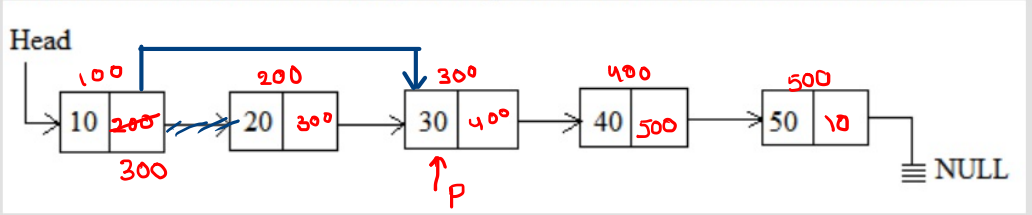
Day-13 [Linked List-2]

1. Given a linked list L with head pointing to the first node of L, shown below:



What is the output when the following sequence of operations applied on the given linked list?

P = 300

P is a node pointer

head → next → next → data

100 → 300 → 400

(i) P = head → next → next;

100, 200, 300

(ii) head → next = P;

(iii) printf("%d", head → next → next→ data);

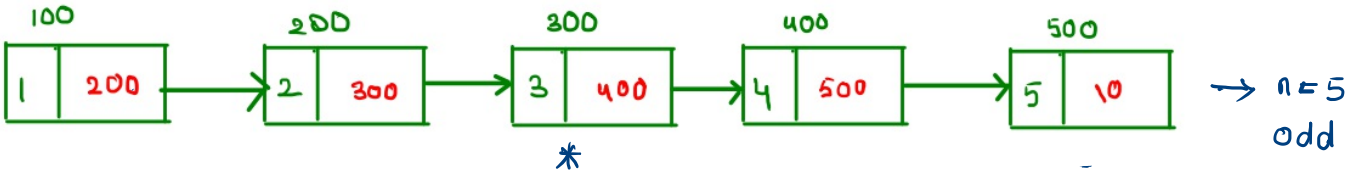
The output of the following code is 40

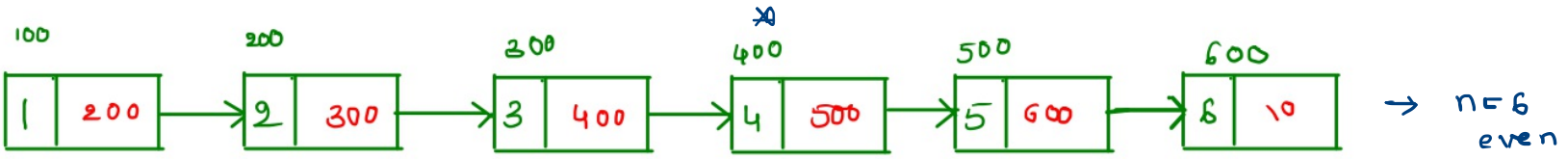
(Marks: 0.00)

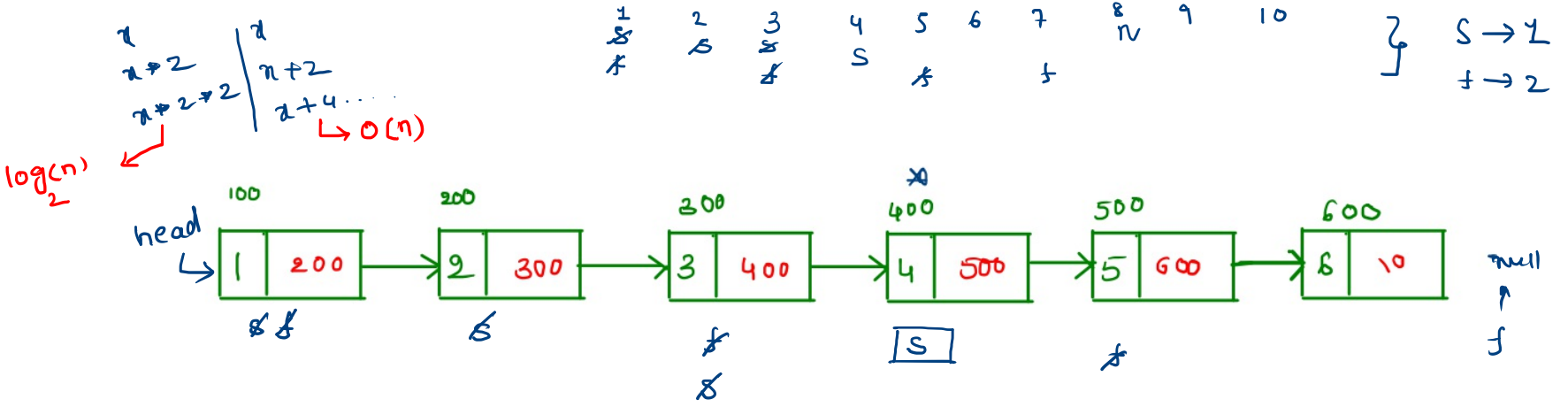
① Find the Middle Node in the Single Linked List

