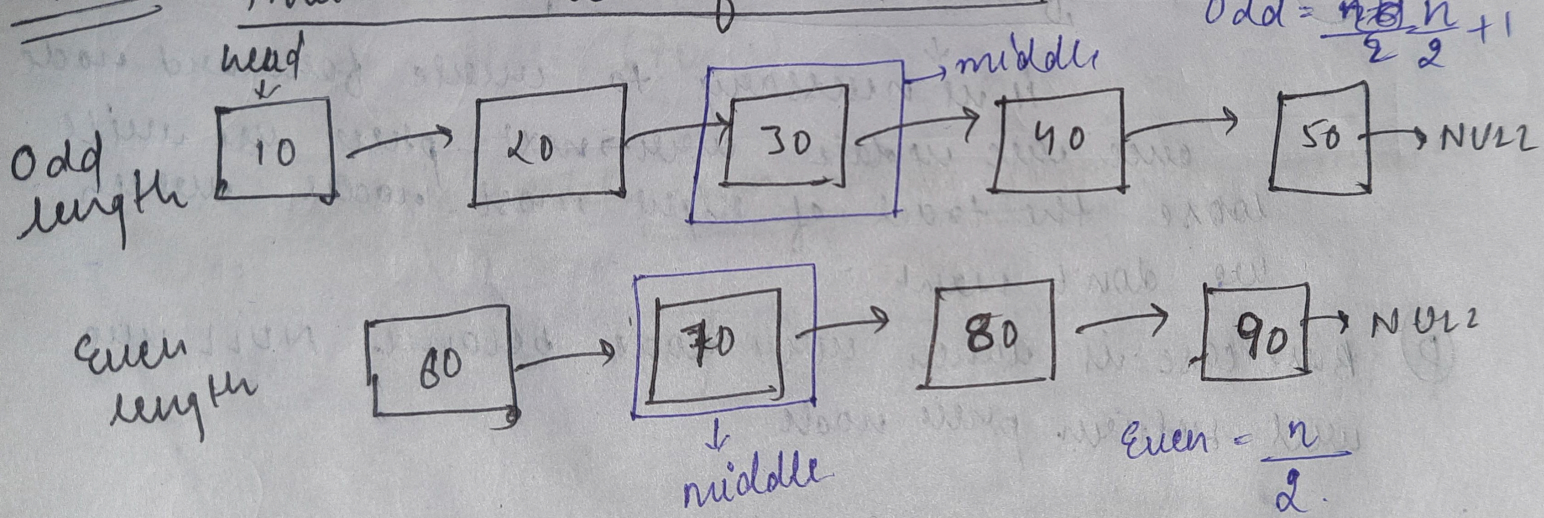
Ques: 2 Find the middle of Linked List:



Odd = $\frac{n}{2} + 1$

Even = $\frac{n}{2}$

Approach 1

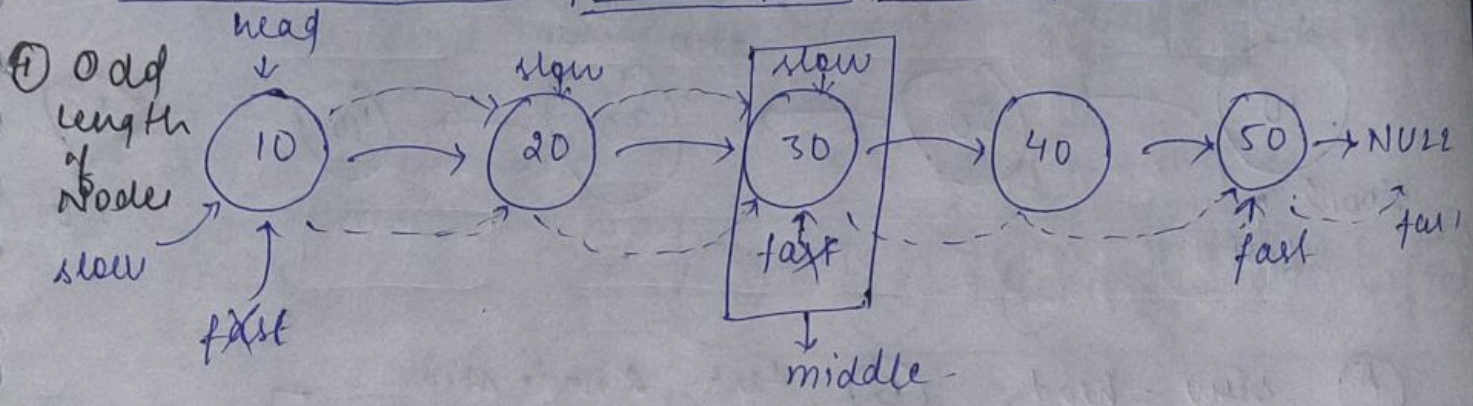
- length → O(n) → ✗
- length → even → $\frac{length}{2}$
  - odd → $\frac{length}{2} + 1$

→ O(n)

Total T.C: O(n)

