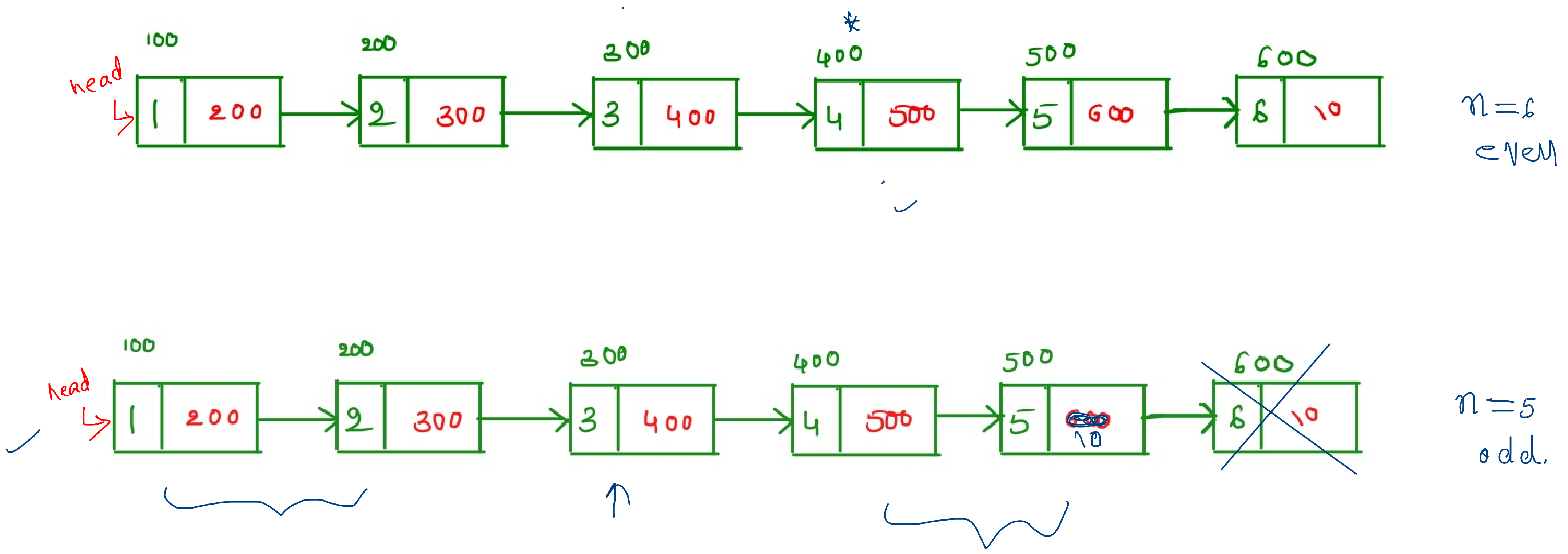
① Print the middle node of a SLL



✓ AP_1 :-

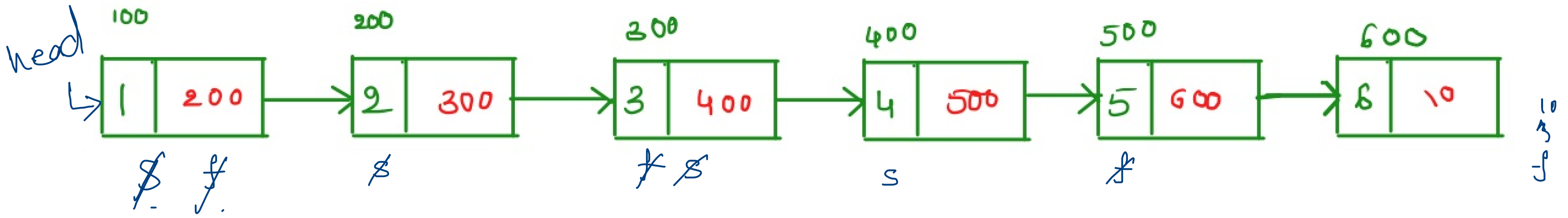
- 1. count # of nodes (let len) → $\underline{n}$ time.

2. print($(len/2)^{th}$) node → $\underline{n/2}$ times

$n + \cancel{n/2} \Rightarrow O(n)$

✓ AP_2

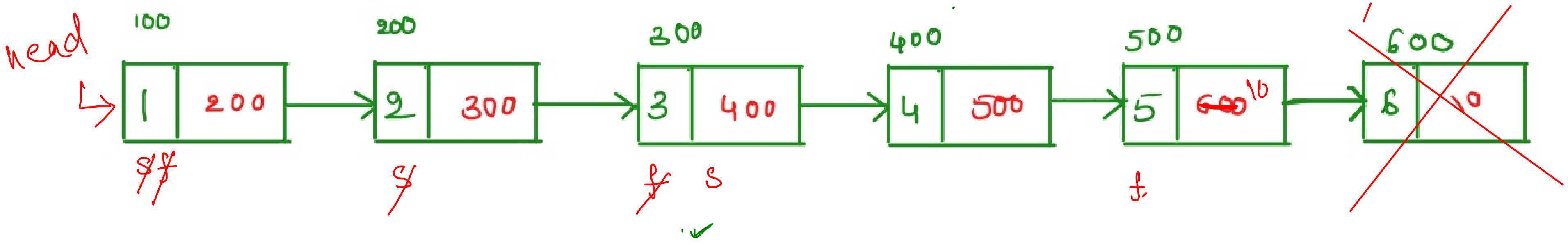
2-ptr tech, s, f



s = s.next

f = f.next.next

} (s.data)



```
function printMiddle(head)
{
      var slow_ptr = head;
      var fast_ptr = head;
        if (head != null)
        {
            while (fast_ptr != null && fast_ptr.next != null)
            {
                fast_ptr = fast_ptr.next.next; ✓
                slow_ptr = slow_ptr.next; ✓
            }
            console.log(slow_ptr.data)
        }
}
```

✓1 2

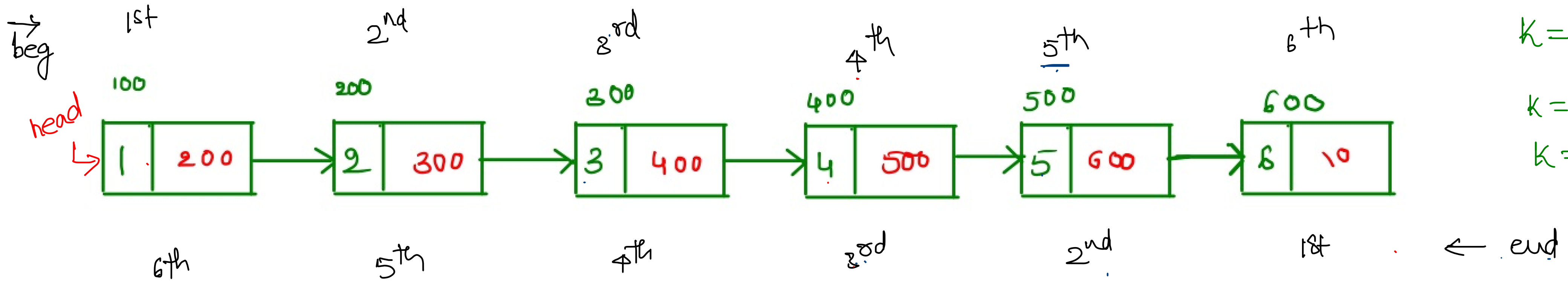
even ✓ odd

⇒O(n)

I/P 1. SLL ✓
2. K ✓

O/P
Kth node from end

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



K=2 → 5
K=3 → 4
K=4 → 3

n=6

from end	beg		
• K=2 ✓	5th	?	⇒ 6 − 2 + 1
• K=3 ✓	4th	?	⇒ 6 − 3 + 1
K = 4	3rd	?	⇒ 6 − 4 + 1

n = 6 ✓
↳ we've to find out.