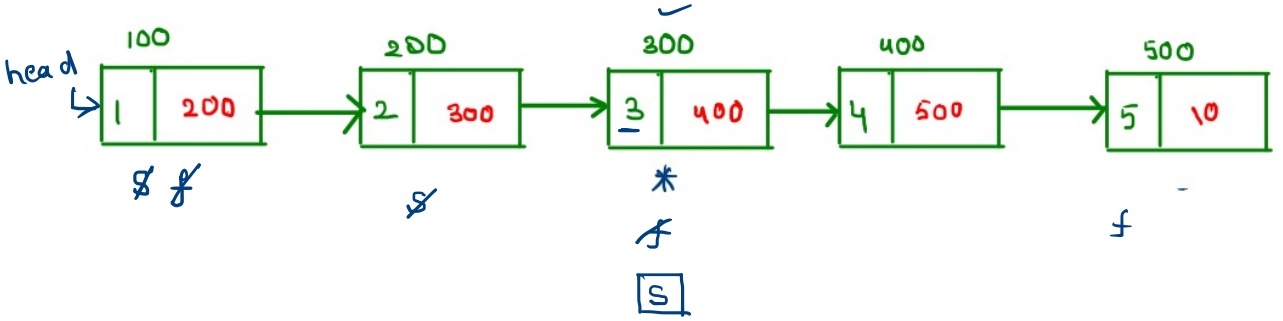
T.C : O(n)
S.C : O(1)

1) find length of SLL → n
2) point to $\frac{n}{2}^{th}+1$ node → $\frac{n}{2}$






n
n+2
n+2+2
log(n) 2
n
n+2
n+4 . . . .
O(n)
1 2 3 4 5 6 7 8 9 10
N
S → 4
f → 2
100
200
300
400
500
600
head
1 200
2 300
3 400
4 500
5 600
6 10
null
S


head
100
1
200
200
2
300
300
3
400
400
4
500
500
5
10

→ O(n)

```
function printMiddle(Node head)
{
    Node slow_ptr = head;
    Node fast_ptr = head;

    while (fast_ptr != null && fast_ptr.next != null)
    {
        fast_ptr = fast_ptr.next.next;
        slow_ptr = slow_ptr.next;
    }
    print(slow_prt.data)

}
```

case 1 → LL empty

Case 2 → n → [1|10] → 1 → $\frac{2}{2}$ = 1

s f

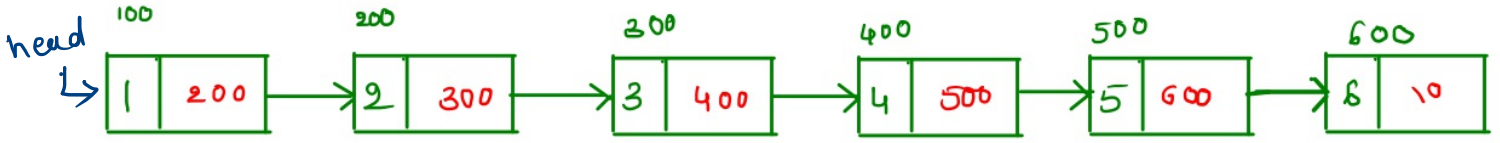
case 3 → head → [1] → [2]

