Date 03 01 25
Page No.

⊛ Linked List - Short Notes

- In Linked list memory is dynamically allocated (only when require) so no extra space/wastage unlike in arrays/arraylist it is fixed & to add more elements in arraylist it doubles itself copy rest element then new added.
- In LL, they don't have any indexes (they are randomly stored in Heap) & are pointed via reference variables.
- We can't get individual here as in arraylist (list.get(5));

* Single Linked List :-

- every single (node) item knows about next only.
- Head is a ref. variable pointing to first node.
- Tail " " " " " " last node & next to it is 'null'.

• Basic declaration :-

```
public class LL {
    private class Node {
        Node next;
        int value;
        const. of value
    } " " both
    private Node head;
    " " tail;
    int size;
    public LL() {
        this.size = 0;
    }
```

Teacher's Signature.....

Date
Page No.

Insert First :-

→ create Node with value
→ determine "head" → node.next = head (or, determining node position & putting)
→ assign head to node → node = head
→ null ptr. exception → if (tail == null) { tail = head }
(tail == null)
→ Don't forget to increase size → size++;

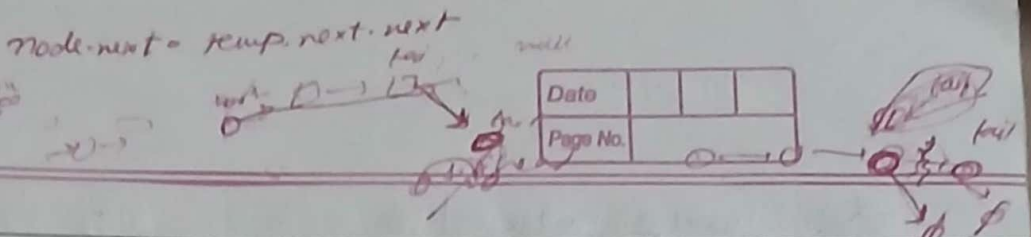
Insert Last :-

→ "if list is empty (null ptr. exception) you can delegate this work to already made insertFirst();
→ else → create node of value
→ tail.next = node → determine node position & put it
→ tail = node → assign it as tail
→ size++ → don't forget this.

Insert at index :

→ here we need node's value as well as index (const? of both used)
→ if index == 0 → insertFirst
→ if index == size → " Last
→ Create a temp node & move it till the index position to perform operation on that place (O(n)).
∴ Node temp = head; → create
for (int i=1; i< index; i++) { → till index
∵ 0 is head only (here, temp at start) ↳ before index
temp = temp.next; → moving
→ Create node of value & next (temp.next) → place of index
temp.next = node; → inserted
size++; → dont forget

Teacher's Signature.....



- **Delete first:**
  - → assign head.next to head (only step required).
  - → size--;
  - ← int value = head.value (for returning purposes, no use)
  - head = head.next
  - if (head == null){ → null ptr exception (null → no ans.) (kya delete karoge aage jaake assign it to tail.
    - tail = null;
    - }
  - size--;
  - return value;

write at start coz head pointer will change

- **Delete last:**
  - → if size <= 1 → null items ∴ return;
  - → else → get the second last node by iterating temp node from head to particular position's node (∵ for ~~second~~ second last → size-2)
  - → int value = tail.value (returning purposes)
  - → assign secondlast to tail
  - → & next to tail as null ∴ removed
  - → size-- (Don't forget)

- **~~Delete~~ Delete at index:**
  - → if index at zero ∴ delete first
  - → " " " size-1 (last) ∴ " last
  - → make & get a prev. node at index-1 position
  - → assign next to next node as next node (bich wala apne aap udd gya)
    - i.e. prev.next = prev.next.next;
    - size--;

- **Display:**

```
for ( Node temp = head; temp != null; temp++){
    sout (temp.value + "→ ");
    temp = temp.next;
}
sout ("end ");
```

Teacher's Signature.....

Date
Page No.

