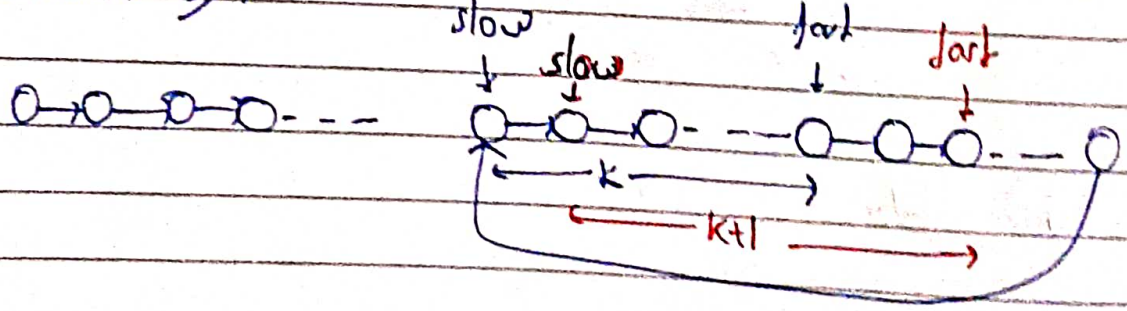
Q(n) Floyd's Cycle Detection / Tortoise & hare Algorithm

```
bool f(Node *head)
{   if(head==NULL || head->next==NULL)
        return 0;
    Node *slow=head, *fast=head;
    while(fast!=NULL && fast->next!=NULL)
    {
        slow=slow->next;
        fast=fast->next->next;
        if(slow==fast)
            return 1;
    }
    return 0;
}
```

Proof

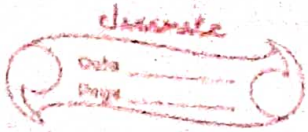
- Fast pointer enters loop before or at the same time slow pointer enters.
- The difference b/w the distance of slow & fast pointer increases by 1

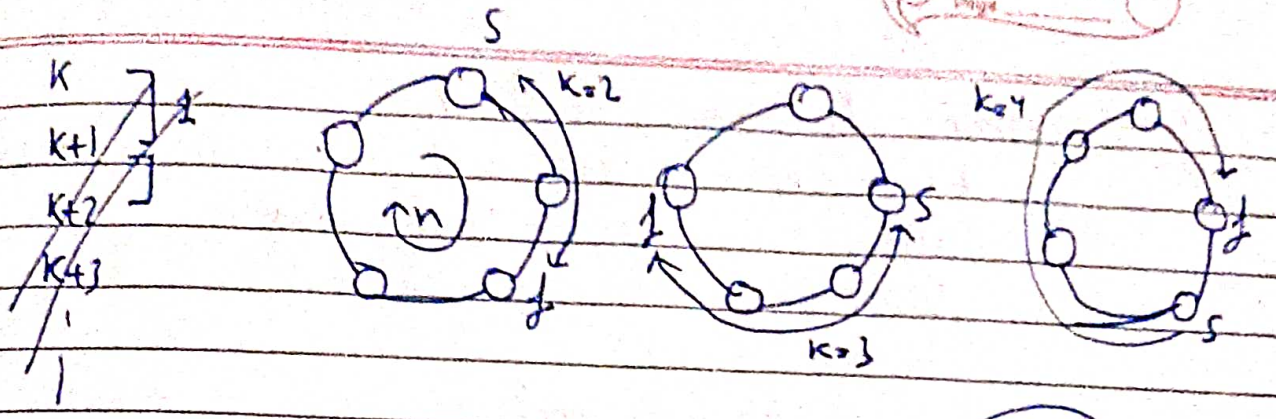


distance increases by 1 every time, so one time will come where distance is equal to n (no. of edge in loop) ↳ cycle length

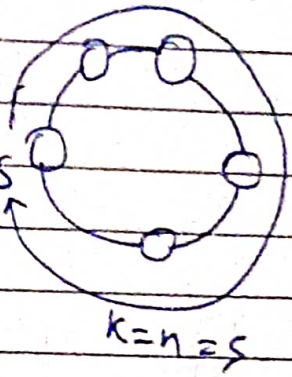
k
k+1
k+2
k+3 - - n

then they meet, they are at same node





K का और एक से increase करते रहते हैं एक समय आएगा जब K=n होगा तब दोनों meet करेंगे।

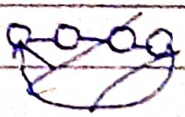


---

(142) ~~Detect & Remove~~ Linked list Cycle II

find the node from where cycle start.

```
listNode *detectCycle(listNode *head)
{  if(head==NULL || head->next==NULL) return NULL;
   listNode *slow=head, *fast=head, *temp=head;
   while(fast->next!=NULL && fast->next->next!=NULL)
   {
       slow=slow->next;
       fast=fast->next->next;
       if(slow==fast)
           break;
   }
   if(slow==temp) return slow;  // first node पर loop है
   while(slow!=temp)
   {  if(slow==temp)
          return temp;
   }
   return NULL;
}
```