s f

s

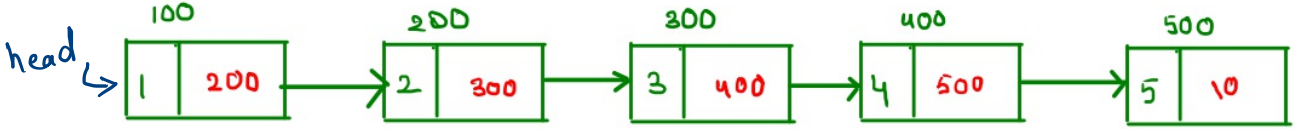
2

(2) Find the Kth Node from the End of Single Linked List



K = 3 ⇒ 4

K = 5 ⇒ 2

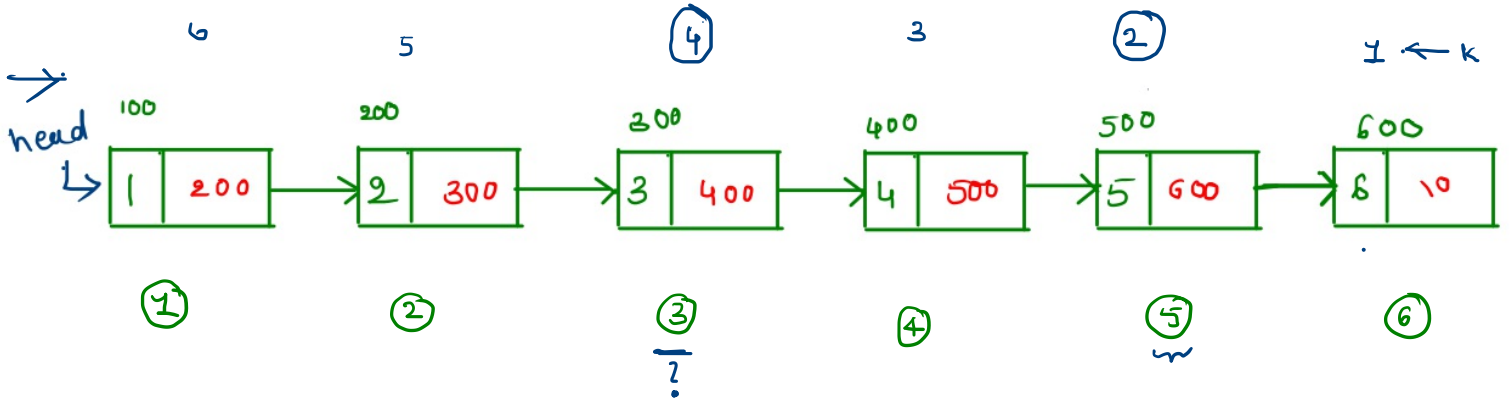


K = 4 ⇒ 2

K = 6 ⇒ -1

Not exists

K ≤ n ✓



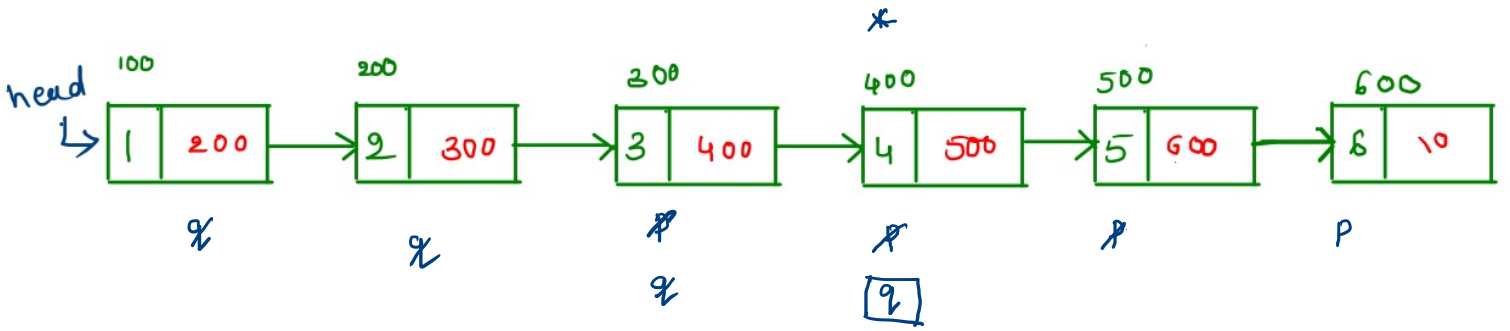
Note:-

> k^{th} from the end $\cong$ $(n-k+1)^{th}$ node from beginning

* Find the Length and print [length-k+1] value

```
printNthFromLast(Node head,k)
    {
        len = 0;
        Node temp = head;

      * // 1) count the number of nodes in Linked List
        while (temp != null) {
            temp = temp.next;
            len++;
        }

        // check if value of n is not more than length of
        // the linked list