* Doubly linked list

- same this just one more ref. variable added i.e. previous
- prev. of head → null
  " " tail → "

- **Basic Declaration:-** (no need to declare tail in DLL)

```
public class DLL{
    private class Node{
        Node next;
        Node prev;
        int value;
        constr of value
        " " all three
    }
    private Node head;
```

- **Insert First:-**
  → Create Node with value
  → determining node position using head → node.next = head
  ~~→ assign head to node~~
  → " " " → node.prev = null;
  → if (head != null){ → * must for this
  ~~node~~ head.prev = node;
  }
- * → assign ~~head~~ node to ~~node~~ head at last → ~~node = head;~~ head = node;

- **Insert last :-**
  → Create node of value.
  → " last " from head → Node last = head;
  node.next = null; → ∵ we're inserting this at last
  → null ptr. exception → if (head == null){
  head = node;
  } node.prev = null;

Teacher's Signature.....

Date			
Page No.			

→ move 'last' till end →
```
while (last.next != null){
    last = last.next;
}
```

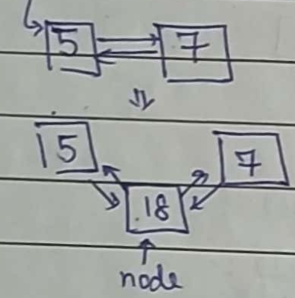
- updated finally last value assigned to 'last' ← last.next = node;

'last' को bhul thodi jaoge ← node.prev = last;
usko bhi define करो using node.

- Insert at Index:

→ method of after (int), value.
→ Get node p after which we've to insert.
→ If p == null ∴ not exist ∴ return
→ Create node of value
(Note:- p is that node just before 'index') — already said
→

find:
→ make temp node from head
→ till it reaches null, if we got the node.value == value
∴ return node (returned as an index)
else
node = node.next;
→ return null (if not found)



① 18 → 7 ∴ node.next = p.next;
② 5 → 18 ∴ p.next = node;
③ 18 → 5 ∴ node.prev = p;
④ 7 → 18 ∴ node.next.prev = node; → null ptr exception
→ if (node.next != null)

- For Deletion in DLL (think on your own)
- Display:-

→ Declare temp node at head
→ Declare last node as null
→ while temp node not reaches null
```
    sout (node.value + "→ ");
    last = node;   → lly last को last tak isi loop के madad से pahucha do
}   node = node.next;
sout ("end");
```
→ sout ("In reverse")
```
while (last != null) {
  sout (last.value + "← ");
} last = last.prev
sout ("start");
```

Teacher's Signature........................

* Circular linked list :-

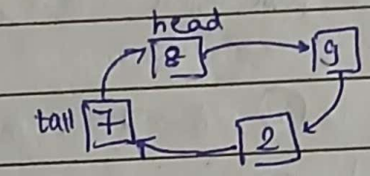
- same as single LL + here no thing as "null".
* - No big concept of index (specific) because we can choose head + tail.

- Basic declaration :- + tail points to head

```
public class CLL {
    private class Node {
        Node next;
        int value;
    } // counter of value.
    private Node head;
    "    "  tail;
```

non-parameterized constr. of CLL → `public CLL(){`

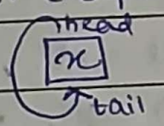
```
    this.head = head;
    ".  tail = tail;
```



- Insert

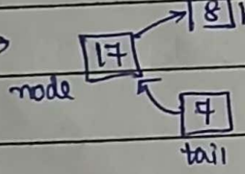
→ Create node of value

→ null ptr exception → `if (head == null){`




```
    head = node;
    tail = node;
}
```

→




```
tail.next = node;
node.next = head;
~~node = tail~~; tail = node;
```

assign node as tail

- Delete, Display ⇒ concept of do while loop check video.