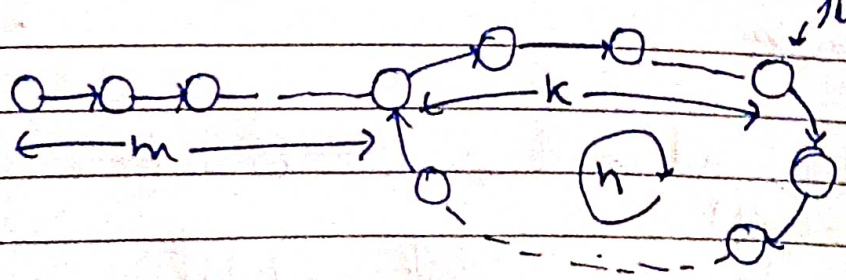


```
if(slow==fast){
while(slow!=temp)
{ slow=slow->next;
  temp=temp->next; }
return temp;
}
```

b) ListNode *f(ListNode *head)
{ if(head==NULL || head->next)==NULL) return NULL;
ListNode *slow=head, *fast=head;
while(fast!=NULL && fast->next!=NULL)
{
slow = slow->next;
fast = fast->next->next;
if(slow==fast)
break;
}
when no loop exist ← if(fast==NULL || fast->next==NULL) return NULL;
while(head!=slow)
{
head = head->next;
slow=slow->next;
}
return head;
}

Proof

- before first meeting point
(distance travelled by slow) * 2 = (Distance travelled by fast)



$(m+k+x*n)*2 = (m+k+y*n)$

where x → no of iterations made by slow pointer before the first meeting point

y → ——— fast pointer ———

अगर loop होता होगा तो, fast loop में घुमता रहेगा। ~~slow loop में~~ ~~तब तक~~

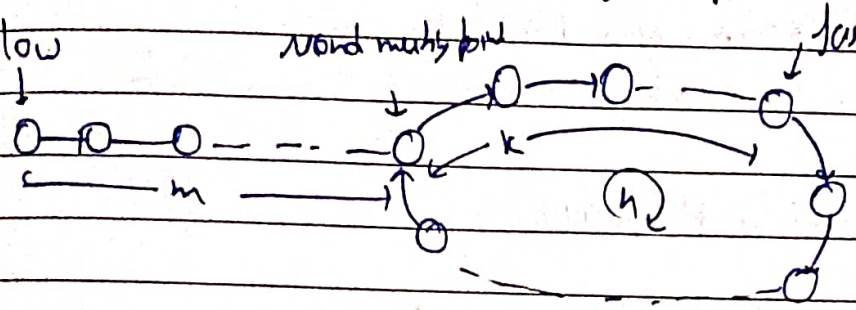
$$(m+k) = n(y-2x)$$

since everything is integer

$\therefore (m+k)$ is a multiple of n

Now if $(m+k)$ is a multiple of n, then second meeting point is going to be the first node of loop

proof that



① slow travel m distance to reach SMP

② fast will reach back to its position in $(m+k)$ times since $(m+k)$ is a multiple of n

$\therefore$ in m distance fast will reach SMP.

$(m+k) = n(y-2x)$

$y=3, x=1$	$y=5, x=2$
$m+k=n$	$(m+k)=n$ — —

• after the first meeting point point $(n=m+k)$ पे आयेगा।

Link को अलग-अलग करने वाले Question में एक extra Node create का लिया जाता है।

↓

(5) Segregate Odd & Even nodes in a LL

12 → 15 → 8 → 9 → 2 → 4 → 6 → N

o/p 8 → 2 → 4 → 6 → 12 → 15 → 9

o/p पहले even then odd order maintain रहे।

```
Node *f(int n, Node *head)
{
    Node *odd = new Node(-1);
    Node *even = new Node(-1);
    Node *o=odd, *e=even; ~~*even=head;~~

    while(head!=NULL)
    {
        if(head->data%2==0)
        {
            e->next=head;
            e=e->next;
        }
        else
        {
            o->next=head;
            o=o->next;
        }
        head=head->next;
    }
    o->next=NULL;
    e->next=odd->next;
    return even->next;
}
```

no change

Similar logic as Previous

(XII) Given a LL of 0's, 1's & 2's, sort it

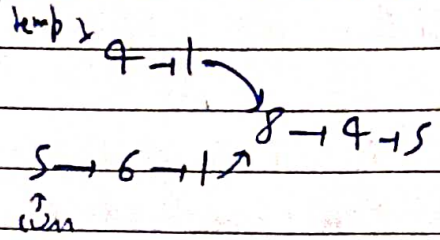
1→2→1→2→0→2→2
dp 0→1→1→1→2→2→2

```
Node *f(Node *head)
{ if(head==NULL || head->next==NULL) return head;
  Node *zno_head = new Node(0);
  Node *one_head ——
  Node *two_head ——
  Node *zno=zno_head, *one=one_head, *two=two_head,
       *curr=head;
  while(curr!=NULL)
  { if(curr->data==0)
    { zno->next=curr;
      zno=zno->next;
    }
    if(curr->data==1)
    { one->next=curr;
      one=one->next;
    }
    if(curr->data==2)
    { two->next=curr;
      two=two->next;
    }
    curr=curr->next;
  }
  zno->next=(one_head->next!=NULL)? one_head->next:
                                    two_head->next;
  one->next=two_head->next;
  two->next=NULL;
  return zno_head->next;
}
```

