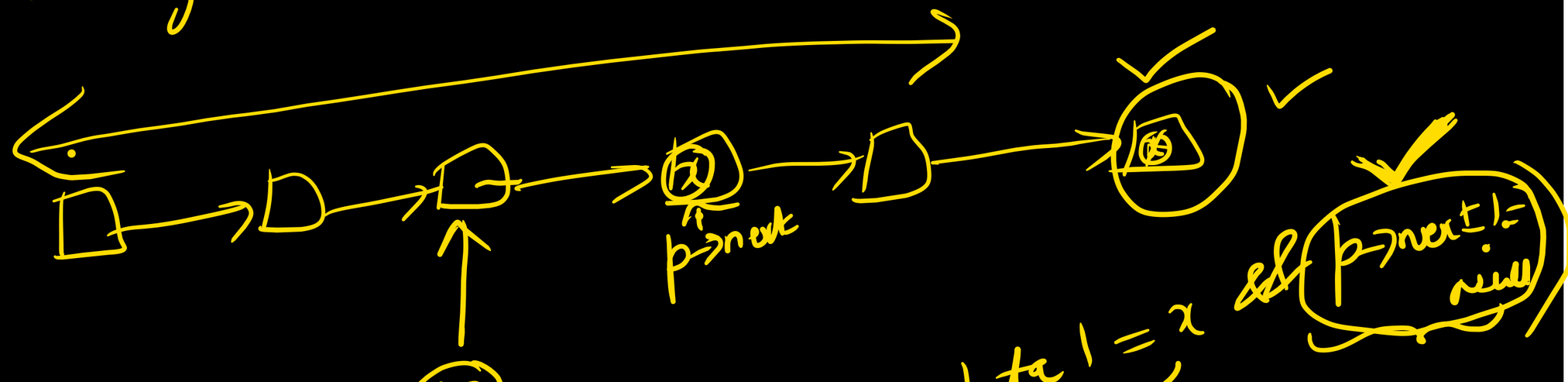
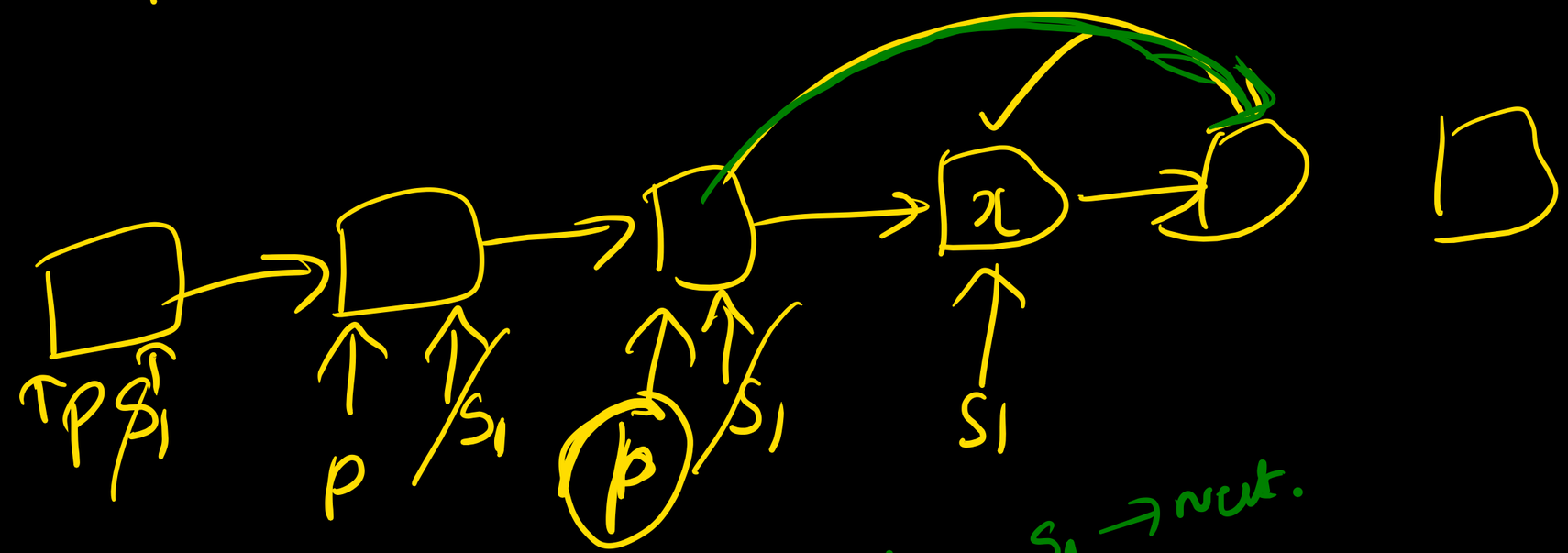
Write an algo to delete node with data 'x' in SLL.



