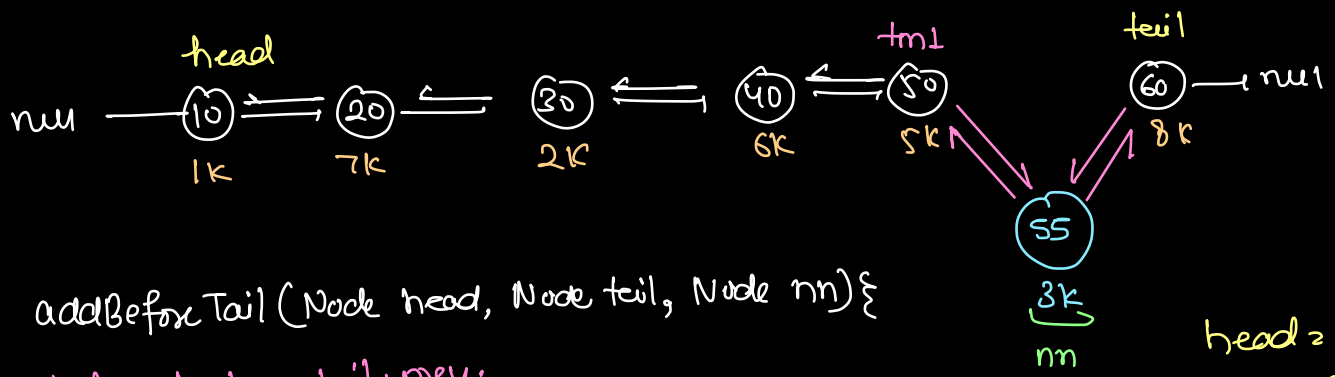
Steps:

① find tail-1 node
Node tm1 = tail.prev;

② connect tm1, tail & that node

n.next = tail
tail.prev = n;

tm1.next = n;
n.prev = tm1;




```
void addBeforeTail(Node head, Node tail, Node nn){
    Node tm1 = tail.prev;

    nn.next = tail;
    tail.prev = nn;

    tm1.next = nn;
    nn.prev = tm1;
}
```

T.C: O(1)
S.C: O(1)

head = 1K
tail = 8K
nn = 3K

Pointers affected → 4 points affect

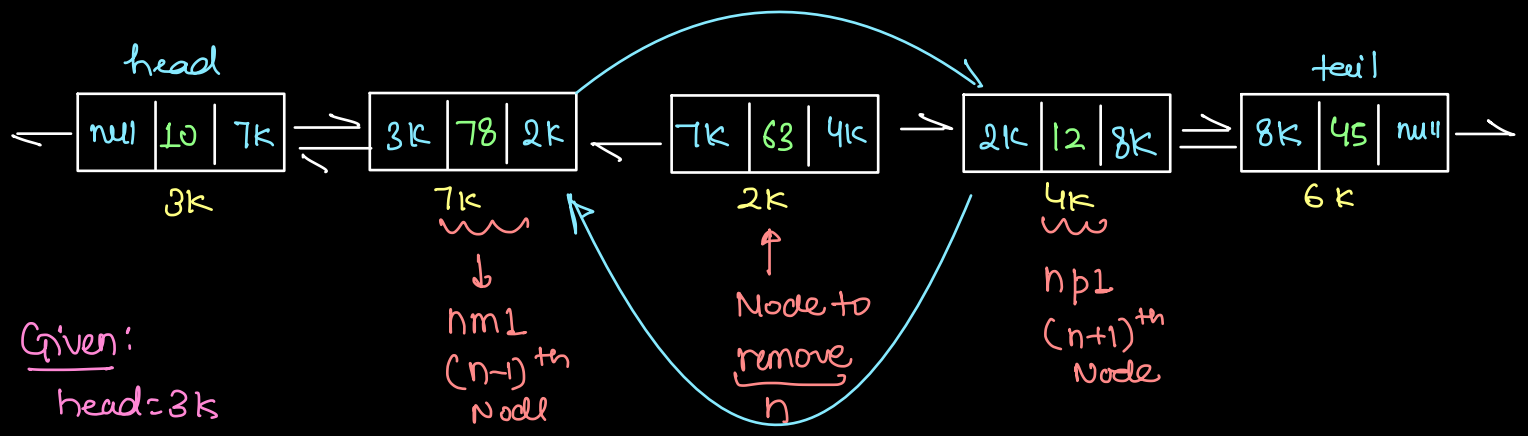
* next of tm1
* prev of nn
* next of nn
* prev of tail

Delete Given Node from LinkedList

Given head of DLL and reference of a Node, remove this node from DLL.