★ See

(XVI) Intersection of 2 LL

Given headA & headB of 2 LL (n) (m) where head at which node they intersect or NULL.



o/p node pointing to 8.

a) naive O(n²) O(nm)

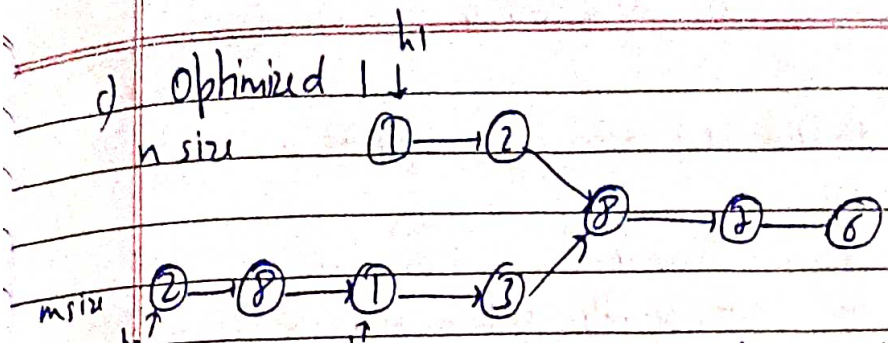
2 for loop — Iterate over one LL, & see if (curr == temp) then return node. (say 2nd LL)

b) Hashing: O(n+m) SC → O(n)

logic Iterate over first LL & store address of every node. Then iterate over 2nd LL & see if address of current node exist in set.

```
Node *f(Node *h1, Node *h2)
{   unordered_set<Node*> s;
    Node *head1 = h1, *head2 = h2;
    while(head1 != NULL)
    {   s.insert(head1);
        head1 = head1->next;
    }
    while(head2 != NULL)
    {
        if(s.find(head2) != s.end())
            return head2;
        head2 = head2->next;
    }
    return NULL;
}
```

d) Optimized 1

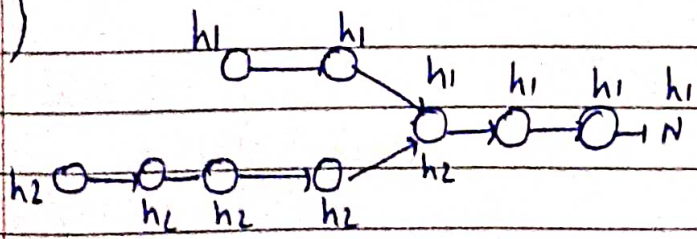


- Just find length of both linked list (say n, m), you can use one loop to find it
  take abs(n-m) & move larger ll h2 pointer ahead (here 2)
- Now move simultaneously h1 & h2 till they are equal or NULL.

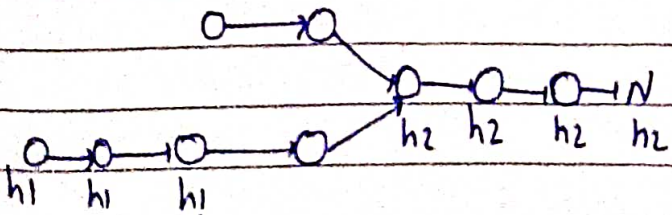
d) Optimized 2:

```
Node *f(Node *headA, Node *headB)
{  if(headA==NULL || headB==NULL) return NULL;
   Node *h1=headA, *h2=headB;
   while(h1 != h2)
   {
      h1 = (h1==NULL)? headB : h1->next;
      h2 = (h2==NULL)? headA: h2->next;
   }
   return h1; // or h2
}
```

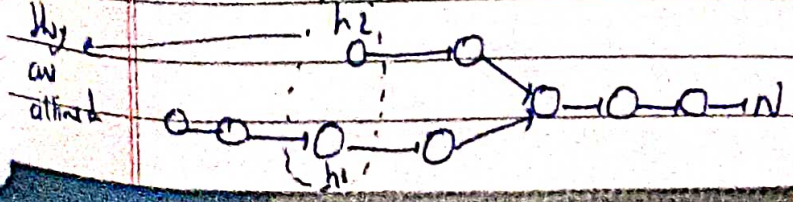
logic



here h1==NULL
take h1 to headB



here h2==NULL
take h2 to headA



Let say you have 3 t-shirt
you used 1 t-shirt on monday
——— 2nd ——— Tuesday
——— 3rd ——— Wednesday
Then LRU t-shirt is on monday

Least Recently Used →

LRU Cache

- This DataStructure should have two function

(i) get(key) function
if key is present return value else return -1.
← make that node last recently used node (basically insert it right after head)

(ii) put(key, value)
we have this (key, value) in Cache DS.

- If size of Cache is full then we remove the LRU guy & then we insert.