→ Approach 2

Tortoise Algorithm / slow & fast pointer approach / 2-pointer

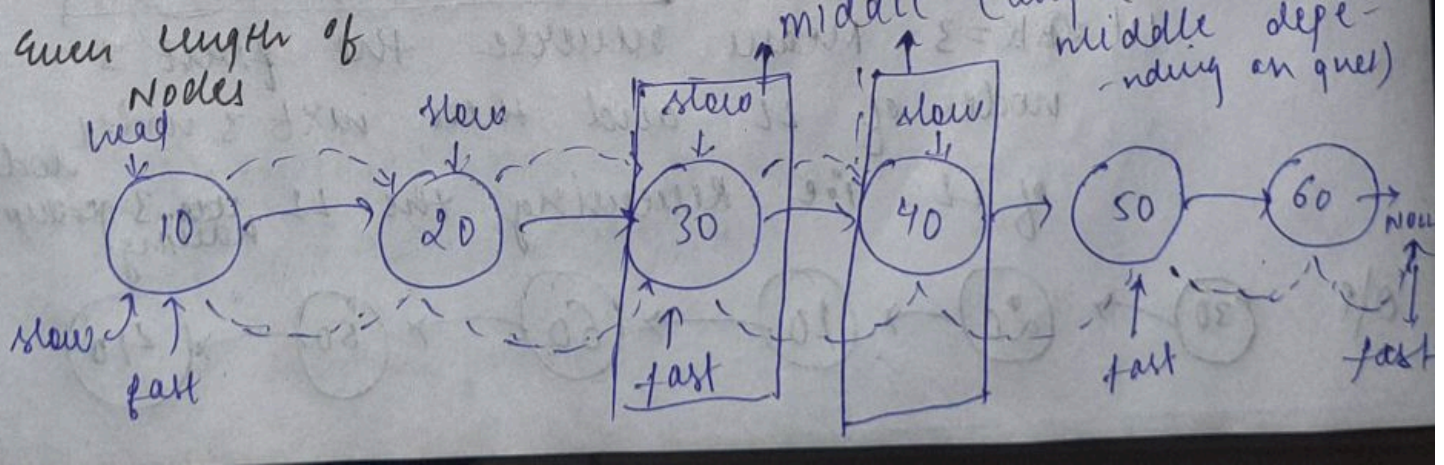


(step 1) slow & fast pointer point to the first node i.e head

(step 2) slow always takes 1 step & fast always takes 2 steps steps means move 1 node forward and 2 node forward.

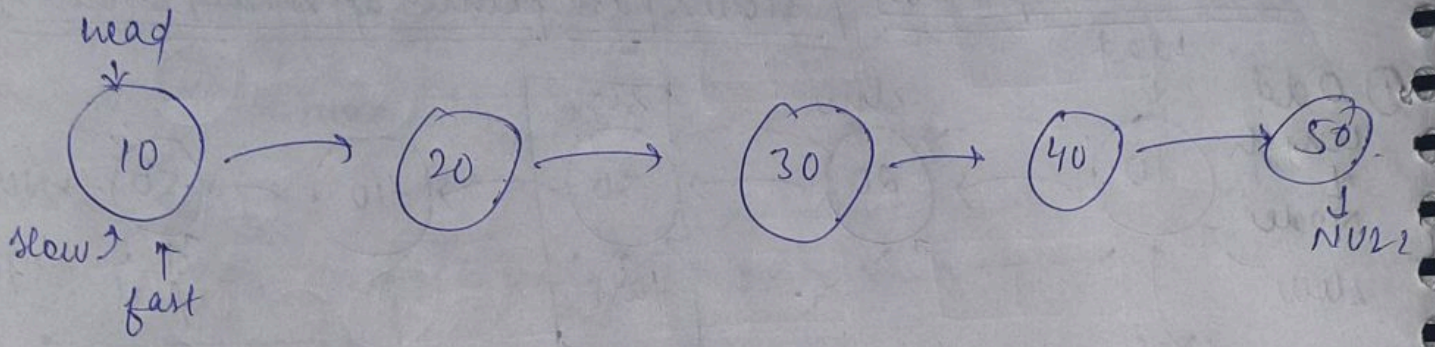
(step 3) If fast node is unable to move 2 nodes ahead then only slow node will move 1 node forward.

② even length of Nodes



Imp ① same as odd no. of nodes

when fast node reach to NULL for even no. of nodes then it's our choice whether to move slow pointer one nodes ahead or to and say that particular node to be middle node or consider the current node where slow pointer point to be the middle node that totally depends on ques



(A) slow = head
fast = head

(B) fast = 2 node step
if fast is able to take 2 step
then slow = 1 step
→ do this i.e traverse the complete LL

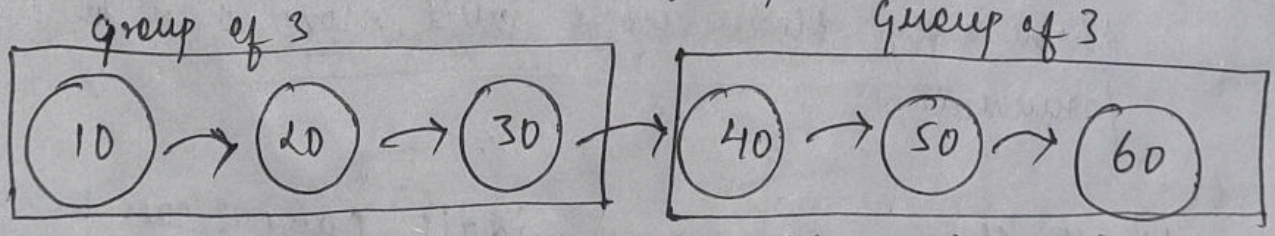
(C) return slow when fast unable to take 2 steps. until slow & fast become NULL