        if (len < k)
            return;

        temp = head;

        // 2) get the (len-n+1)th node from the beginning
        for(i = 1; i < len - k + 1; i++)
            temp = temp.next;

        print(temp.data);
    }
```

$k \leq n \quad \Rightarrow k > n$



o/p

k = 2 ⇒ 5

→ 1) Beg of kth node ✓

2)

k = 3 ⇒ 4

2-pointer Approach

1) take two pointers p and q of type Node

2) put one pointer at beg of kth node

3) p=head, q=head

4) for(int count=1;count<k && p!=null;count++)
   {
      p=p.next
   }

Helps to point p to Beg of k^{th} node.

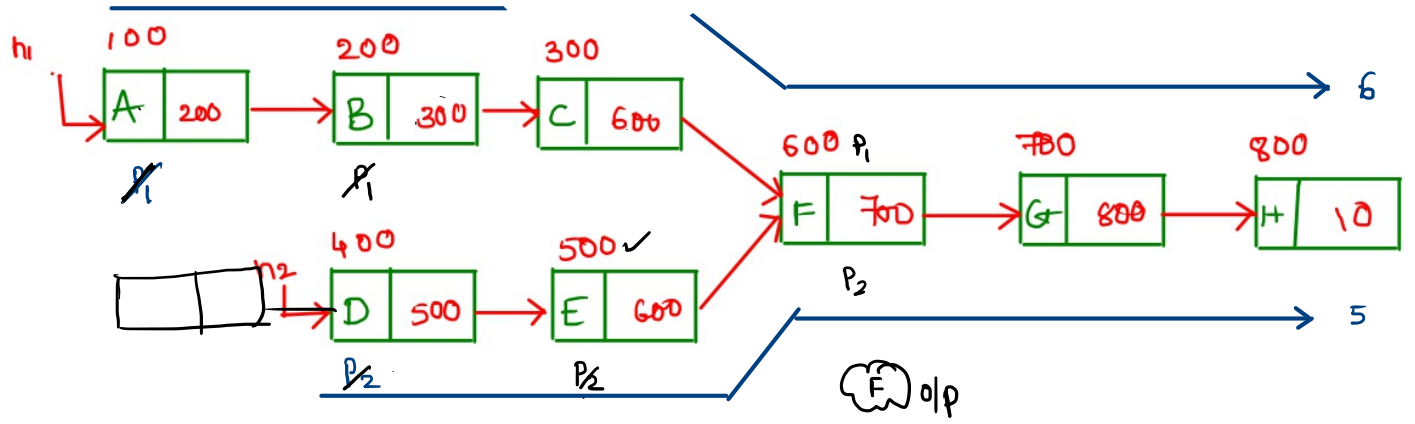
5) if(p==null)
      return

6) while(p.next!=null)
   {
      p=p.next
      q=q.next
   }

Same pace.