Example size of Cache = 2

put(1,1) {(1,1)} LRU is (1,1)

put(2,2) {(1,1),(2,2)} LRU is still (1,1)

get(1) → return 1 LRU is now (2,2) since you have used (1,1)

put(3,3) → {(1,1),(3,3)} LRU is (1,1)

get(2) → -1

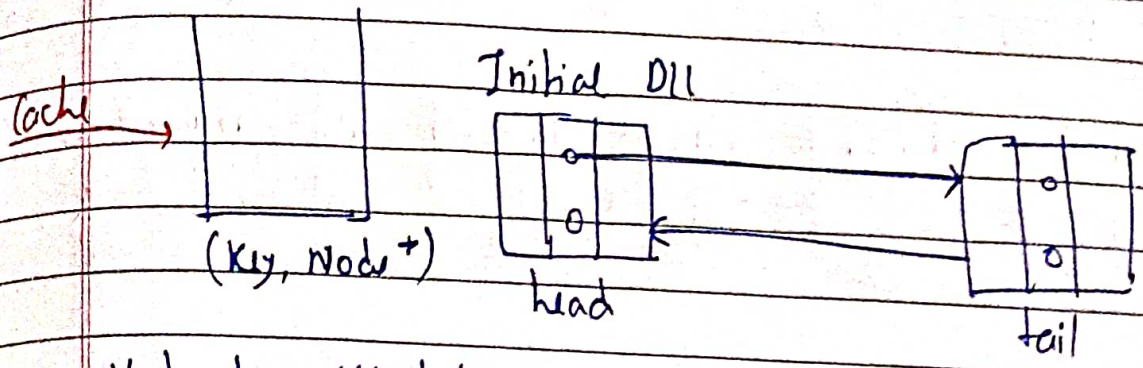
put(4,4) → {(1,4),(3,3)} LRU is (3,3)

get(1) → -1

get(3) → 3

get(4) → 4

* We have to design both these function in O(1) TC
→ We will use Hashmap & DLL

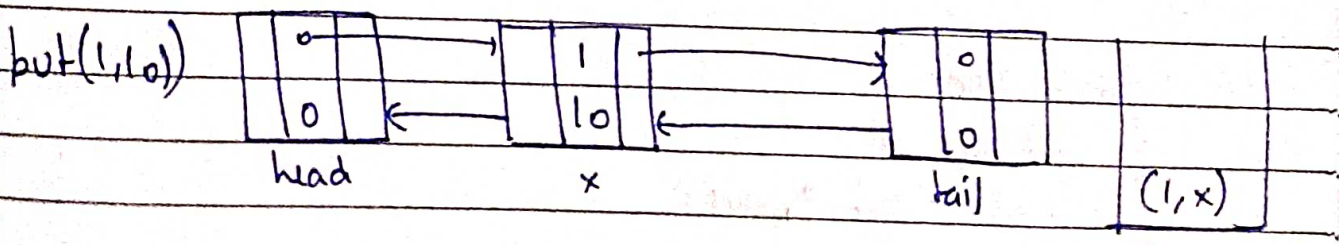


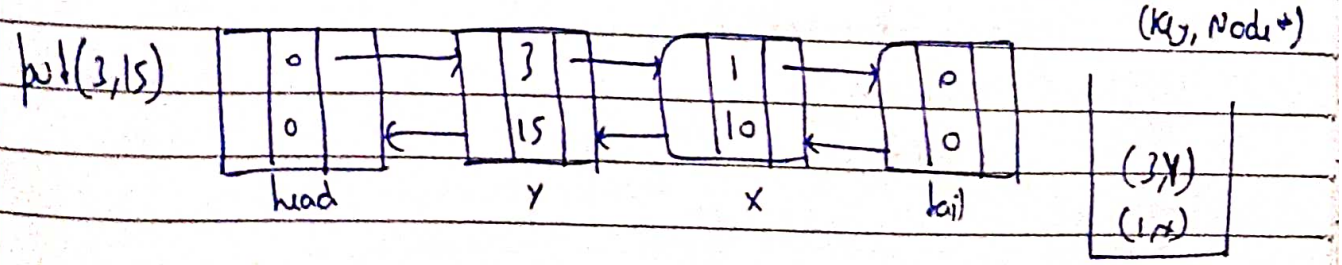
- ~~Hashmap will tell~~

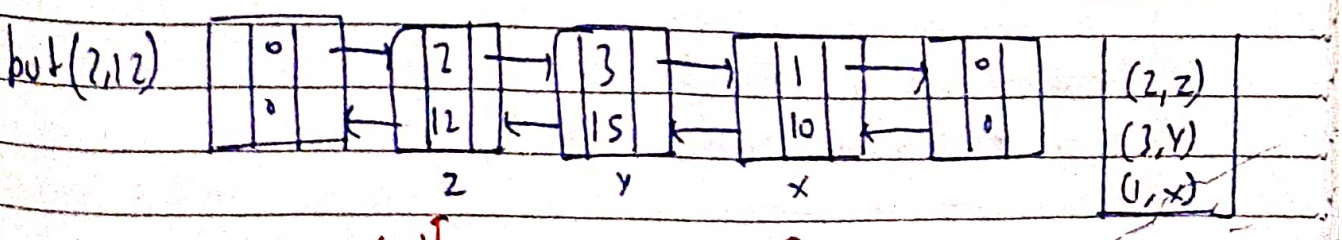
Example : Cache size = 3
Operation : put(1,10) put(3,15) put(2,12) get(3) put(4,20)
get(2) put(4,25) get(6)

* put(1,10) : we go in hashmap & check if 1 is present or not if not present & size is <3 then we insert (1,10).