Kth node from ending ≅ $\underbrace{(n - k + 1)}_{}$th node from begining

Ap1

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

n

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

          temp = head;

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

          print(temp.data);
     }
```

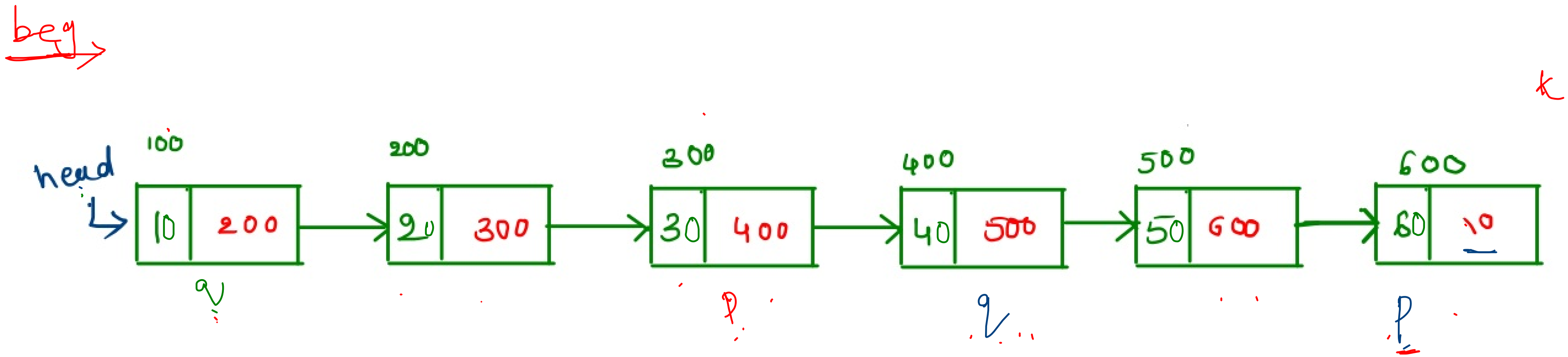
length of SLL

len = 6 ✓

k = 7 ✓ ⇒

len < k

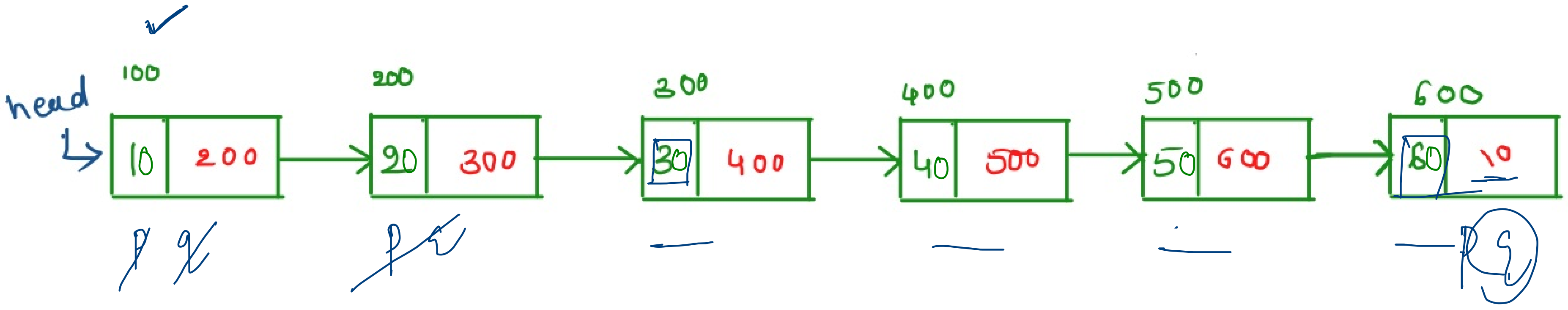
n - 1 + 1



k = 3 ⇒ 40 (o/p)

```
1. p = head,   q = head.

2. put any one-ptr @ exactly Kth node from beg.
                 let p

   while( p.next != null)
3. { move both ptrs @ same pace. ⇒ p = p.next
                                    q = q.next
   }

4. return q.data
```

K = 1 ⇒ 60



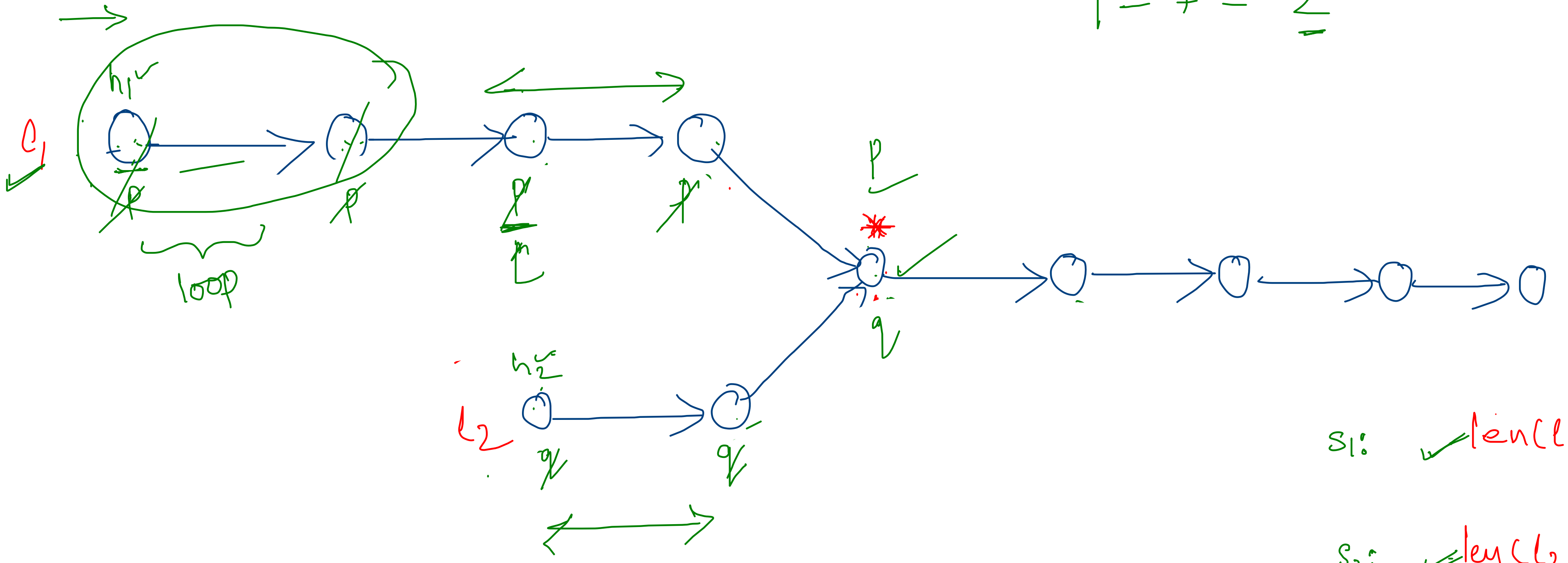
1. p = head, q = head.

2. p : @ Kth node from beg.

2-pointer Approach

```
1) take two pointers p and q of type Node

2) put one pointer at beg of kth node  ⇒ loop

3) p=head, q=head

4) for(int count=1;count<k && p!=null;count++)
    {
        p=p.next
    }
                                        ↳ k > len.
                                          ↓    ↳ 7
                                          8