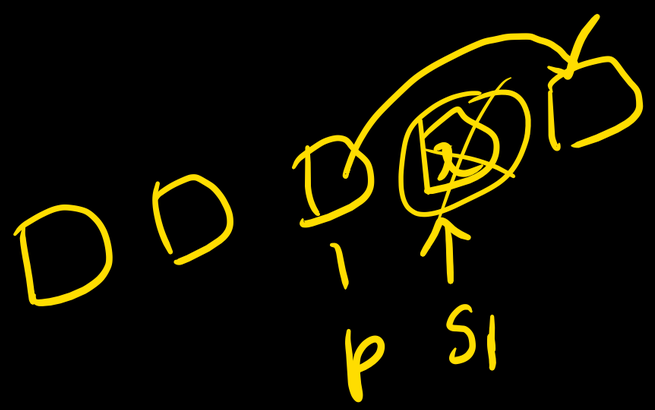
You all know this.

```
while (p->next->data != x)
    p = p->next;
```

Two pointers



Another cases: if (S==NULL) (Pf "no x" in SLL)



$S_1 = S$

while ($S_1 \rightarrow data \; != x$) && ($S_1 \rightarrow next \; != NULL$)

{ $p = S_1$; → p is following S_1

$S_1 = S_1 \rightarrow next$

}

if ($S_1 \rightarrow data == x$)

{ $p \rightarrow nodx = S_1 \rightarrow next$; fun(S)

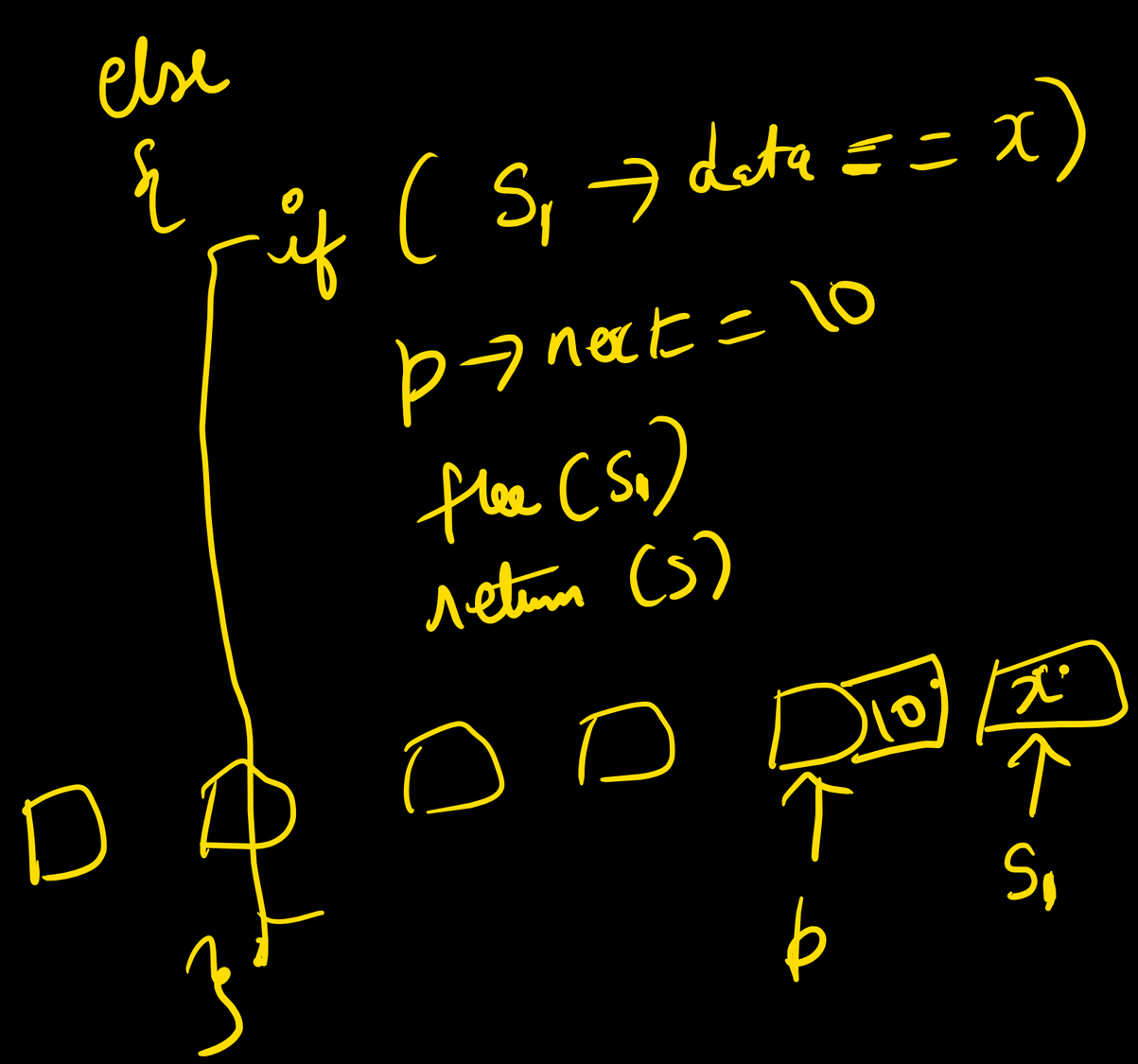
return (S);