7) return q;

③ Find the Intersection point of two Single Linked List of type [Y]



step1 :- → $l_1 = 6$. [h1]

$l_2 = 5$. [h2]

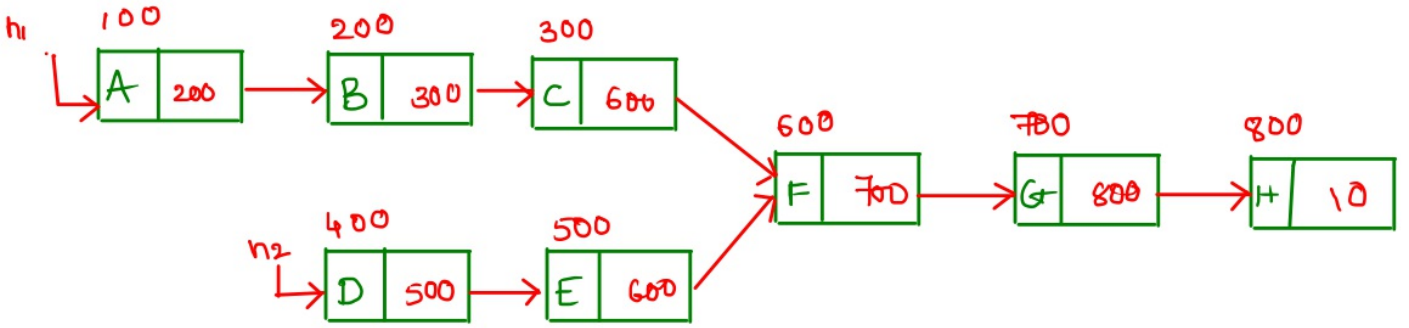
step2 :

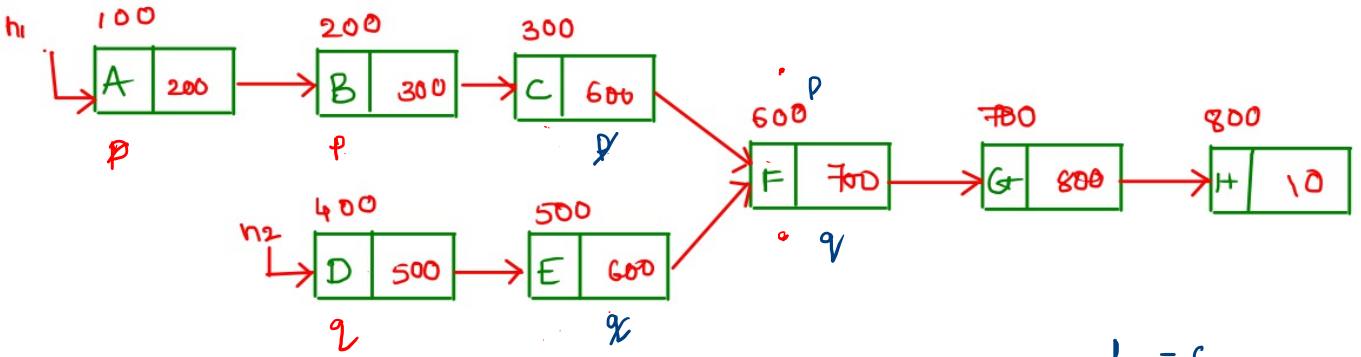
diff = 1 ✓

↳

* step3 :-

step4 :-


h1
100
A
200
200
B
300
300
C
600
400
h2
D
500
500
E
600
600
F
700
700
G
800
800
H
10




```
int l1 = length(head1)
int l2 = length(head2)
int diff= Math.abs(l1-l2)
int result = l1>l2 ? find(diff,head1,head2):find(diff,head2,head1)

int find(int diff, Node p, Node q)
{
     int count=0;
    for(count=0;count<diff && p!=null; count++)
    {
          p=p.next
    }
     while(p!=q)
     {
          p=p.next
          q=q.next
     }
     return p.data;
}
```