→ We insert always right after head





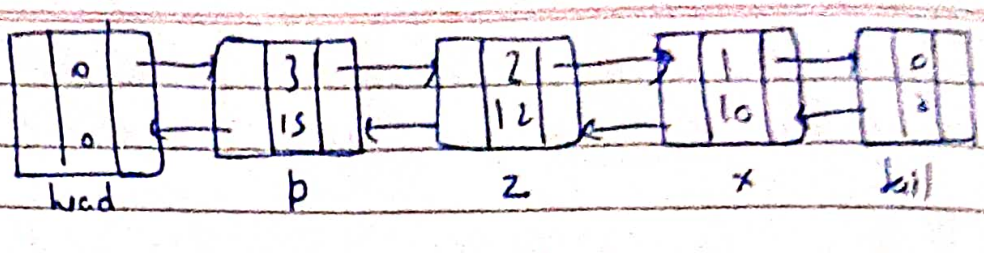


last recently used or Most recently used

last recently used

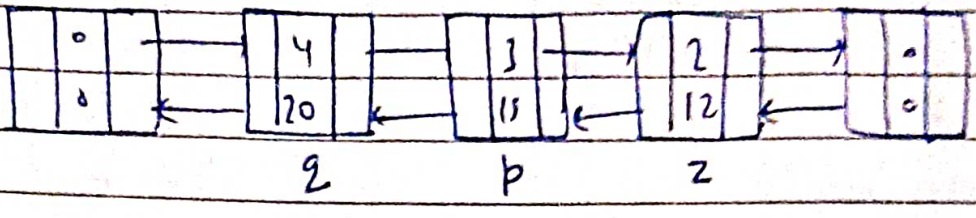
See this step properly

get(3) → 15



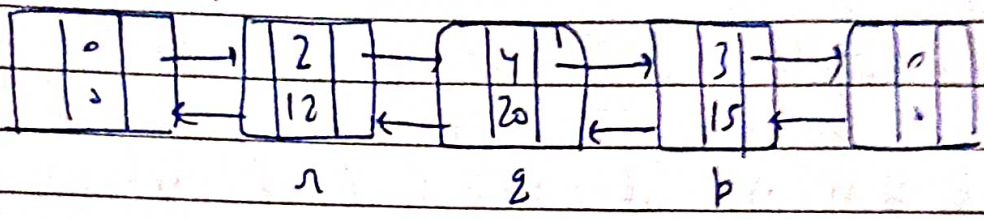
(2,2)
(3,5)
(1,4)

put(4,20)



(4,2)
(2,2)
(3,5)

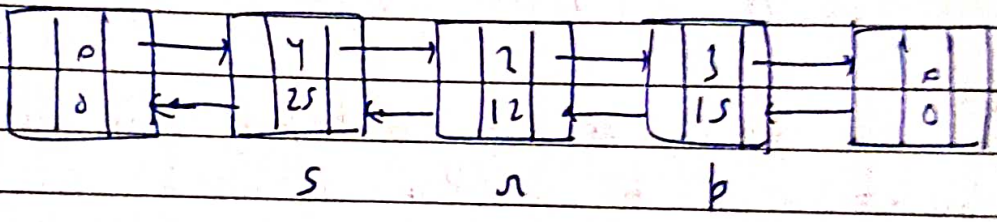
get(2)



(4,2)
(2,r)
(3,5)

See key step properly

put(4,25)



(4,s)
(2,r)
(3,5)

get(6) → -1

```
struct Node
{   int key, val;
    Node *next, *prev;
    Node(int x, int y)
    {   key = x;
        val = y;
    }
};

Node *head = new Node(-1, -1);
Node *tail = new Node(-1, -1);
int size;
unordered_map<int, Node*> m;
```

```
LRUCache(int capacity)
{   size = capacity;
    head->next = tail;
    tail->prev = head;
}
```

```
void deleteNode(Node *del)
{
  del->prev->next = del->next;
  del->next->prev = del->prev;
}
```

```
void addNode(Node *newNode)
{  newNode->next = head->next;
   head->next = newNode;
   newNode->next->prev = newNode;
   newNode->prev = head;
}
```

```
int get(int key)
{  if(m.find(key) != m.end())
   {
     Node *resNode = m[key];
     int res = resNode->val;
     m.erase(key);
     deleteNode(resNode);
     addNode(resNode);
     return res;
   }
   return -1;
}
```

```
void put(int key, int value)
{
   if(m.find(key) != m.end())
   {  Node *existingNode = m[key];
      m.erase(key);
      deleteNode(existingNode);
   }
   if(m.size() == size())
   {   m.erase(tail->prev->key);
       deleteNode(tail->prev);
   }
   addNode(new Node(key,value));
   m[key] = head->next;
}
```

* LRU cache is also implementable using Array but takes O(n) time & O(m) space
  (m = capacity of cache.)

