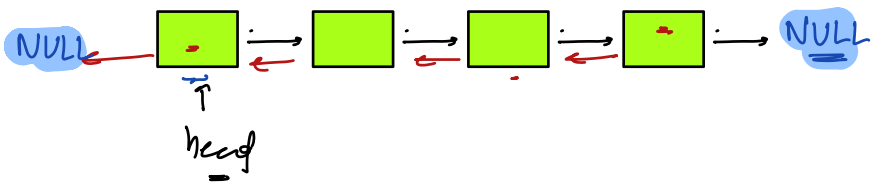
Today's Content

- Double Linked List Basics
- LRU Cache
- Clone LinkedList

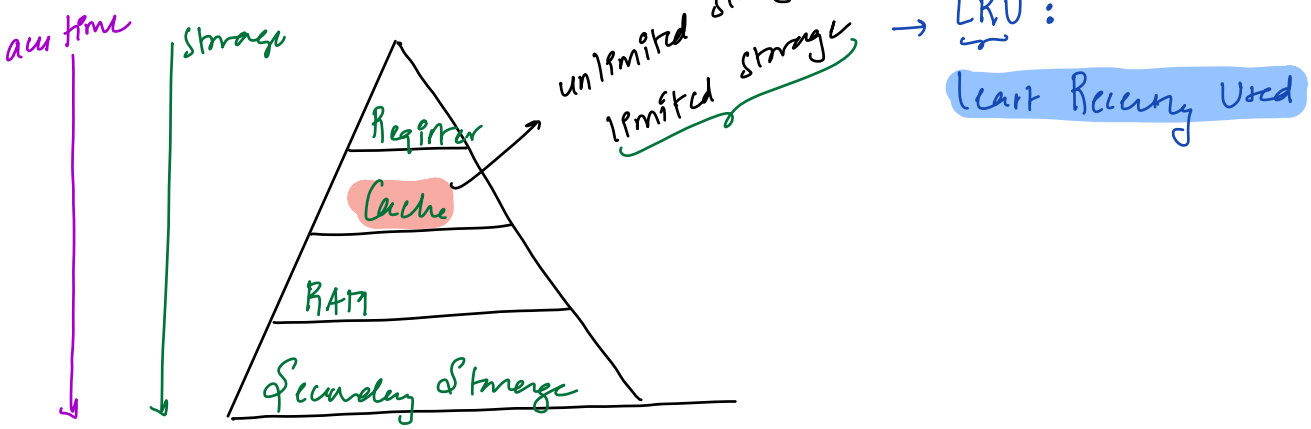
Double linked list :




```
class Node {
    int data;
    Node next;
    Node prev;
    Node(n) {
        data = n
        next = NULL
        prev = NULL
    }
}
```

LRU Cache

→ Memory Hierarchy

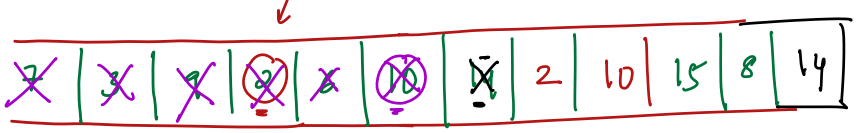


→ LRU :

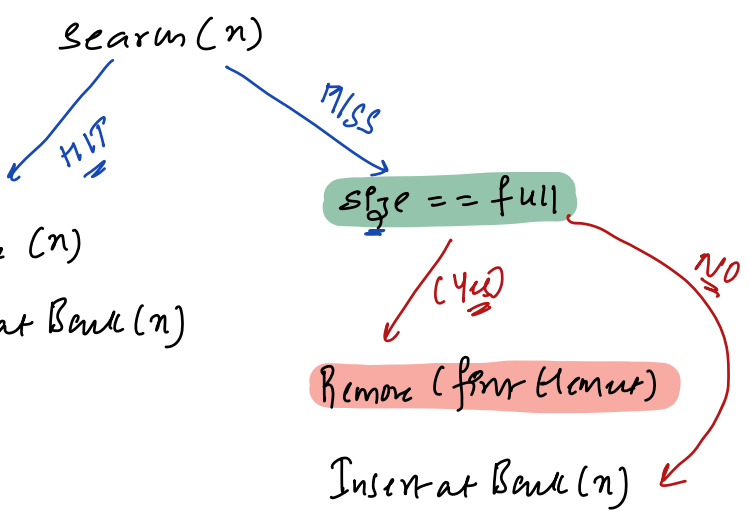
Least Recently Used

Data : 7 3 9 2 6 10 14 2 10 15 8 14

memory size : 5



flow chart

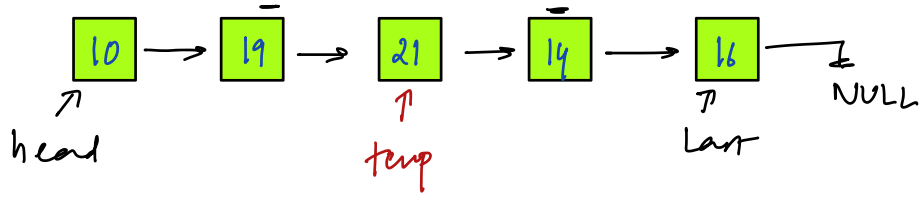


// To get size : (keep a count variable)