5) if(p==null)
        return

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

7) return q;   q.data
```

$9 - 7 = \underline{2}$



s_1: $len(l_1) = 9$ ✓

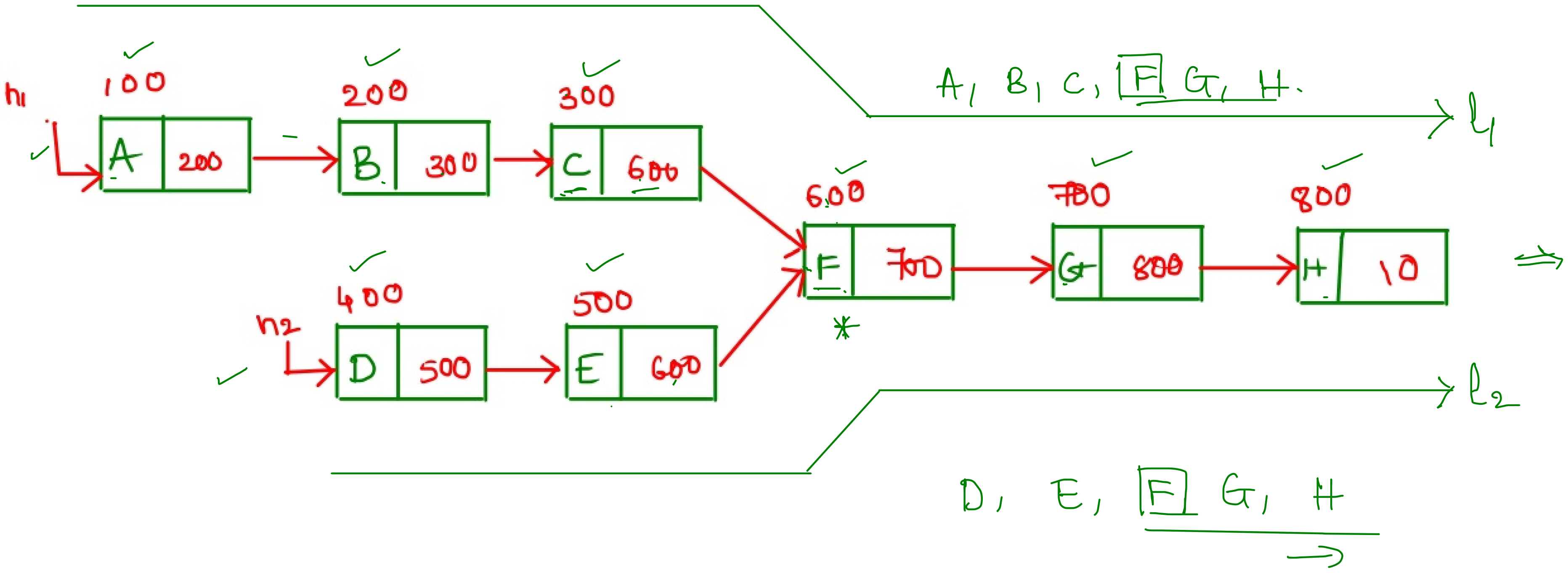
s_2: $len(l_2) = 7$

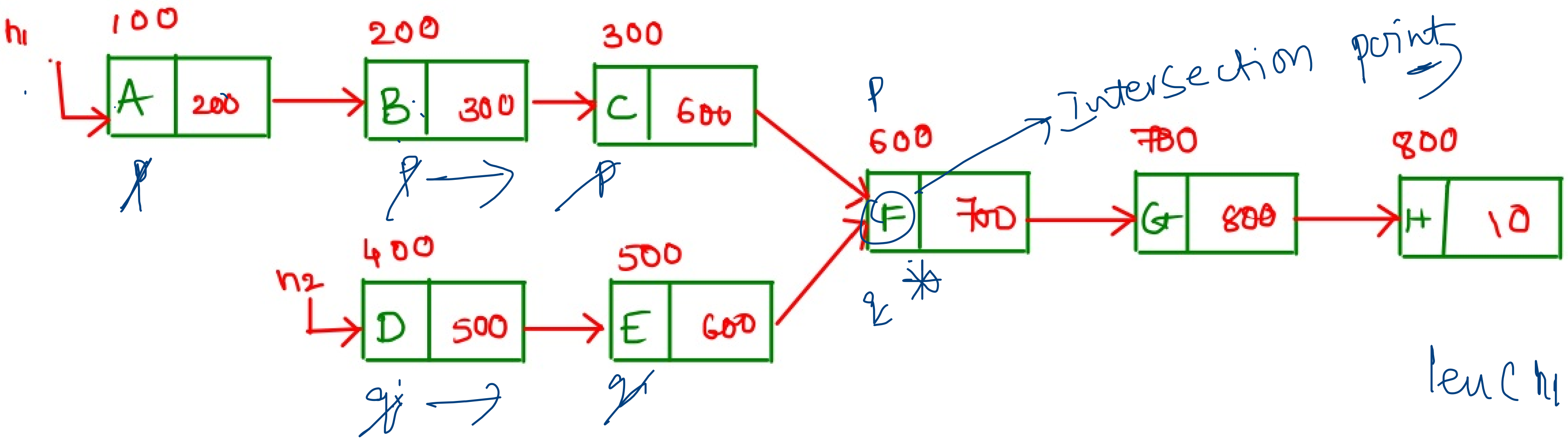
s_3: $|\underline{diff}| = \boxed{2}$ ✓

s_4: point p, $\underline{diff}$ nodes away

s_5 :

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

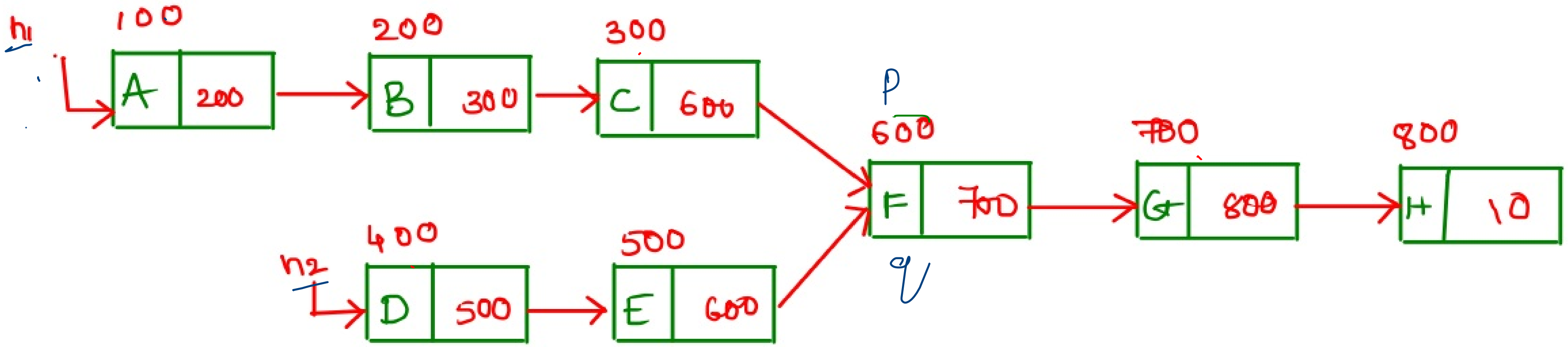


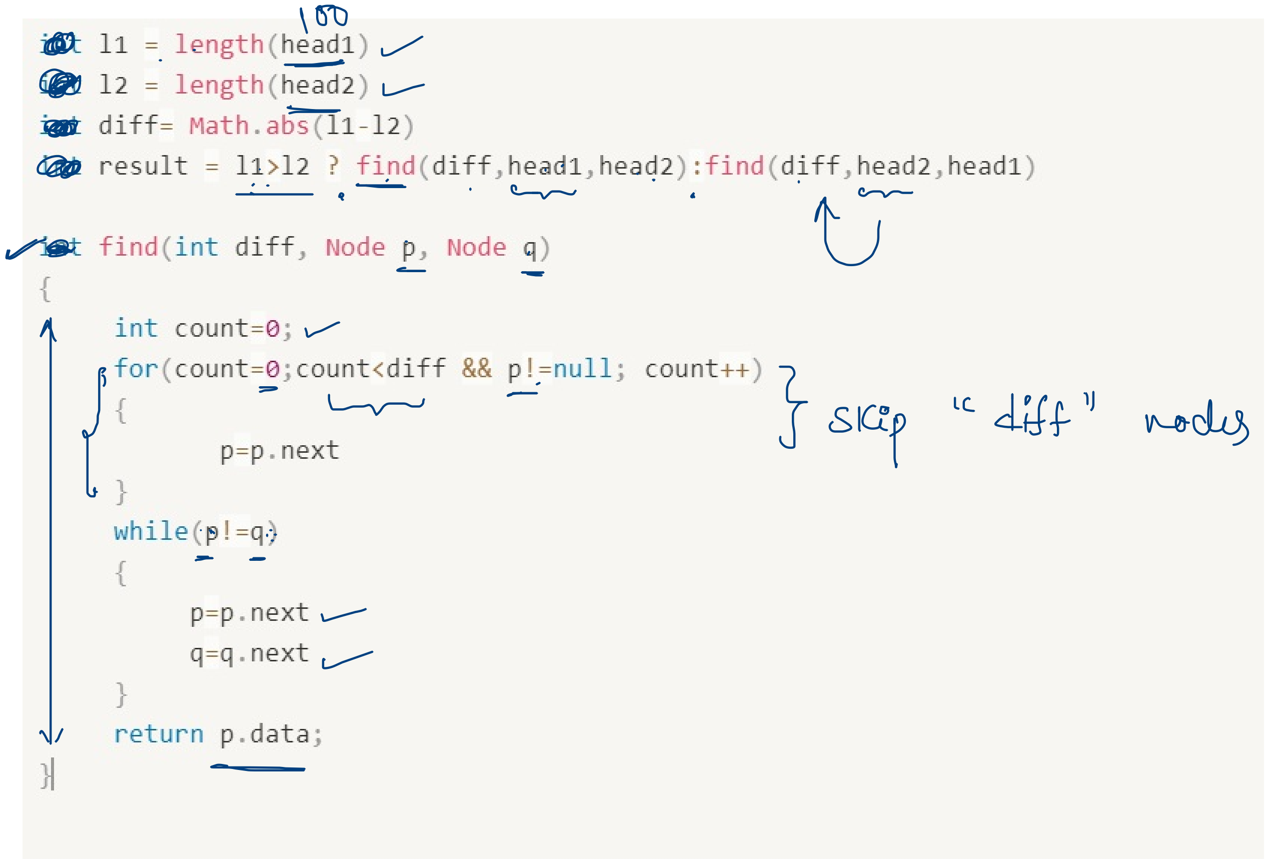


len(h1) = 6 ✓

len(h2) = 5

= $\underline{1}$ ✓