Ques:3 Imp V.V.V

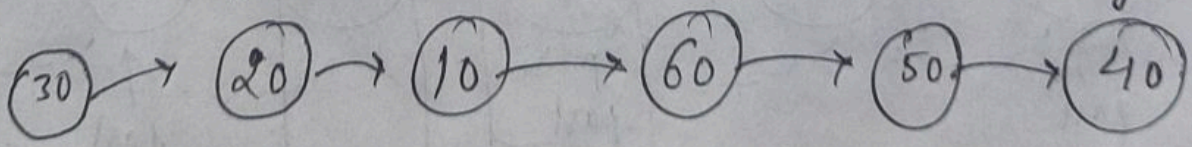
K-group reverse a Linked List

OR

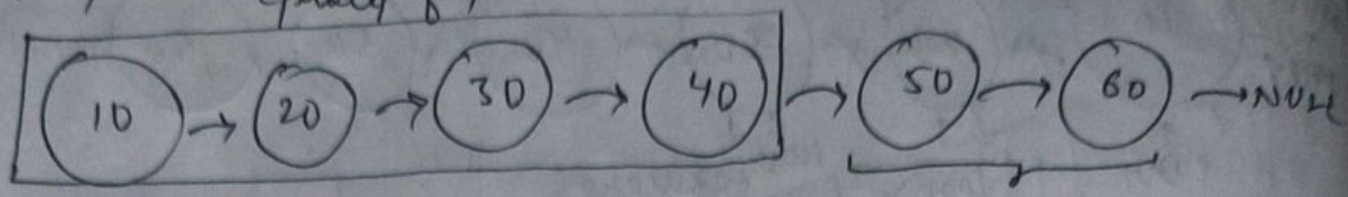
Reverse a LL in K-groups



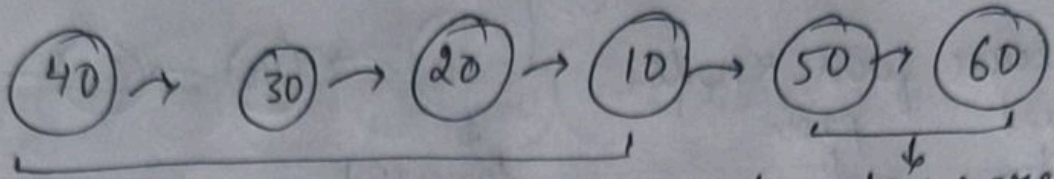
Let K=3 Mean reverse the first 3 nodes of LL and then next 3 nodes of LL i.e Reversing the LL having 3 nodes in group



if K=4 Group of 4

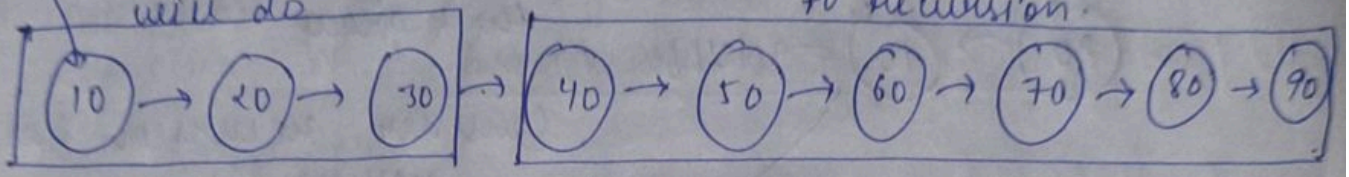
I/P 

can't form group of 4 so will print as it is

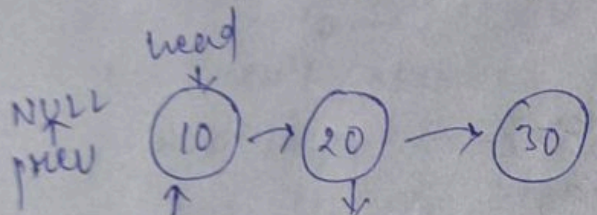
O/P 

first 4 nodes are reversed

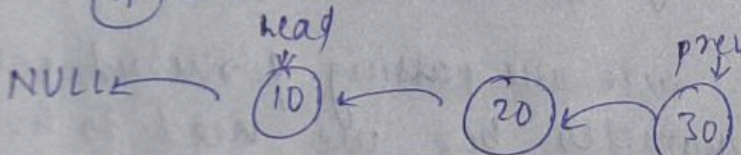
last 2 are same as in input.

⇒ 

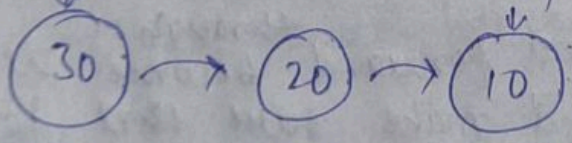
If K=3 mean reverse the nodes in the group of 3



(A) we will reverse K nodes.