	arrays	SLL	SLL + HashMap <Int, Node>
search (x)	: O(N)	: O(N)	: O(1) : using hashmap
Remove (x)	: O(N)	: O(N)	: :
Insert at Back (x)	: O(1)	: O(1)	: O(1) : Insert at Begin

If we keep a last pointer

Ex:



Hashmap

<Key, Address>

10, ref(10)

19, ref(19)

21, ref(21)

14, ref(14)

16, ref(16)

→ x = 21

21 is present

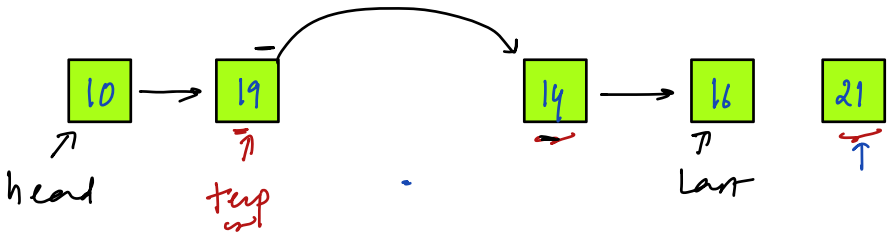
Node temp = hm[x]

temp = ref(21)

Issue: Re-arranging previous node link is not possible.

Idea 2: SLL + hashmap < int, Node >

address of prev



Hashmap

< Key, Addrs > (Addrs of prev node)

10, NULL

19, ref(10)

21, ref(19)] 21, ref(16)

14, ref(21)] 14, ref(19) ✓

16, ref(14)

→ x = 21

↳ 21 is present

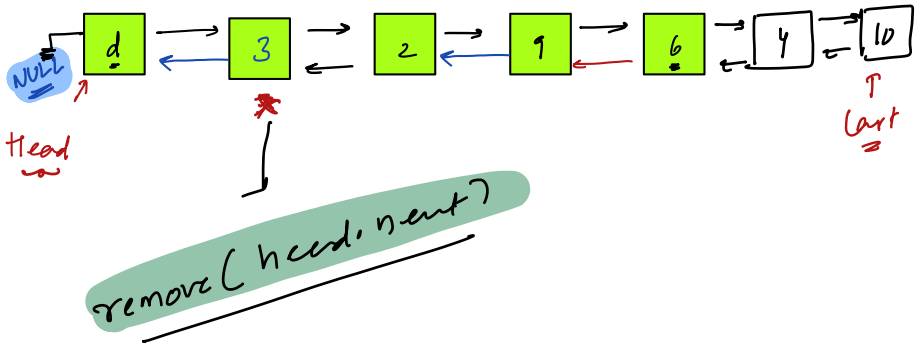
Node temp = hm[x]

Idea: DLL + Hashmap<int, Node>

↳ address of current Node

capacity = size = 5

3 6 2 9 6 6 4 10



Hashmap

remove from hashmap

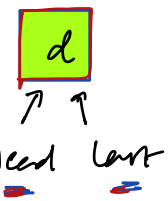
key	Node address
3,	ref(3)
6,	ref(6)
2,	ref(2)
9,	ref(9)
4,	ref(4)
10,	ref(10)

Note: If Node we want to delete is already at last do don't even do anything.

```
InsertatBack(Node T)
    last.next = T
    T.prev = last
    last = T
}
```

```
remove(Node T)
    T.prev.next = T.next ✓
    T.next.prev = T.prev
}
InsertatBack( ){
}
```

// Hashmap < Int, Node > hm



Note: We are not taking dummy into consideration.