```
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;
}
```

100

} skip "diff" nodes

⇒ length ( )
{
}

10:50

* * * *

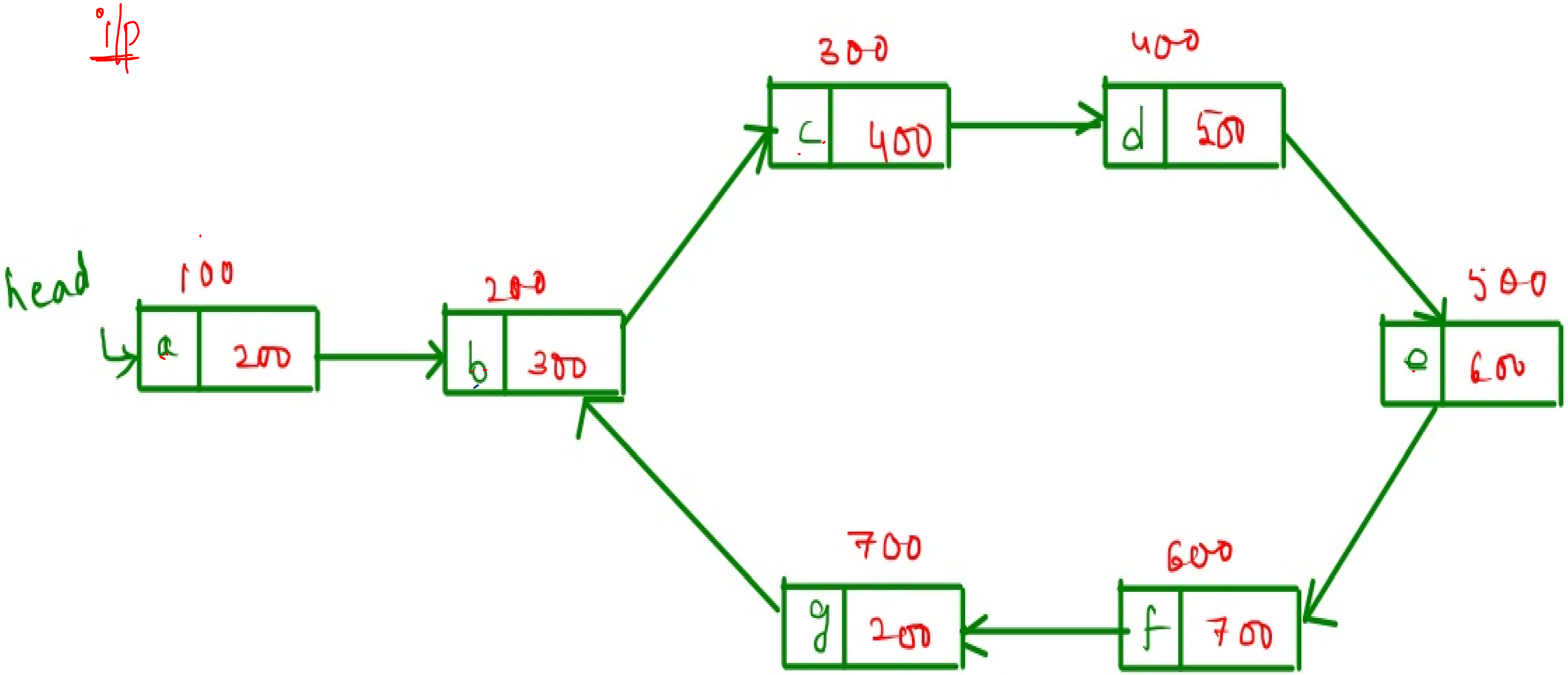
Find the loop in a SLL

cycle.

[a, b, c, . . . g] data.

[100, 200, . . . 700] add's.

i/p

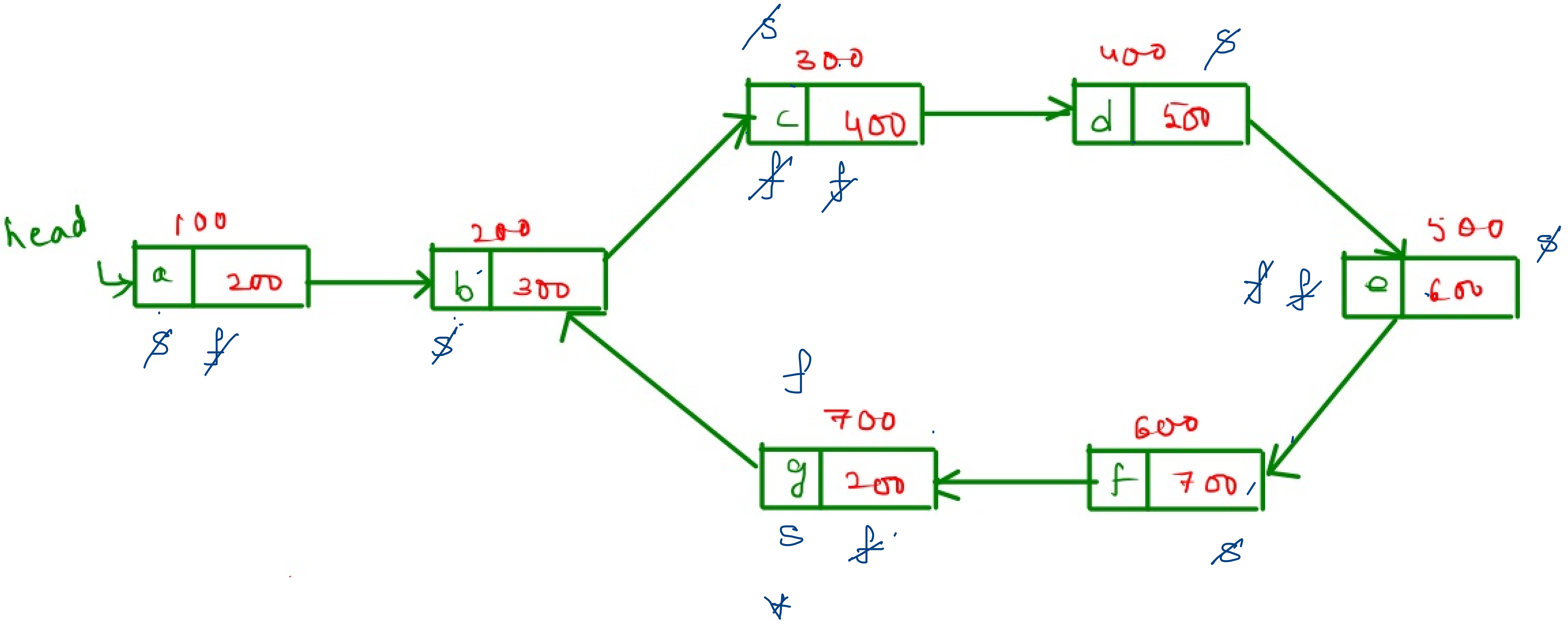