OR

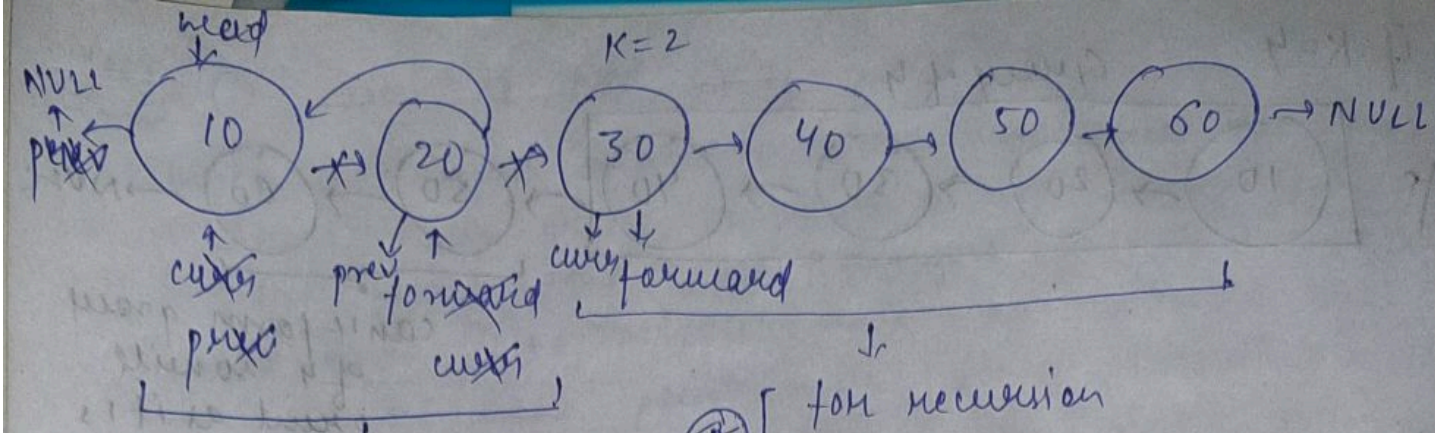


(B) Recursion will return something we will link head→next = Recursion

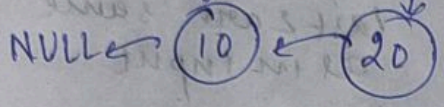
Steps (A) Reverse first K nodes

(B) head→next = Recursion

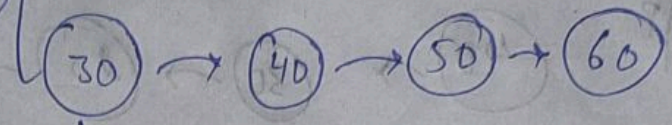
(C) return prev.



(A) reverse first K nodes



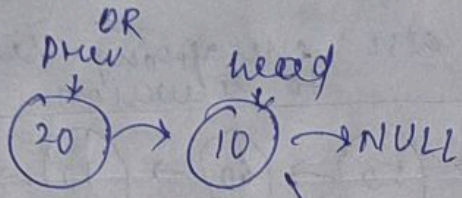
(B) for recursion



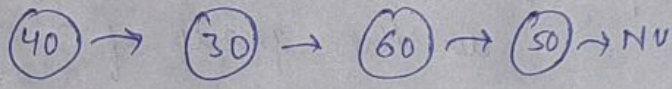
forward

↓

it will become the head of upcoming nodes to reverse the K group nodes.



Consider recursion has reversed it



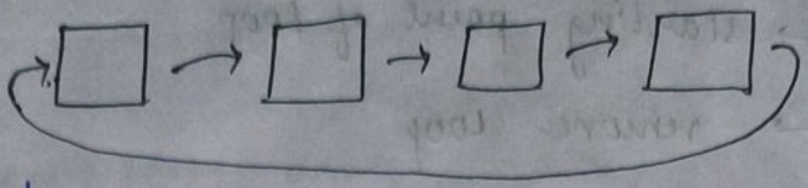
to join the reverse nodes with recursion as i.e recursion

head→next = recursion

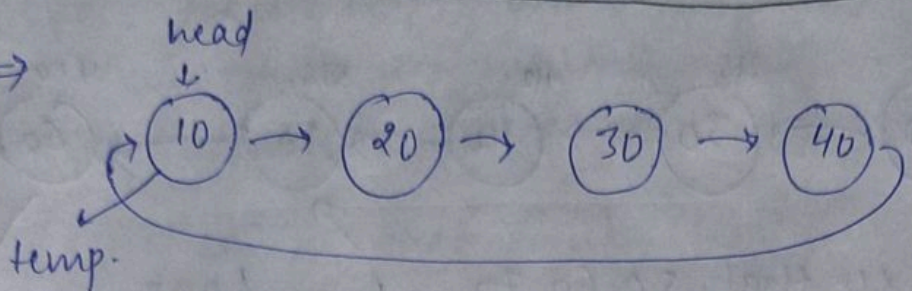
(C) to return the head of updated LL.

Note → for recursion we are passing forward pointer as a head in parameter bez we need to have access of further nodes so that recursion we can reverse K nodes and forward through become head of upcoming nodes and make sure that forward should not be NULL

Ques:4 I/P: Linked List to check it is circular or not return O/P True / False



⇒



(A) temp = head.

(B) Now check from temp→next ~~head~~ i.e from 20 that whether it is head or Not.

i.e 20 → head ✗
30 → head ✗
40 → head ✗
10 → head ✓ [The LL is circular]

We aren't starting from 10 because initial we have ~~mid~~temp = head so if we check whether 10 is head or not it will become useless.

(C) return true/false if it is CLL.

Approaches

(1) 1 pointer approach [temp pointer]