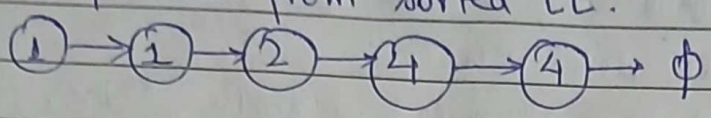
Teacher's Signature.....

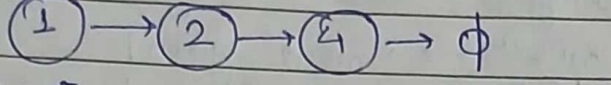
node.val == node.next.val

Date 05/01/25
Page No.

* Linked list imp. Questions → Imp. FAANG level Questions

1) Remove Duplicates from Sorted LL.

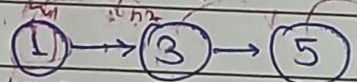
e. 

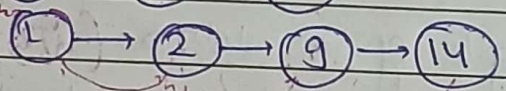
Ans ⇒ 

Steps ⇒ → create a temp "node" to traverse along LL starting from head
→ till it reaches null, if duplicate, next will be node.next.next, else next continues. (node."value" ; node.next."value" ; at here only)
→ Don't forget to decrease size
if tail is mentioned → assign that node as tail & complete by mentioning tail.next = ϕ.
→ otherwise return head only.
→ give null ptr. exception at start.

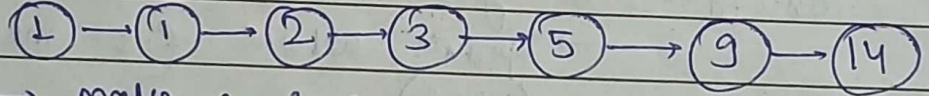
Note imp:- Check what you're traversing & what is given.

2) Merge Two Sorted Lists → in LL code (not in DSA practice) check leetcode

e. list 1: 

list 2: 

& already sorted

Ans. ⇒ 

concept ⇒ → make a final ans list
→ Get head of both list, while both doesn't reach till null compare heads and "insert last" the smaller one & move forward in ans list
→ If any one list reaches null, give while loop till other one reaches null & prints its remaining nodes
→ return ans.

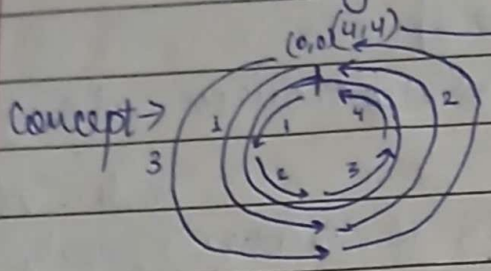
Linked

Teacher's Signature.....

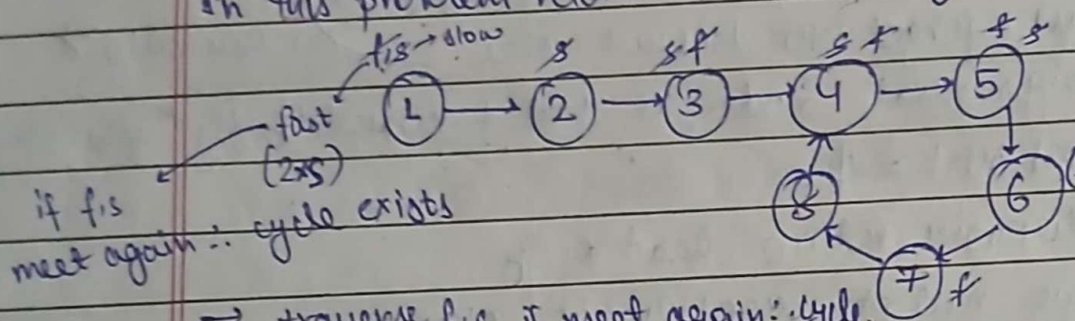
Date
Page No.

3) Linked list cycle

★★ Fast and slow pointer approach
→ Cycle detection
→ Finding a node in cycle, etc.