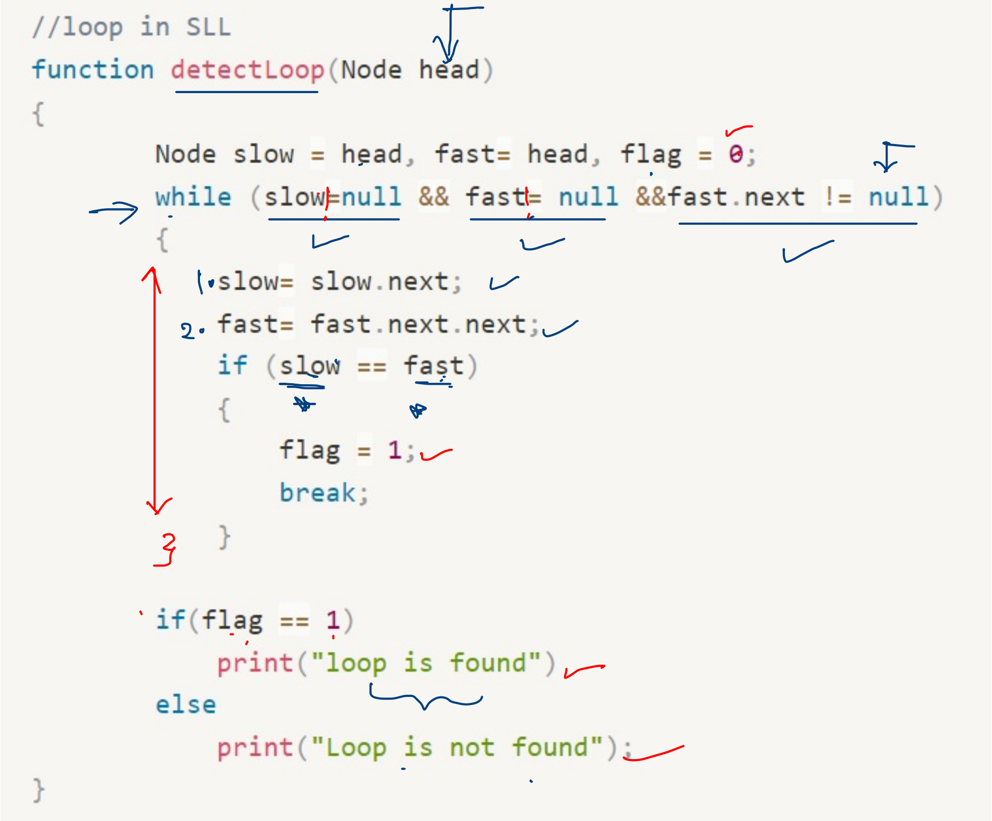
→ Floyd cycle detection Algo:—

→ T : cycle is present.

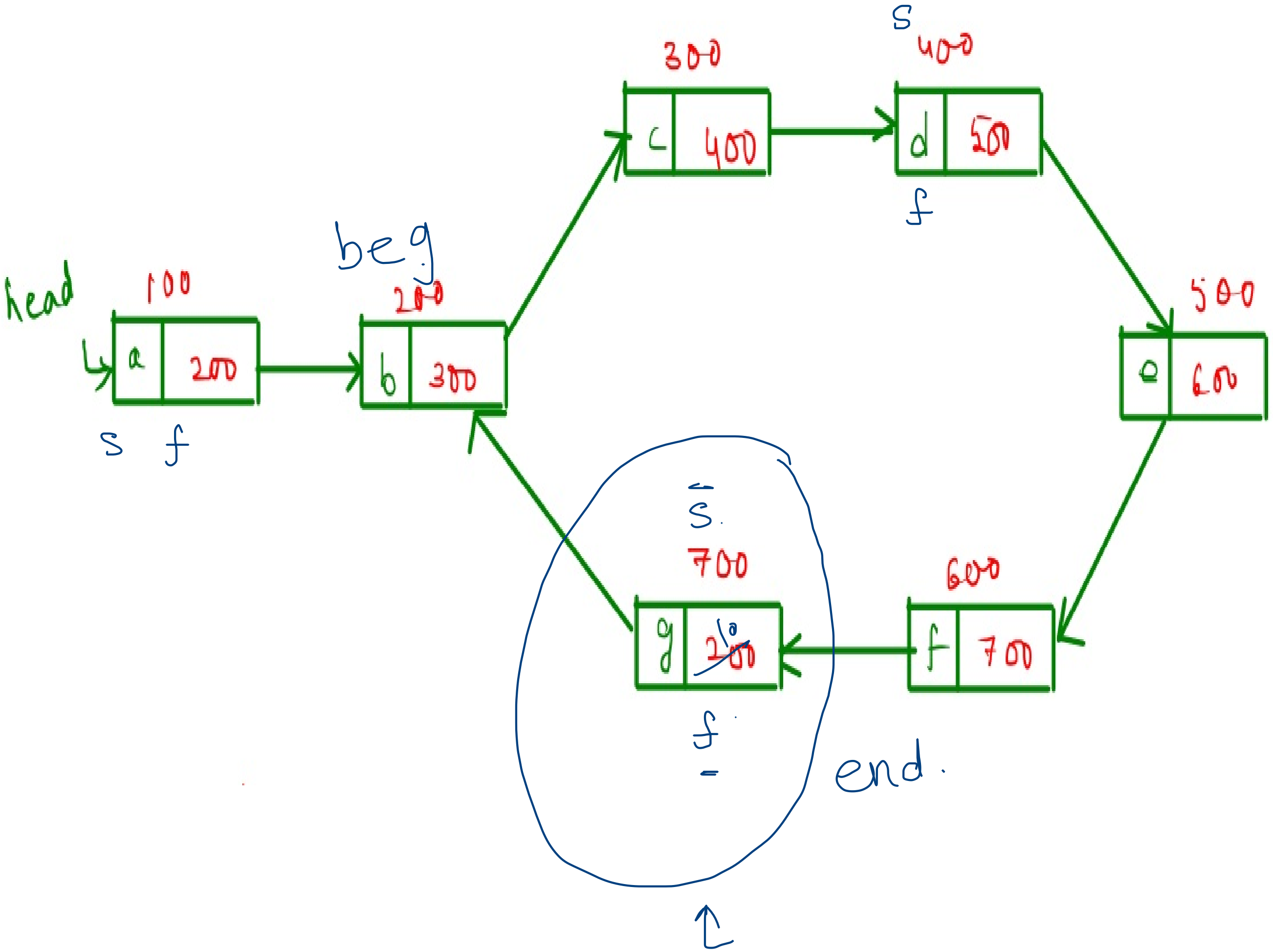
→ F : cycle is not present. ⇒ one node add's should be null.

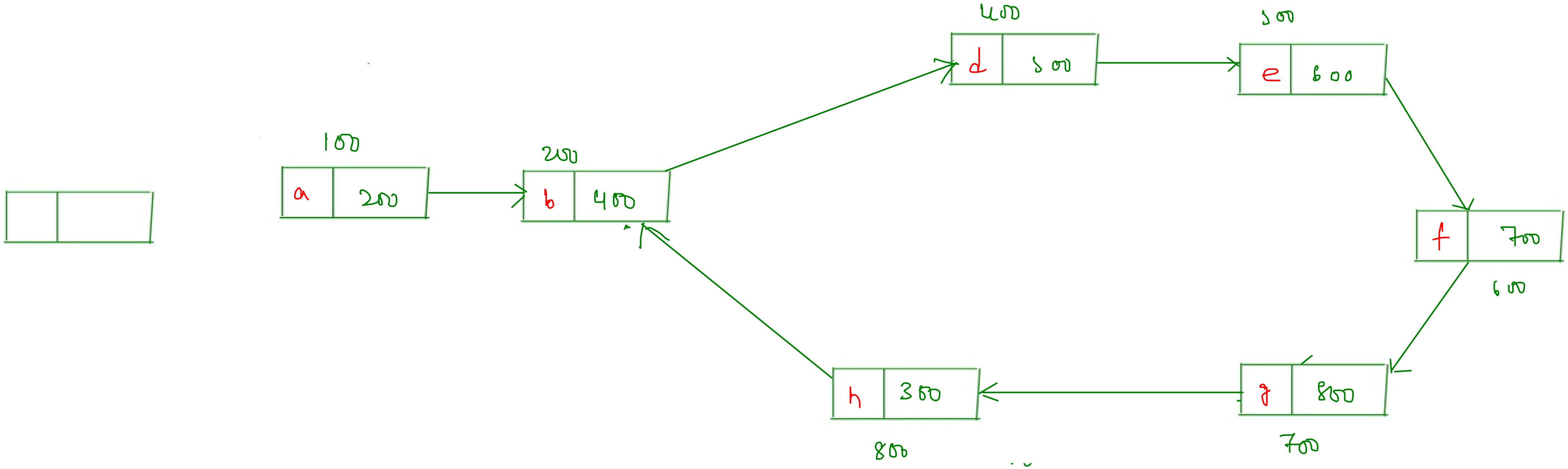
↳ s, f



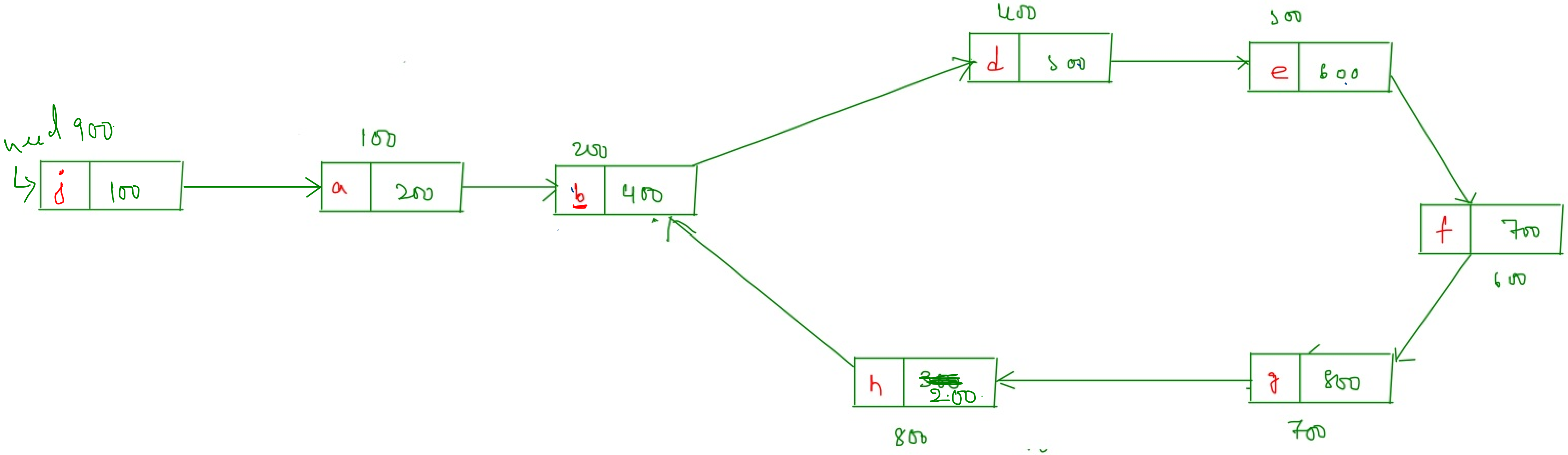