(2) slow/fast pointer approach

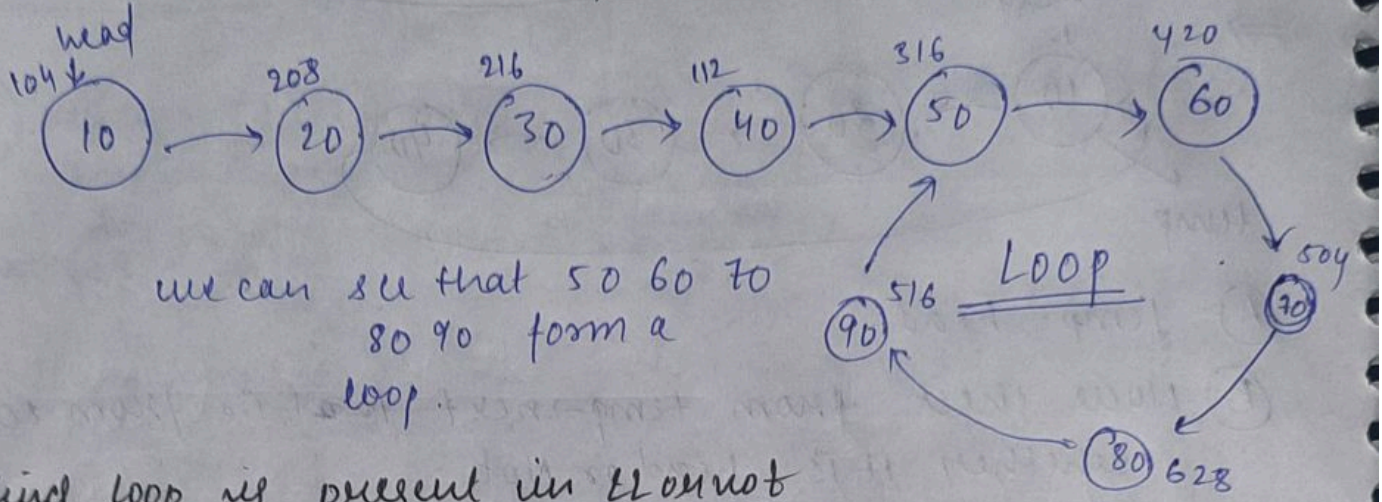
(3) mapping

Do Yourself

Ques:5 Detect and Delete Loop

V.V.V Imp

Different variations of Question
- check loop is present in LL or not
- starting point of Loop
- remove Loop



we can see that 50 60 70 80 90 form a loop.

① find loop is present in LL or not

→ Approach 1 we can solve it with map

map <Node*, bool> map.
- Node* → address
- bool → T/F

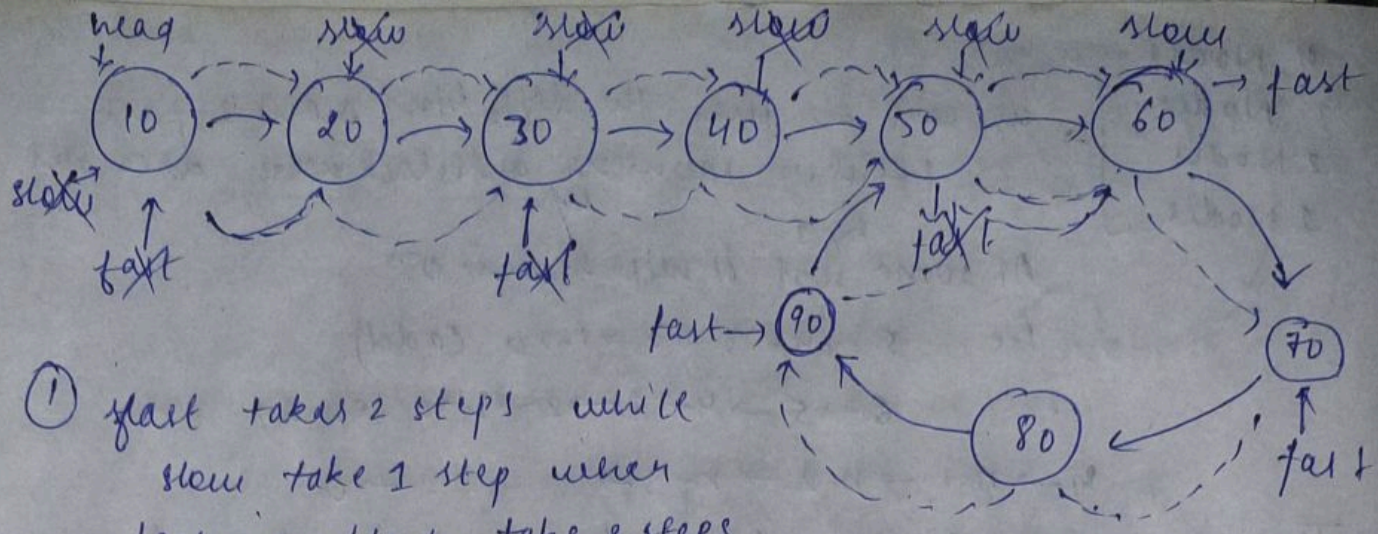
Address	value
104	true
208	true
216	true
112	true
316	true
420	true
504	true
628	true
516	true
316 → Again 316 is present at above	

Here 316 is true already present [we can say that loop is present]

→ Approach 2

Floyd Cycle Detection (FCD) Algorithm can be used to check whether loop is present in LL or not.

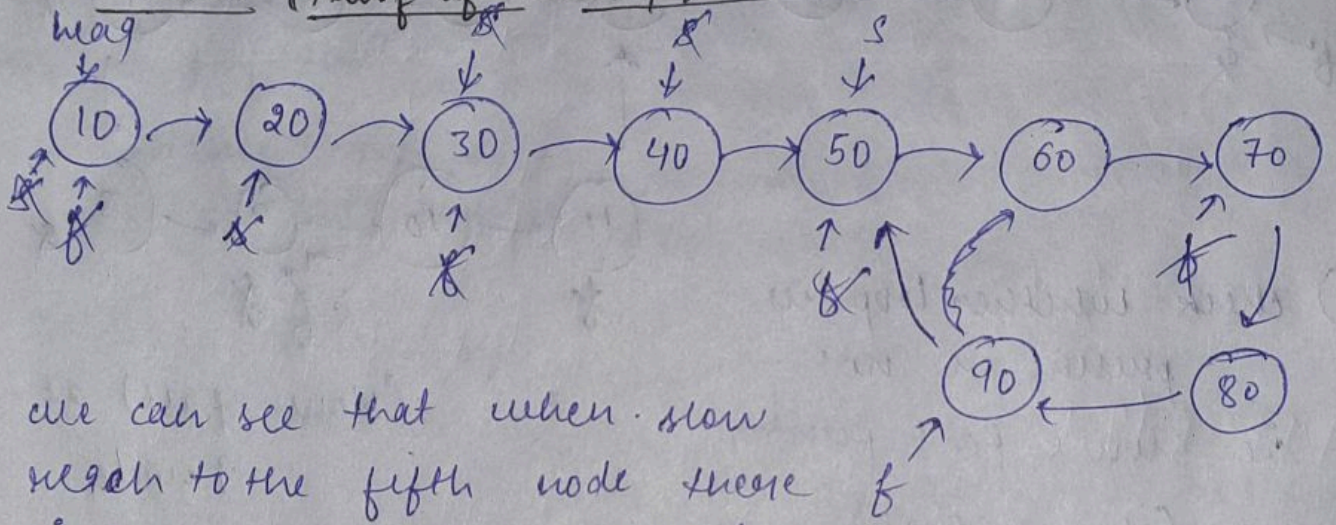
In this we use slow / fast pointers.