In this problem return true if it has cycle else false.



→ traverse f,s if meet again ∴ cycle.
↓
while f != null & f.next != null
∵ f is moving double ∴ ek aage thi pehle hi check karlo ~~the~~ moving ice.

O(N) → Time

4) Find length of a loop (GFG)

concept :-
→ cycle must be there (write code for this)
→ point where f,s meet again, make (node)
a temp node as slow alongwith length (counter) as zero (at start)
→ traverse temp node with condition it does not again equals to slow using "do while loop"

```
do {
    temp = temp.next
    length++;
} while (temp != slow);
return length
```

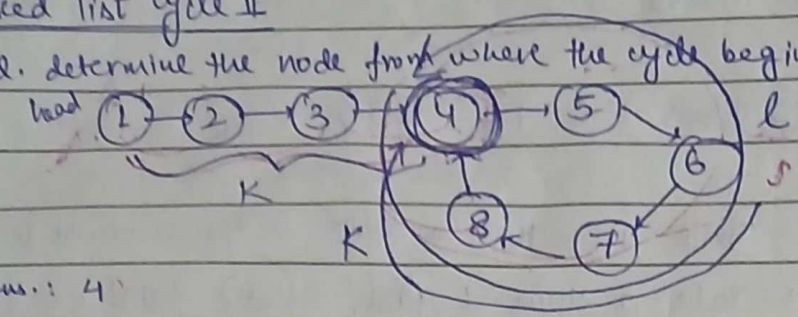
→ else return 0;

Teacher's Signature.....

Date | Page No.

4) Linked list cycle II

Q. determine the node from where the cycle begins.



Ans.: 4

Concepts :- ⇒ → Find the length of cycle = l

→ move s ahead by length of cycle times (while(length > 0) s = s.next length--)

→ Now keep moving f one by one from start & similarly move s one by one from that particular position

- while f != s
- → return s

→ check code

5) Happy Number Google 2017

Q. e.g.1 n=19

$1^2 + 9^2 = 1+81 = 82$

$8^2 + 2^2 = 64+4 = 68$

$6^2 + 8^2 = 36+64 = 100$

$1^2+0^2+0^2 = \boxed{1}$

Ans. true

eg. n=12 → $1^2 + 2^2 = 1+4 = 5$

$5^2 = 25$

$2^2+5^2 = 4+25 = 29$

$2^2+9^2 = 4+81 = 85$

$8^2+5^2 = 64+25 = 89$

$8^2+9^2 = 64+81 = 145$

$1^2+4^2+5^2 = 1+16+25 = 42$

$4^2+2^2 = 20$

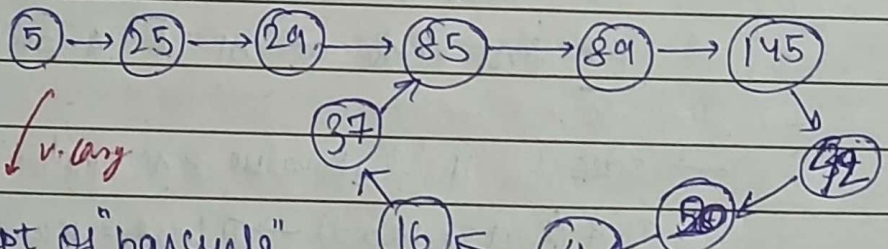
$5^2+8^2 = 89$ (repeated ∴ ∞ terms)

$2^2+0^2 = 4$, $4^2 = 16$, $1^2+6^2 = 37$, $3^2+7^2 = 58$

whary

Ans. false

Concepts :- (n=12)



v. easy

use the concept of "hascycle"

thats it → only diff. is that here LL is not mentioned

∴ ⇒ → initially slow & fast = n Socho

→ while slow != fast do slow → find sq., fast → find sq's sq.

→ if slow = 1 → true, else false

→ For find sq. get individual value sq. it & return the stored value.

Teacher's Signature.....