```
//loop in SLL
function detectLoop(Node head)
{
        Node slow = head, fast= head, flag = 0;
        while (slow!=null && fast!= null &&fast.next != null)
        {
          1.slow= slow.next;
          2.fast= fast.next.next;
            if (slow == fast)
            {
                flag = 1;
                break;
        3   }
        }

        if(flag == 1)
            print("loop is found")
        else
            print("Loop is not found");
}
```

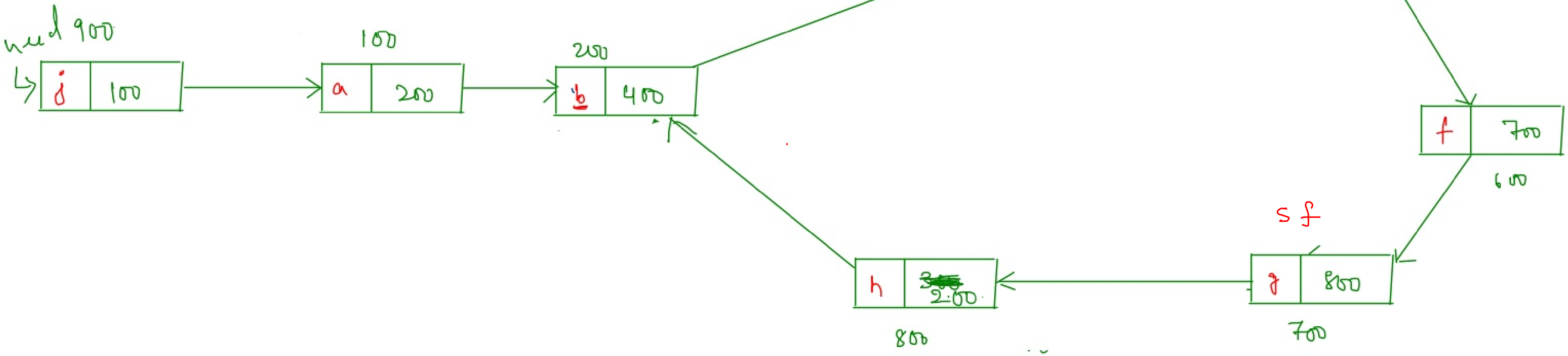


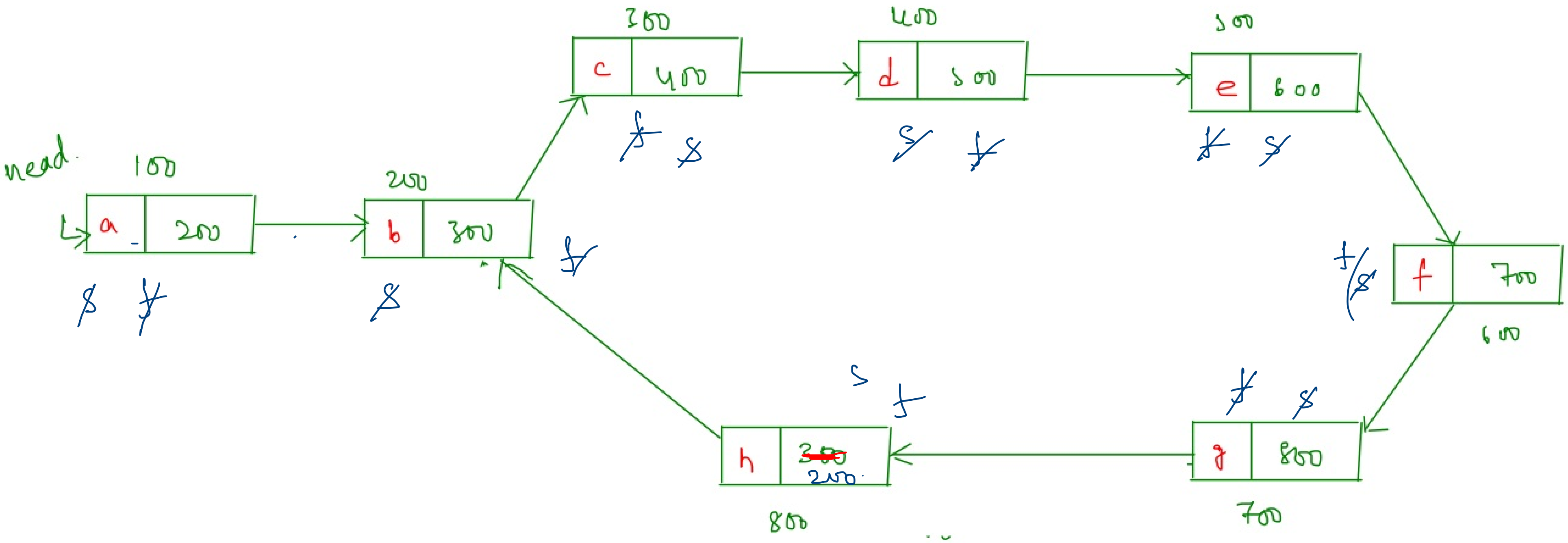


100
a
200
200
b
400
400
d
500
500
e
600
f
700
600
g
800
700
h
300
800

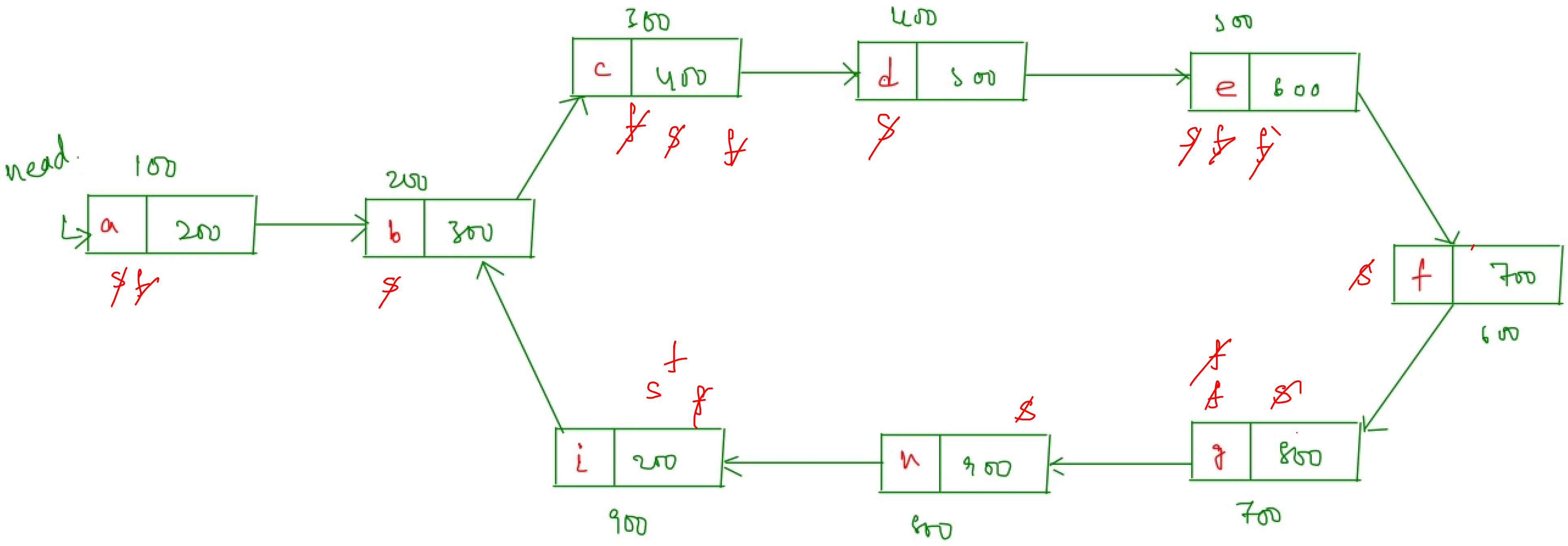