(1) fast takes 2 steps while slow take 1 step when fast is able to take 2 steps.

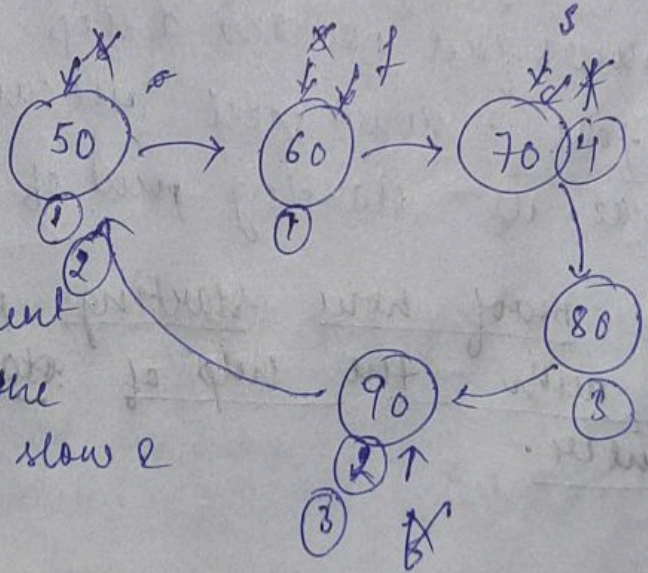
so we can see that when slow == fast → loop present.
else fast == NULL → loop absent.

→ Mathematically how it is working to check loop is present or Not (Proof of slow / fast Pointer)



we can see that when slow reach to the fifth node there is 100% possibility that fast node must be inside loop.

consider



slow is at 50 & fast is at 70. Now count anticlockwise the diffrence b/w slow & fast pointer

4 Nodes.
3 Nodes.
2 Nodes
1 Node

4 Nodes
3 Nodes
2 Nodes
1 Node
→ we can see that the diff b/w slow & fast pointers counting anticlockwise decreases by 1.

At some sort it will become 0.

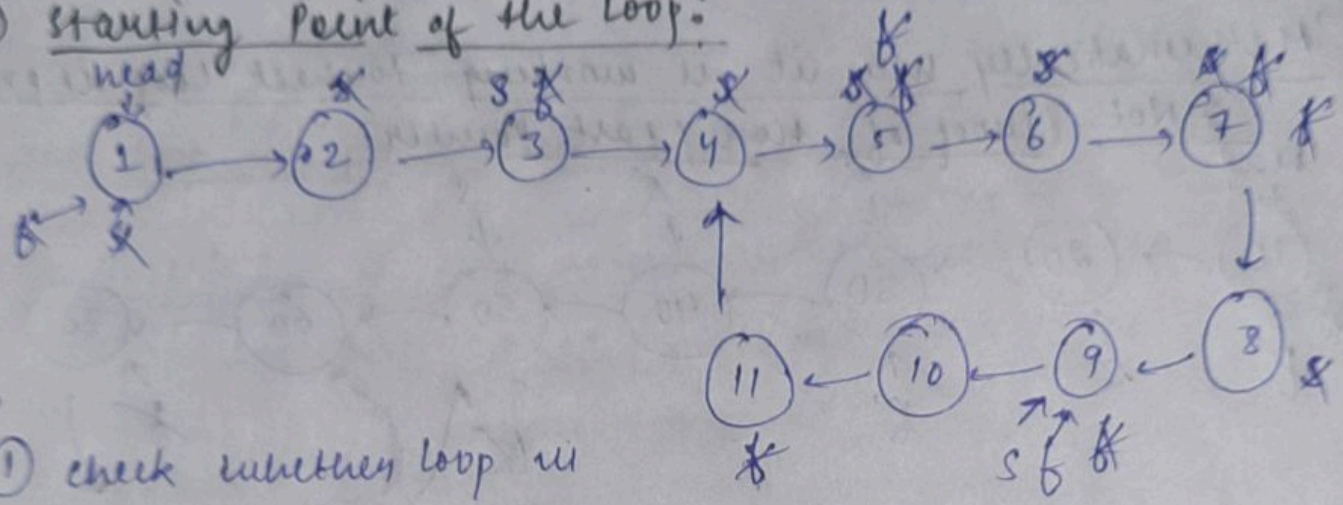
i.e $5 \to 4 \to 3 \to 2 \to 1 \to 0$ (odd)

$6 \to 5 \to 4 \to 3 \to 2 \to 1 \to 0$ (even)

$n \to n-1 \to n-2 \to n-3 \to \dots \to 0$

Thus it mathematically how loop is being checked with the help of slow/fast pointer.