(XIII) Merge two Sorted list

1→2→4
1→3→4

o/p 1→1→2→3→4→4→4

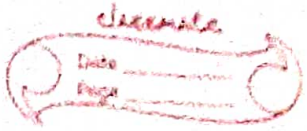
```
Node * merge ( Node * head1 , Node * head2)
{   if(head1==NULL && head2==NULL) return NULL;
    if (head1==NULL) return head2;
    if (head2==NULL) return head1;

    Node * dum = new Node(10);
    Node * ans = dum;
    while ( head1!=NULL && head2!=NULL)
    {
        if( head1->val <= head2->val)
        {   ans->next = head1;
            ans = ans->next;
            head1 = head1->next;
        }
        else
        {   ans->next = head2;
            ans = ans->next;
            head2 = head2->next;
        }
    }
    if (head1!=NULL)
        ans->next = head1;
    if ( head2!=NULL)
        ans->next = head2;

    return dum->next;
}
```

odd → 1→2→3→4→5
1→2→3 | 4→5

even → 1→2→3→4→5→6
1→2→3 | 4→5→6



(XIV) Merge Sort for LL O(nlogn)

Given pointer to the head of LL, sort using mergeSort

```
Node *mergeSort(Node *head)
{  if(head==NULL || head->next==NULL) return head;
   Node *slow=head, *fast=head->next;
   while(fast!=NULL && fast->next!=NULL)
   {
       slow=slow->next;
       fast=fast->next->next;
   }
   Node *first=head, *second=slow->next;
   slow->next=NULL;
   Node *h1=mergeSort(first);
   Node *h2=mergeSort(second);
   return merge(h1,h2);
}
```

same function as previous Question

→ Similar Question MergeSort on Doubly LL. (अब previous pointer भी connect होगा)

```
ans->next=head1;
head1->prev=ans;
```

(XV) Merge k Sorted LL:

lists → का vector

[[1,4,5],[1,3,4],[2,6]] o/p → [1,1,2,3,4,4,5,6]

a) Naive ~~If~~ O(nlogn)

- store all the element in LL & sort it in NlogN.

b) (1,4,5), (1,3,4), (2,6), (7,8) ~~O(n+k)~~ O(nK) (k = no. of lists)

merge taken O(n)

पहले दोनों को merge किया फिर आया उसको 3rd से merge किया so on.

1) Priority Queue (min Heap) (NlogK) SC, O(logK)
एक linked list का पहला element देता है।

```
class comp
{ public:
    int operator()(ListNode *a, ListNode *b)
    {   return a->val > b->val;
    }
};
```

```
ListNode *f(vector<ListNode*> &lists)
{   if(lists.size()==0) return NULL;
    priority_queue<ListNode*, vector<ListNode*>, comp> p;
    for(int i=0; i<lists.size(); i++)
    {   if(lists[i]!=NULL)
            p.push(lists[i]);
    }
    ListNode *dum = new ListNode(0);
    ListNode *ans = dum;

    while(!p.empty())
    {
        ans->next = p.top();
        ans = ans->next;
        ListNode *temp = p.top();
        p.pop();
        if(temp->next!=NULL)
            p.push(temp->next);
    }
    ans->next = NULL;
    return dum->next;
}
```

T(n) = O(N log k) S(n) = O(1)

d) आखरी दो LL को जोड़ कर पहले LL को assign (basically head को return lists[0] से)

```
ListNode *f(vector<ListNode *> &lists)
{   if(lists.size()==0) return NULL;
    int first=0, last=lists.size()-1;
    while(last!=0)
    {
        int i=0, j=last;
        while(i<j)
        {
            list[i] = merge(list[i], lists[j]);
            i++;
            j--;
            if(i>=j)
                last=j;
        }
    }
    return lists[0];
}
```

previous Question

~~first=0~~ ~~last=9~~

Logic first=0 last=9

L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8 L_9 L_{10} → 10 linked list

i i i i i j j j j j

merge L_1 & L_{10} & assign the head to list[0]

i++, j--

i	j
0	9
1	8
2	7
3	6
4	5
5	4

assign j=last means j=4

→ L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8 L_9 L_{10}
i j

→ L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8 L_9 L_{10}
i j

→ L_1 L_1 — L_{10}
i j

→ L_4 L_4 — L_{10}
j i

last=j=0

while(last!=0) stop

return lists[0];

1→2→3→2→1

stack में push करके भी check कर सकते हो for pallindrome. (push all val in stack & then iterate over LL & check (stack.pop() != curr.data) return 0;

(XVI) Check for Pallindromic LL

1→2→3→2→1	1→2→2→1	[]	1→2→3→2→1
o/p → True	o/p → True	T	True

a) Naive → put elements in Array then check for Pallindrome
T.C → O(N) + O(N) S.C → O(N)

b) Find middle of LL, reverse right half of LL, now check the value of both LL by iterating

after finding middle: 1→2→3→3→2→1 (s, f) → reverse → 1→2→3→1→2→3 (he, s) → iterate both & check value

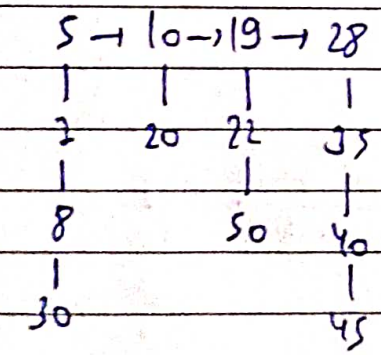
```
bool f(listNode *head)
{   listNode *slow = head, *fast = head;
    while(fast→next != NULL && fast→next→next != NULL)
    {
            slow = slow→next;
            fast = fast→next→next;
    }
    listNode *h = slow→next;
    listNode *temp1 = NULL, *temp2;
    while(h != NULL)
    {
            temp2 = h→next;
            h→next = temp1;
            temp1 = h;
            h = temp2;
    }
    slow→next = temp1;
    slow = slow→next;
```