head 900
j
100
100
a
200
200
b
400
400
d
500
500
e
600
f
700
600
g
800
700
h
~~300~~
200
800

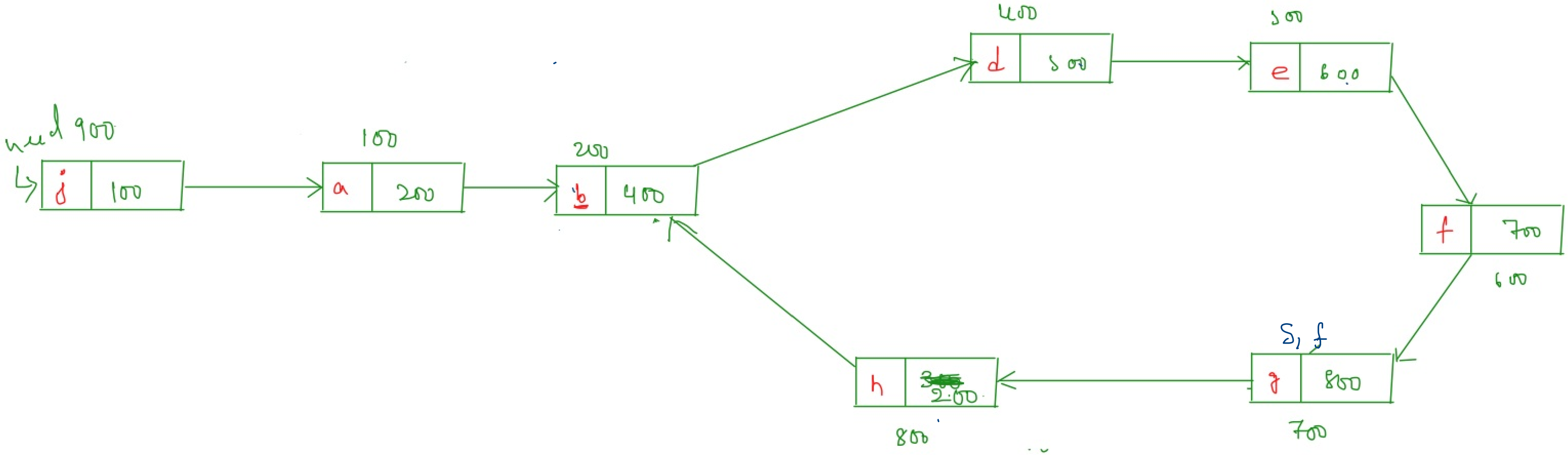

head 900
j
100
100
a
200
200
b
400
400
d
500
500
e
600
f
700
600
sf
g
800
700
h
~~300~~
200
800



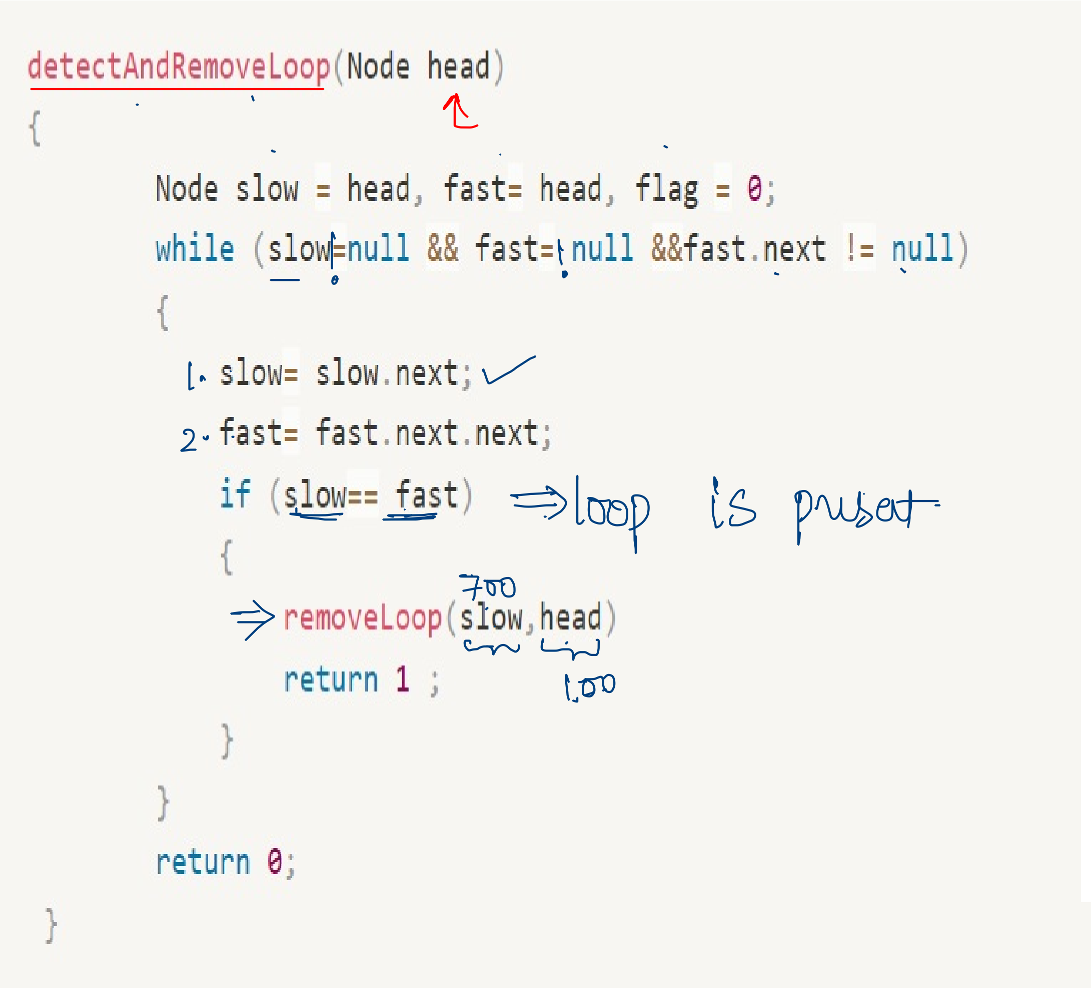
head
100
a
200
200
b
300
300
c
400
400
d
500
500
e
600
600
f
700
700
g
800
800
h
~~300~~
200
s
f



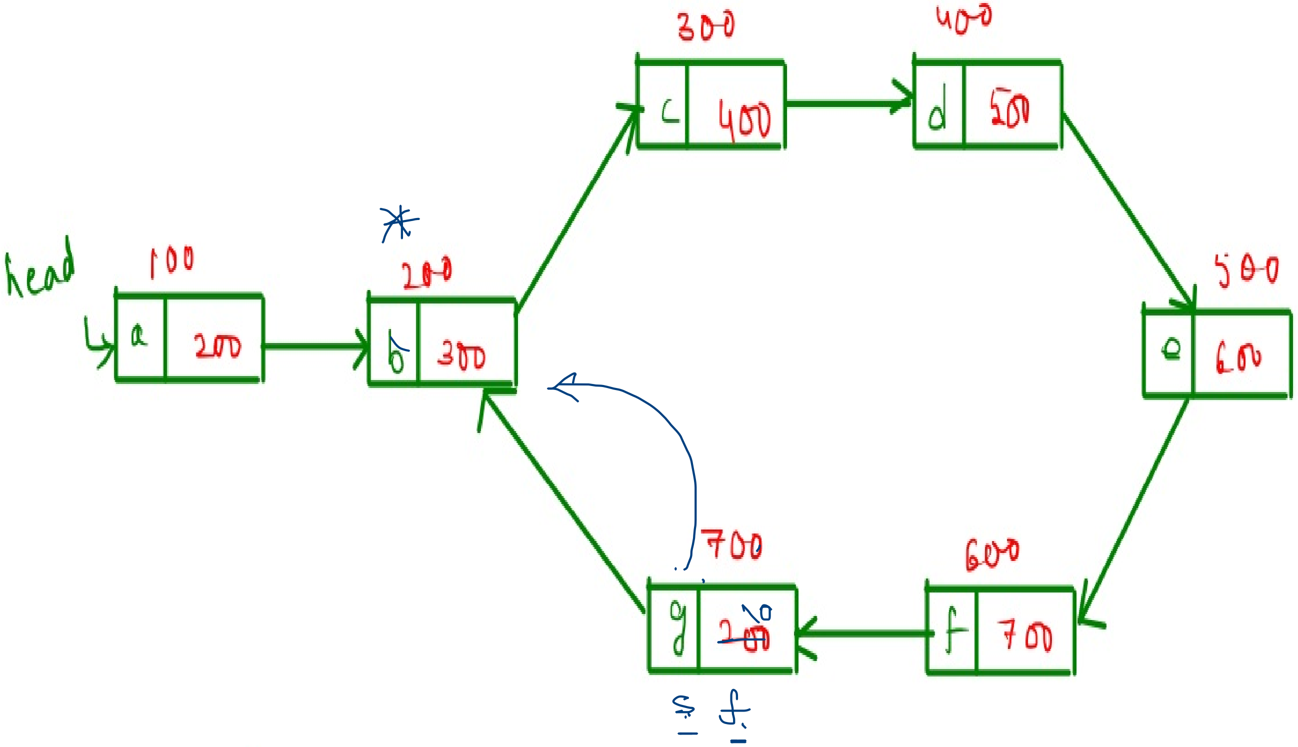
head
100
a
200
200
b
300
300
c
400
400
d
500
500
e
600
f
700
600
g
800
700
h
900
800
i
200
900
s
f

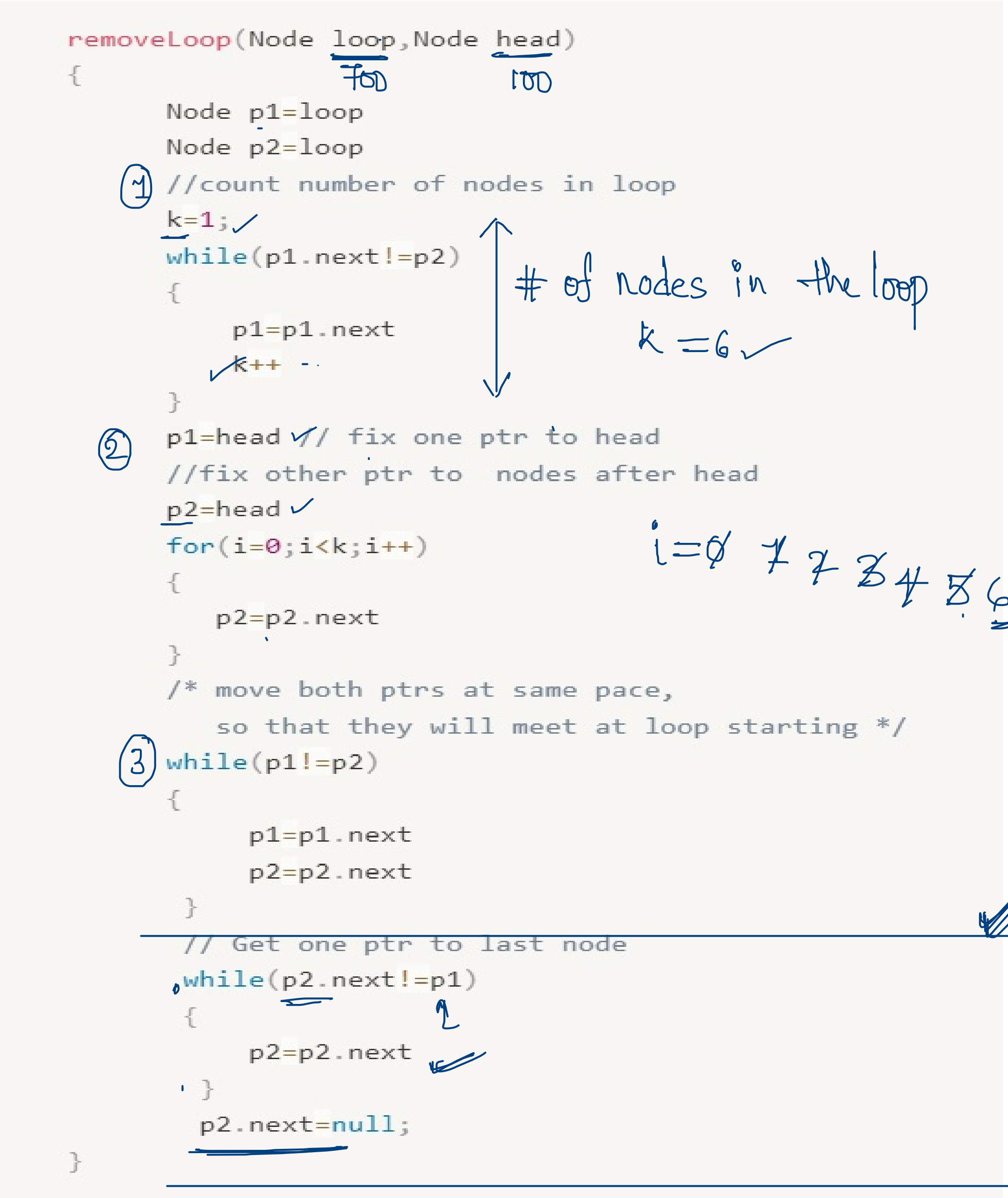


new 900
j
100
100
a
200
200
b
400
400
d
500
500
e
600
f
700
600
S, f
g
800
700
h
~~300~~
200
800