(2) Starting Point of the loop:



step

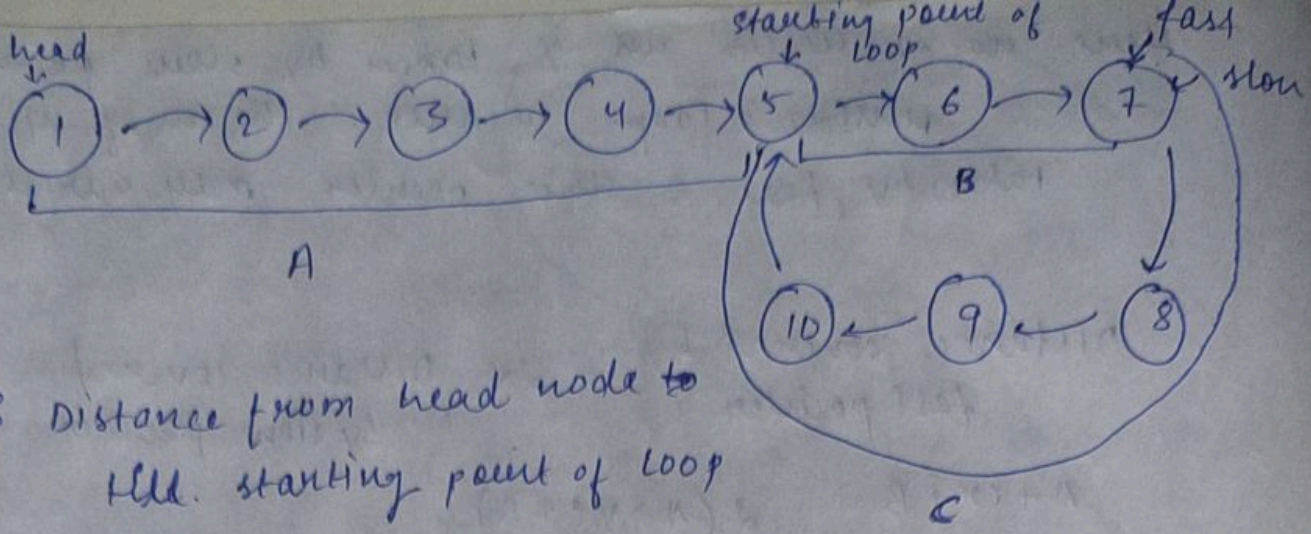
(1) check whether loop is present or not.
i.e slow & fast pointers (slow == fast)

(slow == fast) at 9 node

(2) assign slow = head (when slow == fast become)

(3) Now move slow & fast pointers 1 step till slow != fast when (fast == slow) meet we can say that, it starting point of loop.

* mathematically proof how starting point of a loop is found with the help of slow and fast pointers.



A: Distance from head node to the starting point of loop

B: Distance from starting point of loop till where fast & slow meet

C: Distance of complete loop in a linked list.

As fast takes 2 step whereas slow take 1 step. we can say that

Distance covered by fast pointer = 2 * Distance covered by slow pointer

Distance covered by fast pointer = A + xC + B.

- As fast pointer must have started from head till starting point of loop.
- Here x C: no. of cycles fast pointer takes inside a loop
- As after covering x cycles the distance B will be covered

A + xC + B [Distance covered by fast pointer]

Distance covered by slow pointer = A + yC + B.

Here no. of cycles is y taken by slow pointer inside a loop. because the no. of cycles taken by fast & slow pointer may varies.

Distance covered by fast pointer = 2x Distance covered by slow pointer

$$A + cx + B = 2(A + yc + B)$$

$$A + xc + B = 2A + 2yc + 2B$$

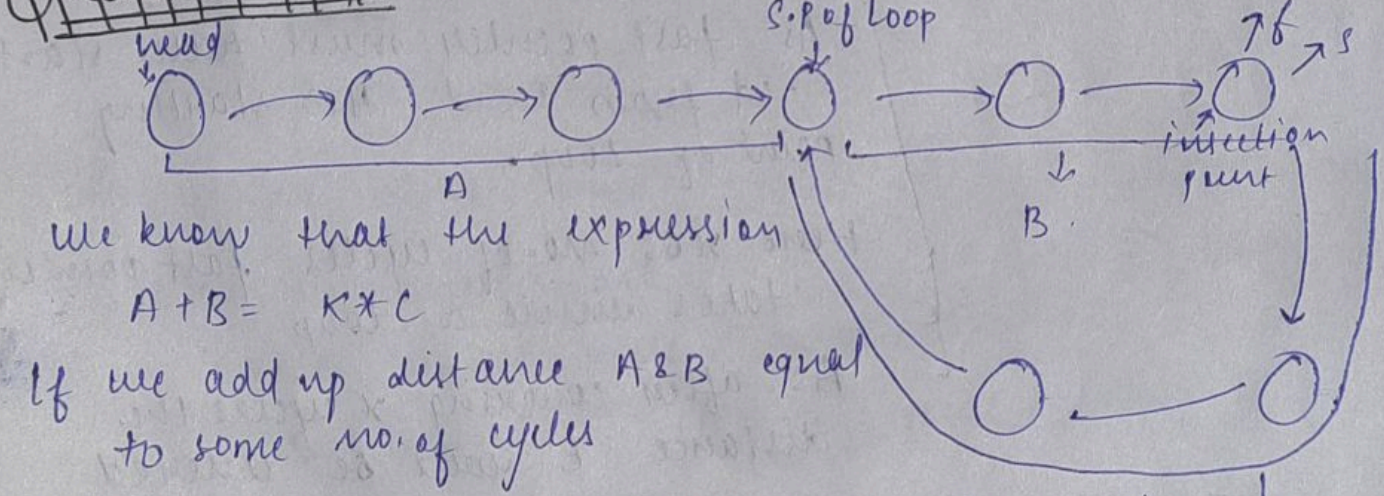
$$(x - 2y)c = A + B$$

let $K = x - 2y$

$$\boxed{A + B = K * C}$$

so we can see that if we add up distance A and B. means to reach where fast & slow pointer meet. there is requirement some no. of cycles taken by both fast & slow pointer inside a loop.