$l_1 = 6$

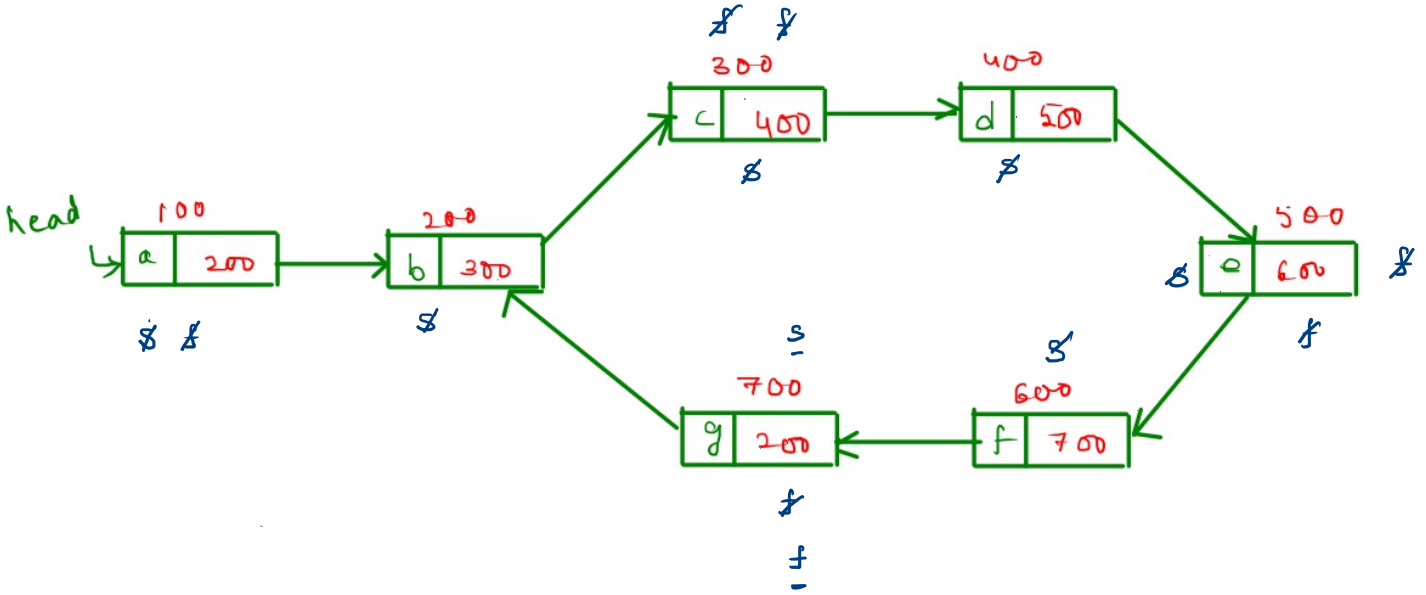
$l_2 = 5$

diff = 1

$l_1 > l_2$

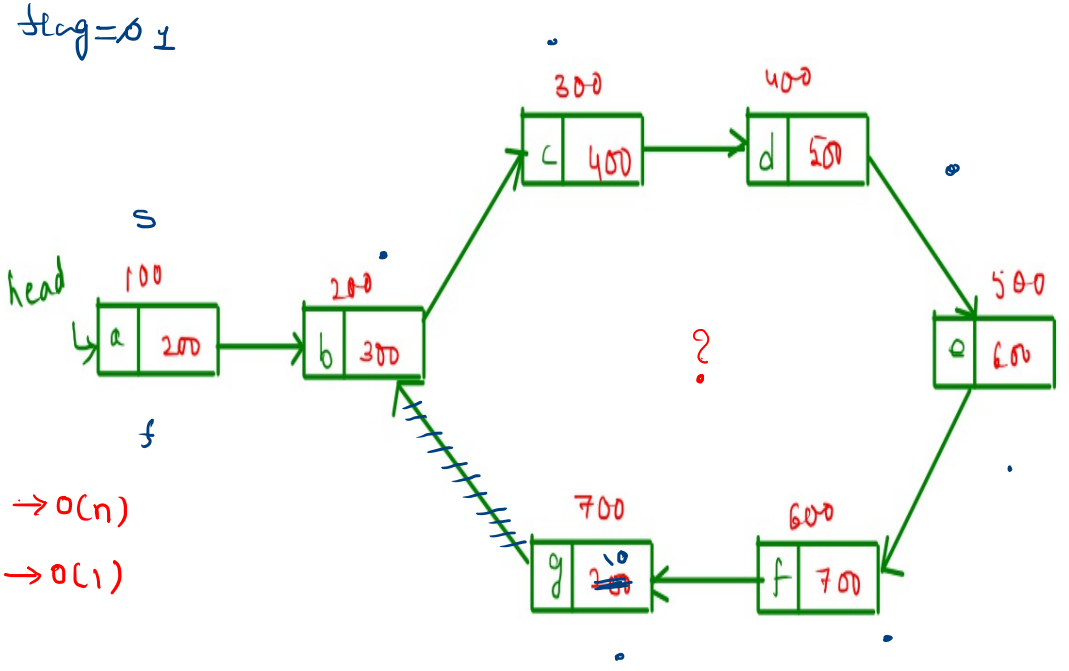
EA

4. Find the loop in a SLL

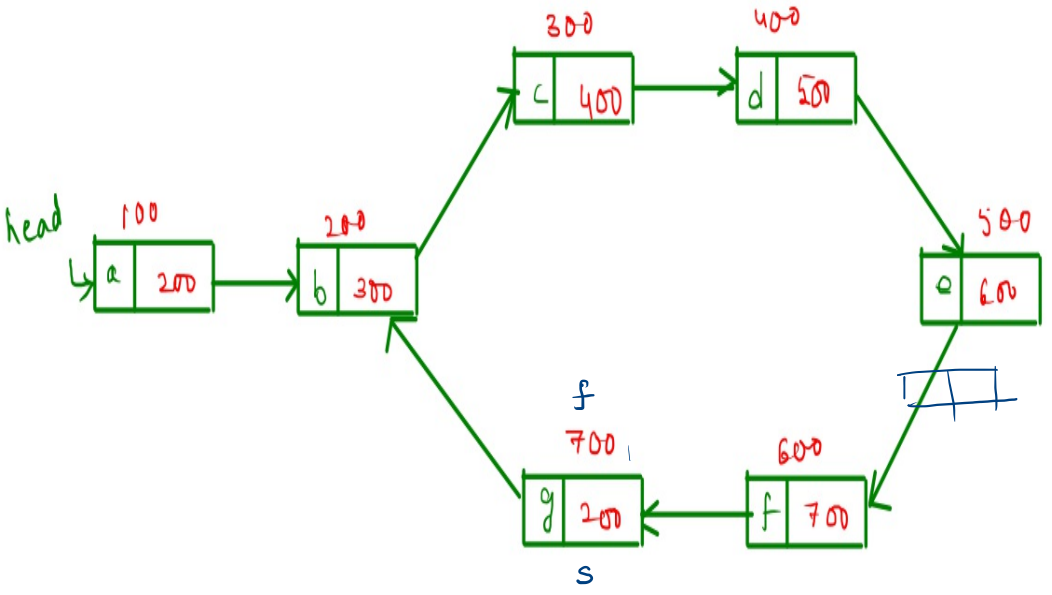


* Floyd-cycle Algo

```
//loop in SLL
function detectLoop(Node head)
{
        Node slow = head, fast= head, flag = 0;
      * while (slow!=null && fast!= null &&fast.next != null)
        {
            slow= slow.next;
            fast= fast.next.next;
            if (slow == fast)
            {
                flag = 1;
                break;
            }
        }
    → if(flag == 1)
            print("loop is found")
        else
            print("Loop is not found");
}
```



```
detectAndRemoveLoop(Node head)
{
        Node slow = head, fast= head, flag = 0;
        while (slow=null && fast= null &&fast.next != null)
        {
            slow= slow.next;
            fast= fast.next.next;
            if (slow== fast)
            {
                removeLoop(slow,head)
                return 1 ;
            }
        }
        return 0;
 }
```



```
removeLoop(Node loop,Node head)
{
    Node p1=loop
    Node p2=loop
    //count number of nodes in loop
    k=1;
    while(p1.next!=p2)
    {
        p1=p1.next
        k++
    }
    p1=head // fix one ptr to head
    //fix other ptr to k nodes after head
    p2=head
    for(i=0;i<k;i++)
    {
       p2=p2.next
    }
    /* move both ptrs at same pace,
       so that they will meet at loop starting */
    while(p1!=p2)
    {
         p1=p1.next
         p2=p2.next
     }
     // Get one ptr to last node
     while(p2.next!=p1)
     {
         p2=p2.next
      }
      p2.next=null;
}
```