Mocha

Date 07 01 25
Page No.

20 easy

4) Middle of the linked list

Q. 1 → 2 → 3 → 4 → 5

Ans. 3

Note:- if there are two middle nodes ∴ return second middle

easy Concept:- → Take a slow & fast (2×slow) pointer.

→ Traverse both from start till fast reaches null or (fast.next) → to save time

→ The moment at which fast reaches null slow ptr will be at (1/2) ∴ return slow.

5) Sort List.

150

Q. 5 → 4 → 3 → 2 → 1

Ans. => 1 → 2 → 3 → 4 → 5

not Optimized O(n)

the merge sort algo. checks code

Concept => → use Bubble sort algo. approach.

→ Firstly make bubblesort method with no parameters & one with two parameters row & col (obv. 0).

→ Pass/initialize row as size-1 & col as 0 ~~in row~~ by calling parameterized fn. from non-para. fn.

→ null ptr. exception ~~(if size=1)~~ i.e. if row == 0 → return.

→ For condition col < row (till row reaches till end) get first node at col & second at col+1.

Obvious points

→ case:1 if first value > second ∴ swap :-

① if (first == head) → ① head = second, ~~second.next = first~~,

② first.next = second.next

③ second.next = first.

agar iska pehla link daal to first ka value garbage main chala jayega

② else if (second == tail) → ① make a prev node by getting at col-1

② prev.next = second

③ tail = first

④ first.next = null;

⑤ second.next = tail;

Teacher's Signature.....

not first coz you're at the end of the statement first tail bann chuka hai.

③ else → ① ② of ② → else if same

→ ③ first.next = second ④ second.next = first.

→ call bubblesort (row, col+1) → moving block to next (bubble sort concept)

→ else " " (row-1, 0)

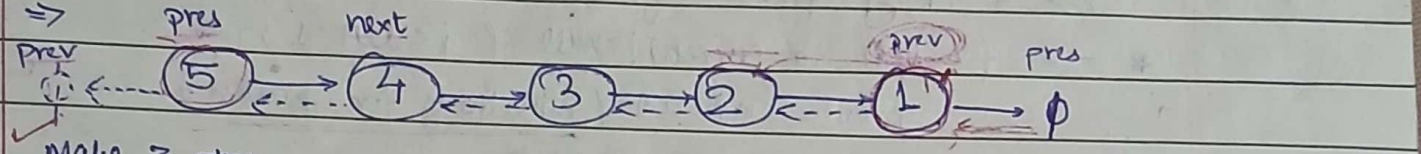
9. Reverse Linked List

Q. 5 → 4 → 3 → 2 → 1 → ϕ

Ans. ϕ ← 5 ← 4 ← 3 ← 2 ← 1

in place reversal (iterative approach)

⇒



Make 3 ptrs → prev, present, next

⇒ iterate present till null and assign backwards

while present != null → pres.next = prev (i.e. next of 5 is null)

prev = pres

pres = next

next = next.next

only if next != null ← can give null ptr except

now traversing

in the end prev at ① & pres at ϕ

∴ head = prev

⇒ (base condition if size < 2 ∴ return) → if (head == null) → return head; (think according to ques.)

recursive approach :→

1. temp node pointing at head 2. if node is at tail ∴ head = tail, return (assign it as head)

3. pass node.next to argument fn. (node is one step ahead now) → till tail

4. tail.next = node ∴ while returning (remain values of ~~node is at tail-1 position~~ stack)