2) Remove loop:



We know that the expression

$A + B = K * C$

If we add up distance A & B equal to some no. of cycles

Now B is: Distance from S.P till intersection

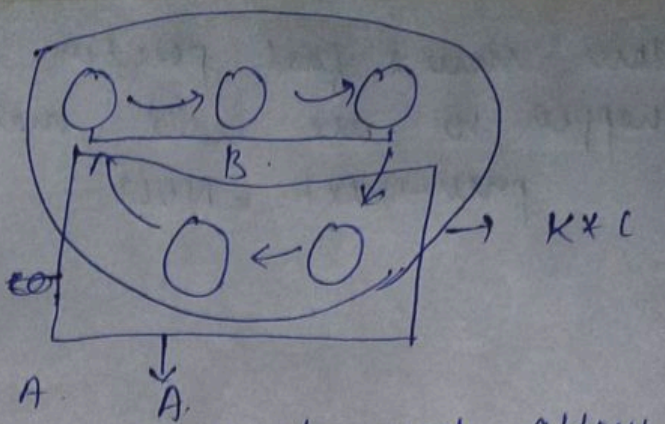
If C: distance of whole loop

We can say that

Then half cycle will become

$A + B = K * C$

$A = K * C - B$

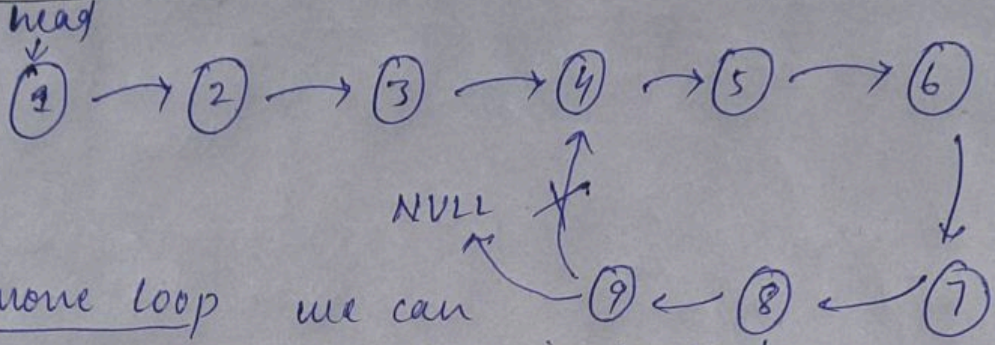


so lower part of cycle become distance A

therefore we point slow to head. and allow slow & fast pointer to take 1 step

∴ Through this mathematically fast & slow will meet at starting point

③ Remove loop:

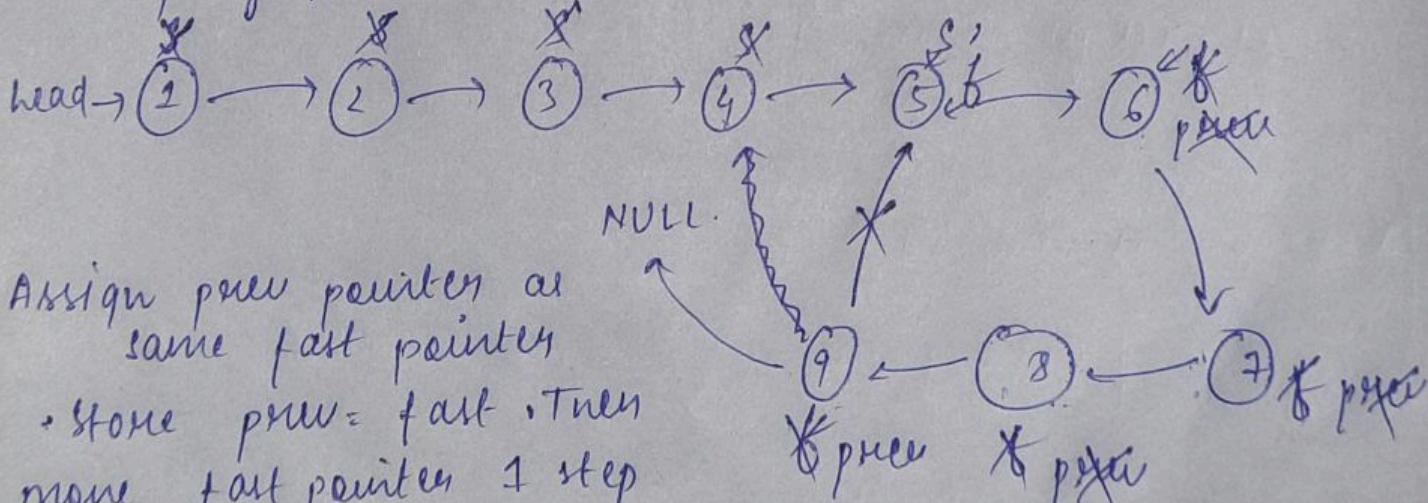


Remove loop we can see that loop is formed

→ To remove loop we will point out last node to the NULL.

→ This can be done through previous pointer.

Assigning previous pointer ~~next~~ same as to fast pointer.



Assign prev pointer as same fast pointer

• Store prev = fast. Then move fast pointer 1 step ahead.