Note:

* Node with given reference is already present in DLL.
* Given node will definitely not be the first or last node.



Given:
head = 3K
rm = 2K

```
Node nm1 = n.prev;
Node npl = n.next;
//make link b/w nm1 & npl
nm1.next = npl;
npl.prev = nm1;
```

```
void removeNode(Node head, Node n){
    Node nm1 = n.prev;
    Node npl = n.next;

    nm1.next = npl;
    npl.prev = nm1;
    //for languages in which we have to delete memory manually.
    n.next = n.prev = null;
    free(n); ——→ to delete that memory
}
```

T.C: O(1)
S.C: O(1)

~~~~~~~~~~~~~~~~~~~~~~~~~~

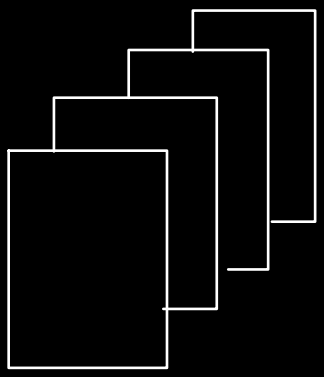
Introduction to LRU cache

~~~~~~~~~~~~~~~~~~~~~~~~~~

What is LRU Cache and how it's works?
Let us understand this with some examples.

LRU → least Recently Used

Click on Recent:



Capacity = 4 apps

Recent App

Scenario which we have to implement

10:25 - 10:35 PM

Breaks

~~~~~~~~~~~~

LRU Cache

~~~~~~~~~~~~

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

* get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
* set(key, value) - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least recently used item before inserting the new item.

The LRUCache will be initialized with an integer corresponding to its capacity.
Capacity indicates the maximum number of unique keys it can hold at a time.
Definition of "least recently used" : An access to an item is defined as a get or a set operation of the item. "Least recently used" item is the one with the oldest access time.

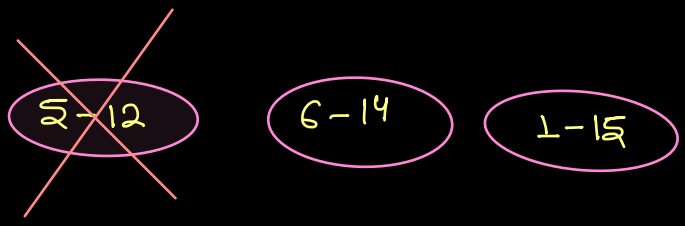
Input:

capacity = 2
set(1, 10) (key, value)
set(5, 12)
get(5) → 12
get(1) → 10
get(10) → -1
set(6, 14)
set(1, 15)
get(5) → -1

Cap = 2

for understanding
Key ⇒ Apps Name
value ⇒ task we are performing in that app.

front side



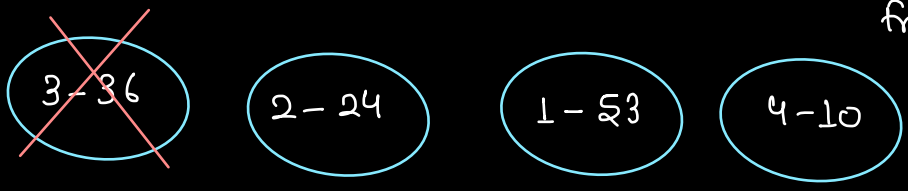
Example2:

capacity = 3
set(1, 13)
set(2, 24)
set(1, 53)
set(3, 36)
get(2) → 24
get(1) → 53
set(4, 10)
get(3) → -1

Cap = 3

Key ≡ Apps Name
value ≡ Action OR task

front side



set(key, value) →
- If key is not available, making a new key-value pair and add it in LRU.
- If key is already available, then we have to make update in value and move it in front.

get(key) →
- If key is available, move it in front & return value associated with it.
- If key is not available, return -1.