→ The flattened list will be printed using the bottom pointer instead next pointer

```
while(slow != NULL)
{   if(head->val != slow->val)
        return 0;
    head = head->next;
    slow = slow->next;
}
return 1;
} // Remember interviewer may ask to reverse back the original LL then you again find middle & reverse the second half of LL.
```

(XVI) Flattening a LL

- Given LL every node represent a sub-linked list.
- each of the sub-linked list is in sorted order



o/p: 5→7→8→10→19→20→22→28→30→35→40→45→50

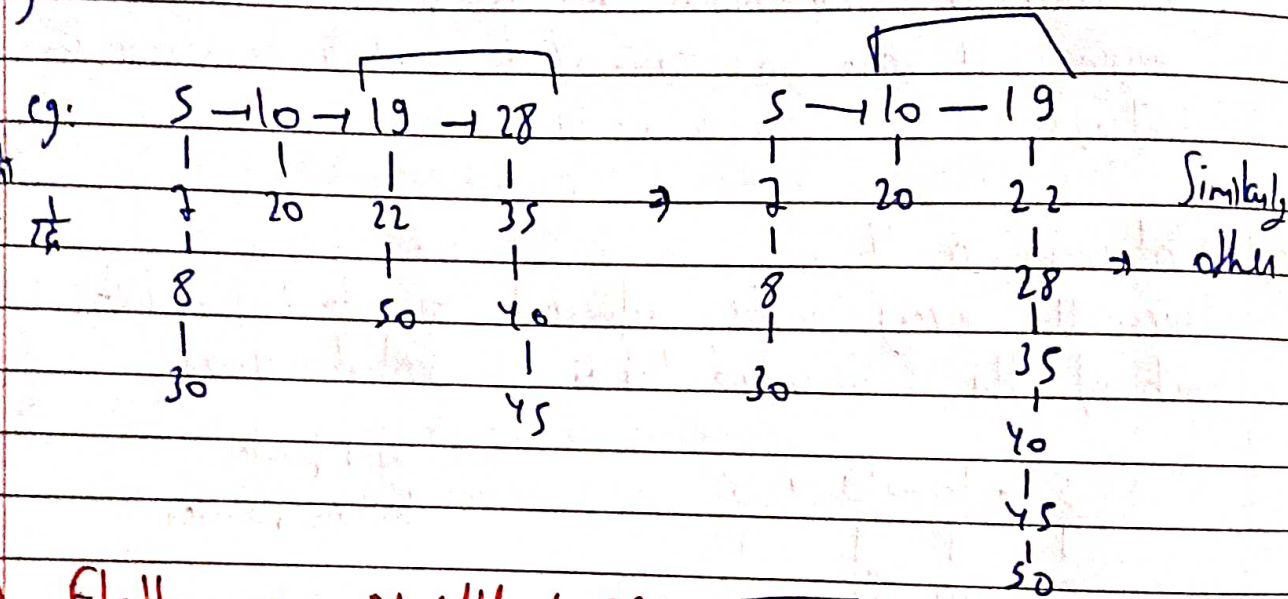
a) Iterative

```
Node * flatten(Node *root)
{ if(root==NULL || root->next==NULL) return root;
  Node *head = root, *ns = root;
  while(head->next != NULL)
  {
      Node *h = head->next;
      ns = merge(ns, head->next);
      head = h;
  }
} return ns;
```