tail = node assign it to tail

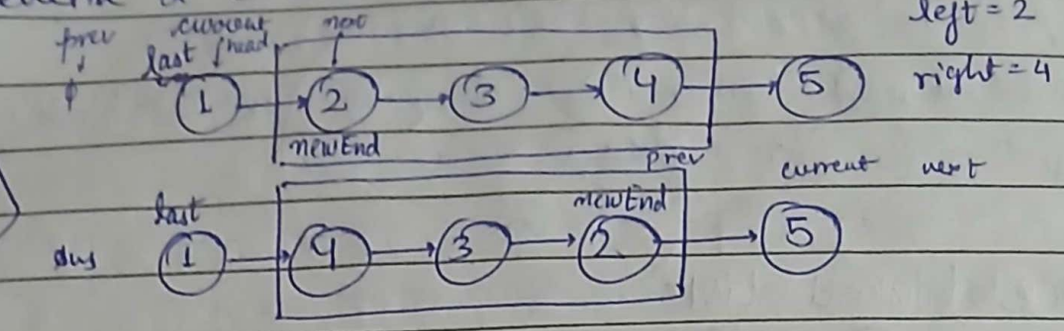
tail.next = ϕ

this is happening when you are coming out of rec call. ∴ traverse tail till end.

Teacher's Signature.....

Date | Page No.

18. Reverse a Linked list II



last is reverse hone wale jaga se just pehle wala

concept ⇒ → first of all if left = right → return head.

→ Now to reach (by skipping) at left then do reversing.

∴ skipping first left − 1 nodes

* → for i=0 to current != null && i < left − 1 ; i++

∴ prev = current (initialising for reverse)

current = current.next

Now → Define last & newEnd & next (now)

prev (after skipping) — current — otherwise error

last = prev

→ Now reverse b/w left & right

condition → current != null & i < right − left + 1

→ Note:- if last != null (must) ∴ last.next = prev | else, head = prev

Explain (very imp.)

→ newEnd.next = current

return head;

19. Palindrome Linked list. 1 → 2 → 2 → 1 ∴ true

Concept ⇒ → get middle node

→ reverse the list from mid & assign it as second head

→ while head != ϕ && head.second != null

if (head.value != " ")

break;

traverse head & secondhead

→ now reverse that list from middle node

Hence, if both heads reached till null either ∴ true

else false.

Teacher's Signature.....

1221

abba

Date | Page No.

11. Reorder List

temp → use for storing hs

Q. → hf 1 → 2 → 3 → 4 → 5 → 6 → 7 ← hs

hf 1 → 2 → 3 → 4 → 7 → 6 → 5 ← hs

Qu. → 1 → 7 → 2 → 6 → 3 → 5 → 4

Concept :→ → if head = null or head.next = null (only 1 node) ∴ return
→ get mid node
→ hs (second head) → reversed list from mid's head
→ hf (first head)

→ Now rearrange: till hf & hs != null
→ create a temp node next of hf then
→ put hf.next = hs → hf = temp → temp = hs.next → hs.next = hf → hs = temp

rearranged 1 | rearranged 2

→ put tail at last as if (hf != null)
i.e. hs reached null first
hf.next = null;

15. Reverse Nodes in K-Group

newLast | prevLast | current | next

1 → 2 → 3 → 4 → 5 → 6 → 7

K=3

3 → 2 → 1 → 6 → 5 → 4 → 7

Steps
Concept → → first if all K<=1 or head == null then return head
→ ~~ListNode~~ get current & prev pointers at head & null resp.
→ get length of list → count = length/K → socho K must be > length
→ till count > 0 [→ last = prev , newEnd = current, againe 3rd ptr next = current.next
→ for loop till i<K && current != null → ① assigning backwards current.next = prev
② Now continue traversing : prev = current, current = next, (next = next.next → nullptr exception)
→ if last != null → last.next = prev , else head = prev
newEnd.next = current , prev = newEnd
count-- ;]

reverse LL

return head ✓

Teacher's Signature.....

Date
Page No.

Concept → till count > 0 → count = length/k ∵ k jo group ke wo atleast length ke hyado ho

→ point last and new tail → revise (rev.LL2) (Q.10)

reverse wale block ke pehla (prev) — maya last jo obv. reverse se pehle first (current) pe hoga

last

also make next node

→ Do reverse Linked list 2 till i<k and decrease count

→ return head.

(14,15) → rev alt. k nodes & rotate list } → practice yourself