Data structures which requires to implement LRU cache:

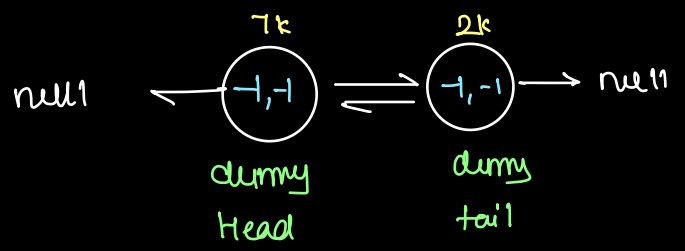
* Doubly LinkedList
* HashMap<Integer, Node> map

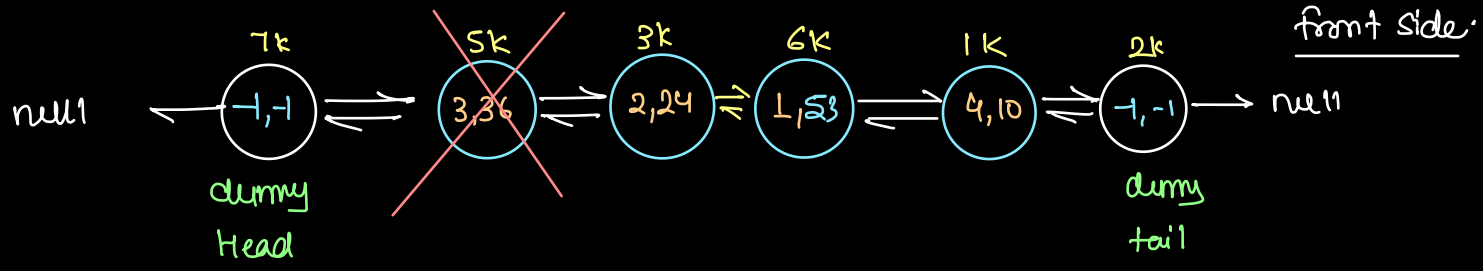
Structure of DLL:

```
class Node {
    int key;
    int value;
    Node next;
    Node prev;
    Node(int key, int value){
        this.key = key;
        this.value = value;
    }
}
```

To avoid multiple checks:

Initial your DLL with dummy head & dummy tail





- capacity = 3
- set(1, 13) → create nn → add Before tail
- set(2,24) → create nn → add Before tail
- set(1,53) → if key available → Yes → Remove 6k, add Before Tail, update the value of 6k
- set(3,36) → create nn → add Before Tail
- get(2) → if key available → Remove 3k, add Before Tail, Return value of 3k (24)
- get(1) → " " → " 6k (53)
- set(4,10) → create nn → add Before tail
- get(3) → if key available → No → return (-1)

Hashmap

Key vs. Node

1 → 6k

2 → 3k

~~3 → 5k~~

4 → 1k

Code Structure:

① Node class

→ key → next → constructor

→ value → prev

② HashMap < Integer, Node > map

get(key)

↓

if key is not present.

return -1;

if key is present

1. get that node

Node n = map.get(key);

2. make it most recent used

removeNode(n);

addBeforeTail(n);

3. Return value

return n.value;

Set(key, value)

↓

if key not present

// 1. create a nn

Node nn = new Node(key, value);

// 2. add it Before tail

addBeforTail(nn);

// 3. add it in map

map.put(key, nn);

// 4. Check the cap.

if(map.size() > cap){

// Remove LRU

Node n = head.next;

removeNode(n);

map.remove(n.key);

}

if key is present

1. Get the node

Node n = map.get(key);

2. Update the updt

n.value = value;

3. make it most recent used

removeNode(n);

addBeforTail(n);

~~~~~~~~~~~~~~~~~~

Clone a LinkedList (Next Session)

~~~~~~~~~~~~~~~~~~

Given a linked list where each node has a next pointer and a random pointer. Create a clone of this linked list and return the head of the cloned list. Ensure that the random pointers in the cloned list have the same connections as in the original list.

Agenda:

- Implementation of LL
  - Using Head
  - Using OOPS (inbuilt class)
- Clone a LL
- Realism of two sorted array