→ Data x is coming :

x is in hm

MISS

Inserting <k, v> in hm

Node T = new Node(x)

hm.insert({x, T})

HIT

Node T = hm[x]

If T.next == NULL

Else

x is last node

Do nothing

remove(T)

InsertatBack(T)

If hm.size() > Capacity

True

(remove <k, v> in hm)

hm.erase(head.next.data)

remove(head.next)

InsertatBack(T)

False

If false only Insert at Back

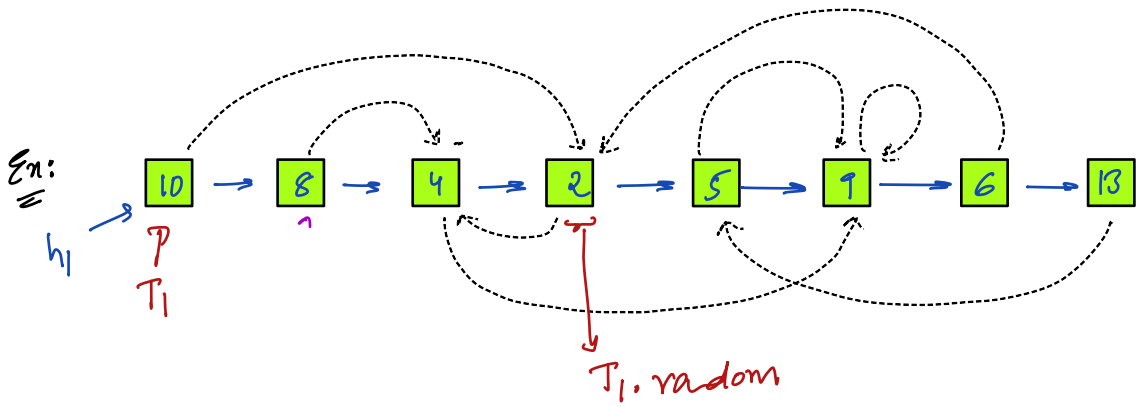
10:40pm

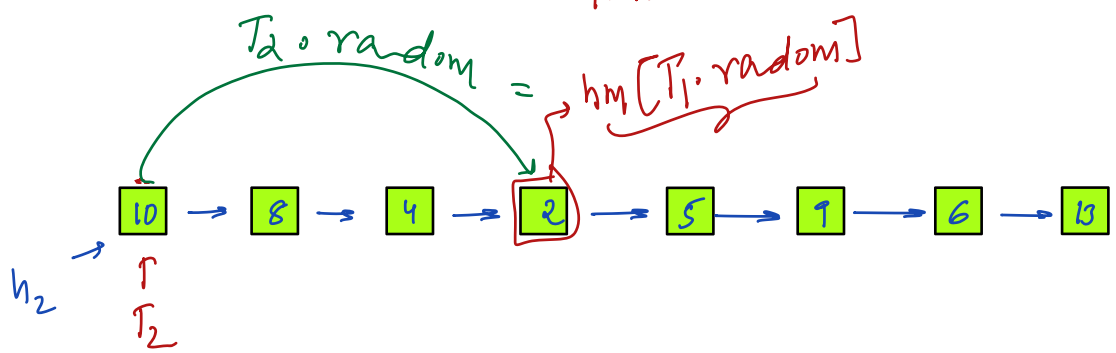
Ex: D=D=D=D

Clone Linked List

Obs: data can repeat in LinkedList

```
class Node {
    int data;
    Node next;   -> (points to next node)
    Node random; -> (Any where in LinkedList), random is not NULL
}
```





Idea:

1) Create a new linked h_2 with next pointer filled } TC: O(N)

2) HashMap<Node, Node> hm

Node $T_1 = h_1$, $T_2 = h_2$

while (T_1 != NULL && T_2 != NULL)

{

hm [T_1] = T_2

$T_1 = T_1$.next

$T_2 = T_2$.next

}

} O(N)

3) $T_1 = h_1$, $T_2 = h_2$

while (T_1 != NULL && T_2 != NULL) {

T_2.random = hm [T_1.random]