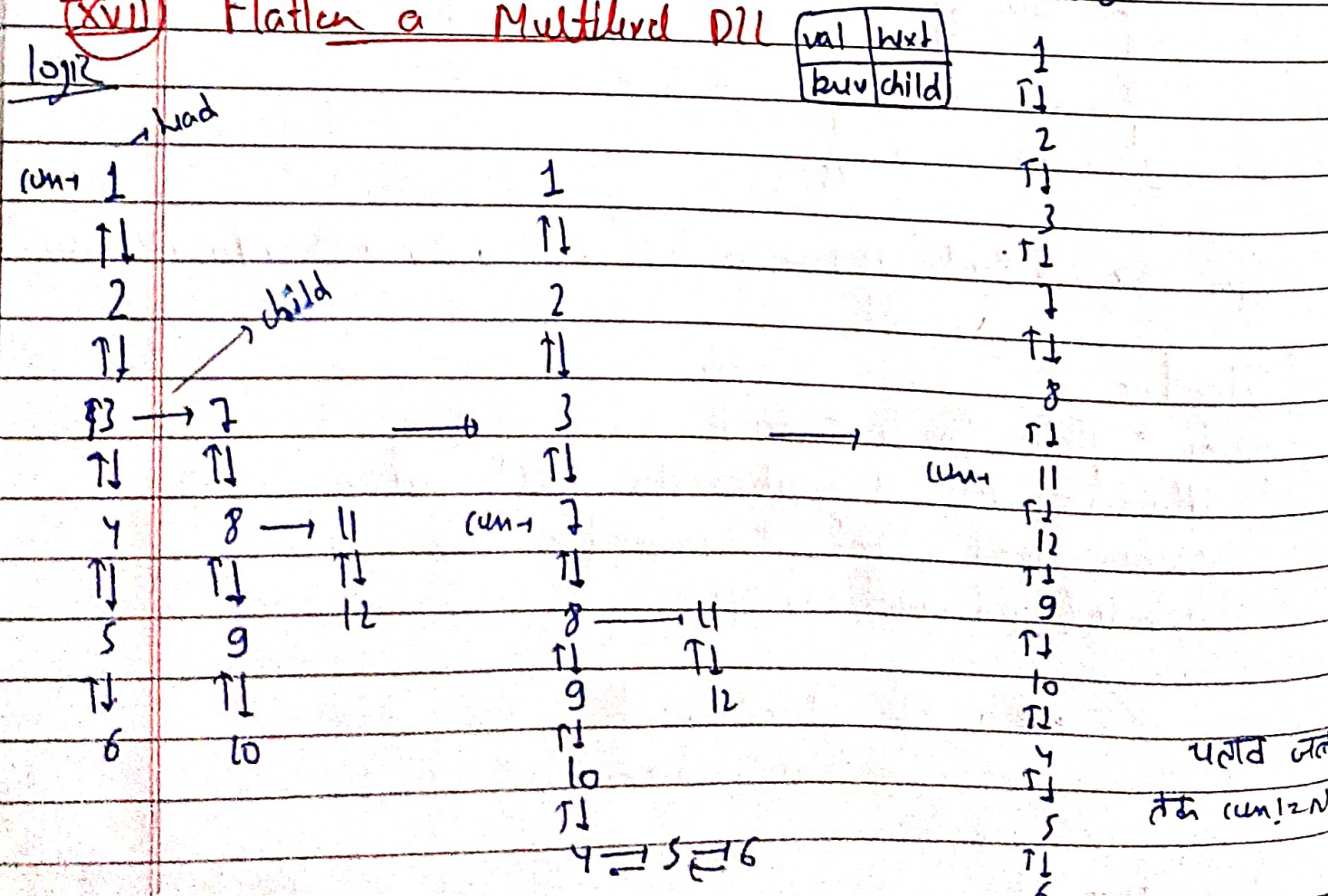
remember !!!!! (->next) change to (->bottom)

b) Recursive

```
Node *flatten(Node *root)
{
    if(root==NULL || root->next==NULL) return root;
    Node *ans = flatten(root->next);
    Node *ny = merge(root, ans);
    return ny;
}
```



(XVII) Flatten a Multilevel DLL



→ Return the head of the flattened list & make the child = NULL of all Node

```
Node *f(Node *head)
{
    Node *cur = head;
    while(cur != NULL)
    {
        if(cur->child != NULL)
        {
            Node *temp = cur->next;
            cur->next = cur->child;
            cur->next->prev = cur;
            cur->child = NULL;
            Node *h = cur->next;
            while(h->next != NULL)
                h = h->next;
            h->next = temp;
            if(temp != NULL)
                temp->prev = h;
        }
        cur = cur->next;
    }
    return head;
}
```

(XVIII) Swap Nodes in Pair

1 → 2 → 3 → 4 → 5 → 6 → N

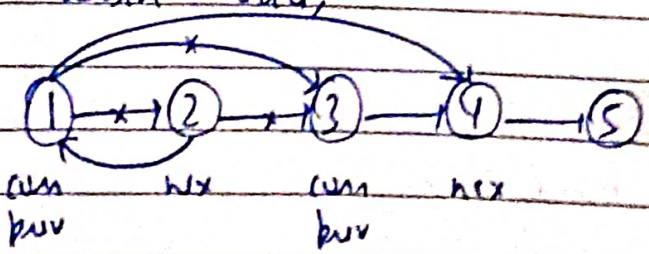
o/p 2 → 1 → 4 → 3 → 6 → 5 → N

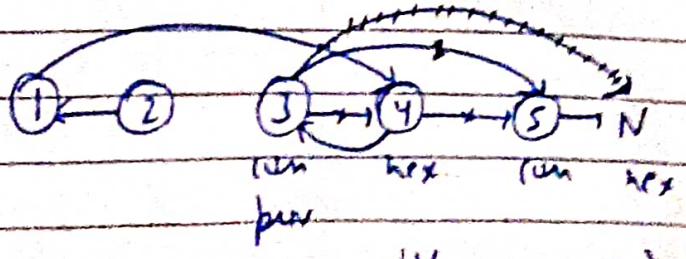
[ ]	[ ]
[ ]	[ ]

(4-4) एक linked list को उल्टा करो।

```
ListNode *f(ListNode *head)
{
    if(head==NULL || head->next==NULL) return head;
    ListNode *cur=head, *prev=head, *nex=head->next;
    head=head->next;
    while(cur!=NULL && cur->next!=NULL)
    {
        cur->next=nex->next;
        nex->next=cur;
        cur=nex->next->next;
        if(cur==NULL)
            break;
        nex=cur->next;
        if(nex==NULL)
        {   prev->next=cur;
            break;
        }
        prev->next=nex;
        prev=cur;
    }
    return head;
}
```

Eg :





if(nex==NULL)
prev->next=cur
break;