```
detectAndRemoveLoop(Node head)
{
        Node slow = head, fast= head, flag = 0;
        while (slow!=null && fast=!null &&fast.next != null)
        {
          1. slow= slow.next; ✓
          2. fast= fast.next.next;
            if (slow== fast)  ⇒ loop is prsent
            {
                              700
              ⇒ removeLoop(slow,head)
                return 1 ;          100
            }
        }
        return 0;
 }
```





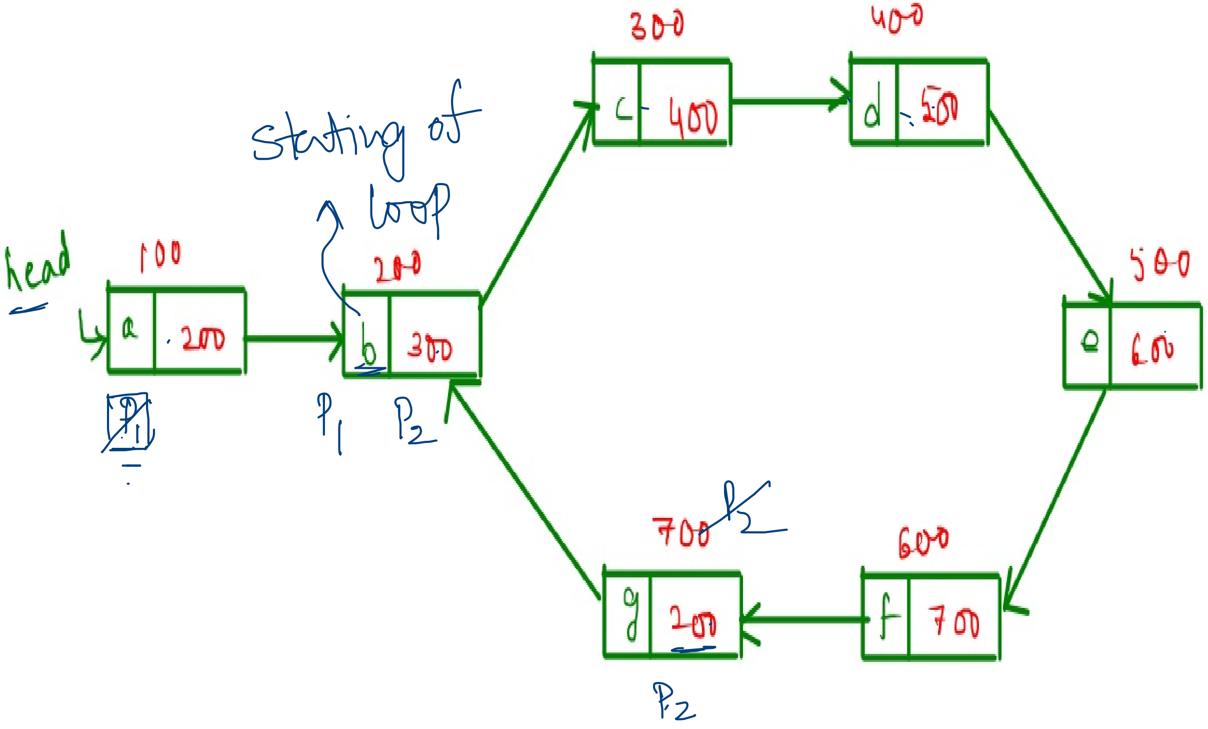
```
removeLoop(Node loop,Node head)
{                   700         100
        Node p1=loop
        Node p2=loop
     ① //count number of nodes in loop
        k=1;
        while(p1.next!=p2)
        {
            p1=p1.next
            k++
        }
    ②  p1=head // fix one ptr to head
        //fix other ptr to  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 */
     ③ 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;
}
```

# of nodes in the loop

k = 6

i = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6

starting pt loop



1. find loop

2.

3. breaking loop

Add two linked lists