$T_1 = T_1$.next, $T_2 = T_2$.next

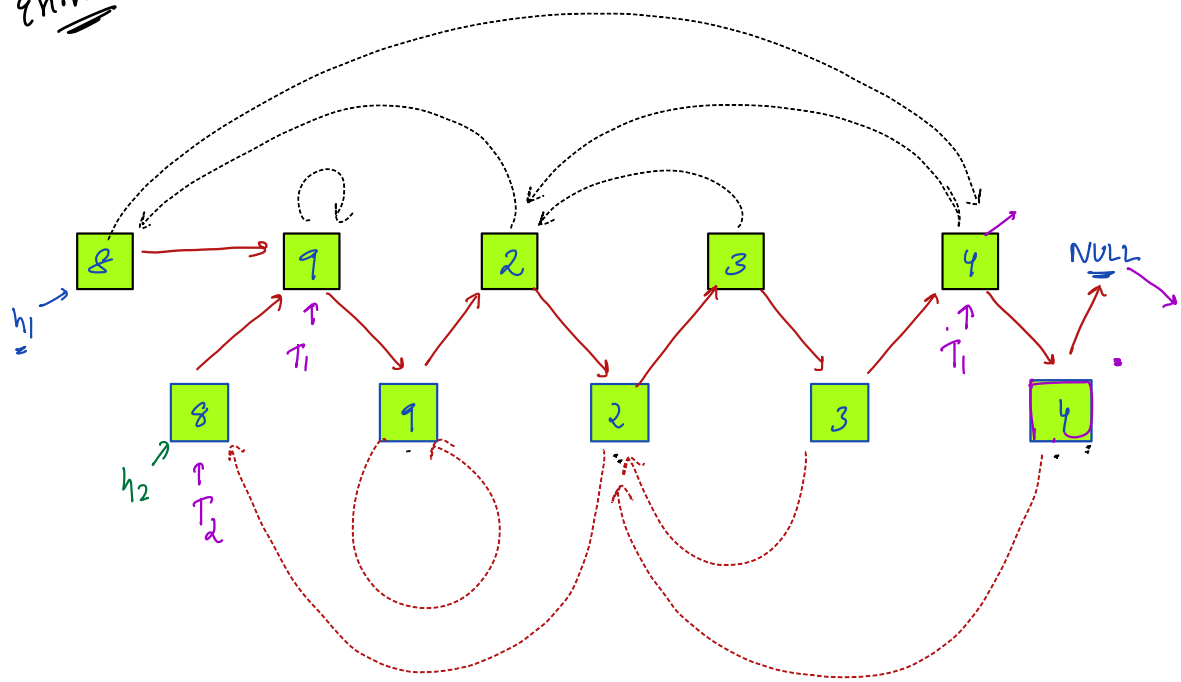
}

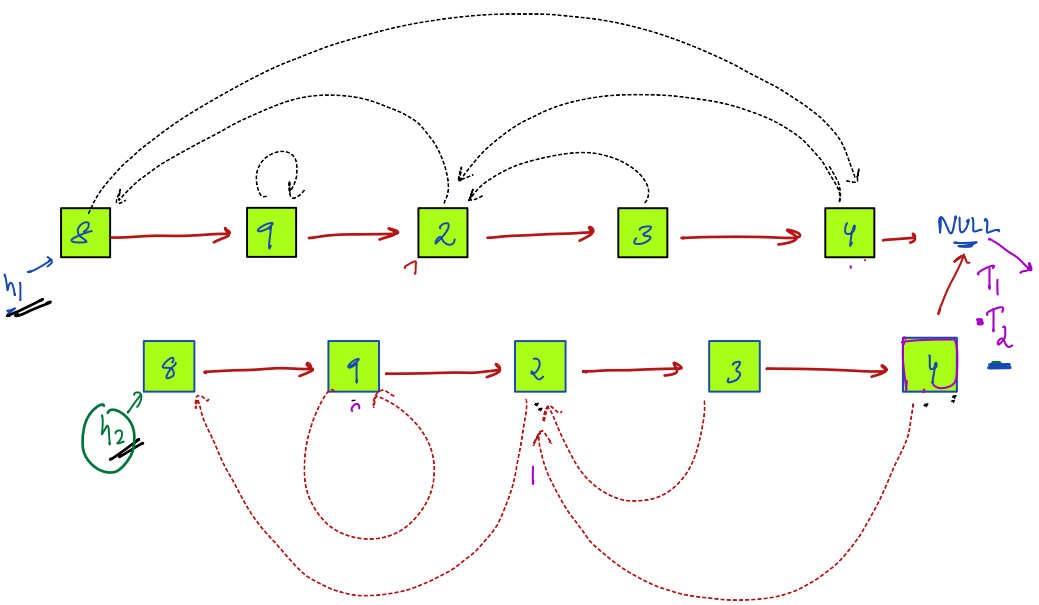
} O(N)

TC : O(N)

SC : O(N) }

Without Extra Space





$T_1.next = T_2.next$

$T_1 = T_1.next$

$T_2.next = T_1.next$ } Edge Case

$T_2 = T_2.next$

Pseudo code

1) Step1 : Insert nodes in between

2) Initializing Random pointers

```
T1 = h1 , T2 = h2
while ( T1 != NULL)
    T2.random = T1.random.next
    T1 = T1.next.next
    if( T1 != NULL) {
        T2 = T2.next.next }   // new can still get exception
    }
}
```

3) Separate :

```
while ( T1 != NULL) {
    T1.next = T2.next
    T1 = T1.next
    if ( T1 != NULL) {
        T2.next = T1.next
    }
    T2 = T2.next
}
```

{ T1 : O(N)
  S1 : O(1)

Thursday
Cs Happy Holiday

4) Return h_2