else

end is reached

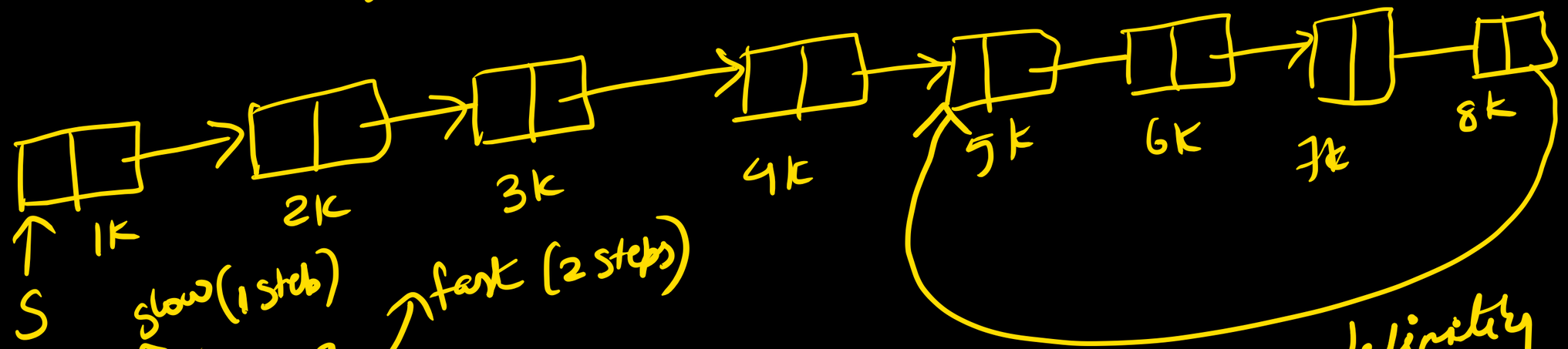
F

```
else
{
    if ( S1 -> data == x)
    {
        p -> next = 10
        free (S1)
        return (S)
    }
```



P
S1

Write an algo to detect cycle / loop in a given SLL.



p	q
1K	1K
2K	3K
3K	5K
~~4K~~	~~7K~~
5K	5K

→ meet

If there is cycle, then definitely p, q will meet.

Gate:

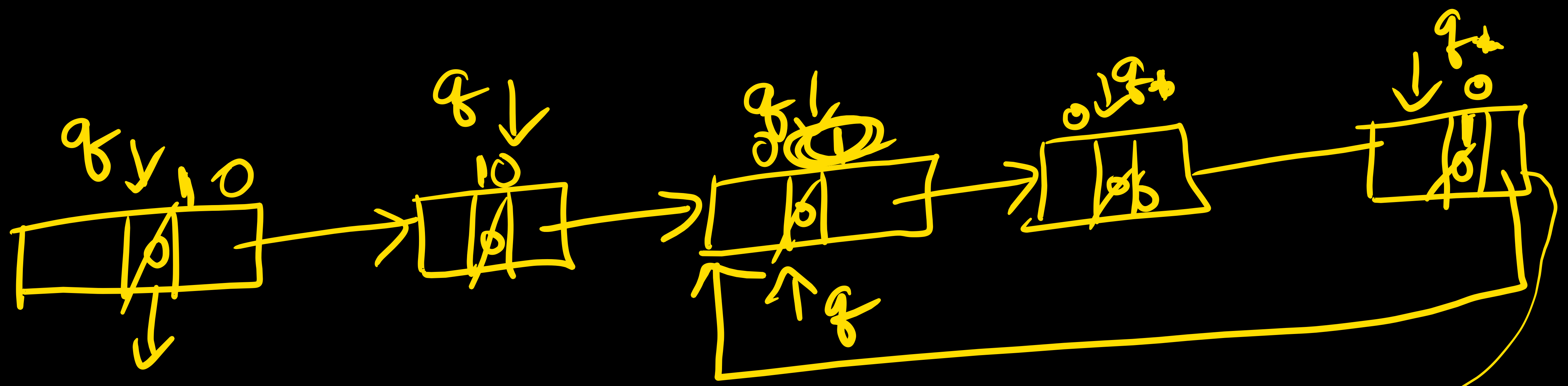
```
p = q = s
while ( q != NULL && q->next != NULL && q->next->next != NULL)
{
    p = p->next;
    q = q->next->next;
    if (p == q)
        pf (loop found);
        exit → algo → return
}
pf ( no loop found)
```

In gate example what is line ① ✓

add all test cases:

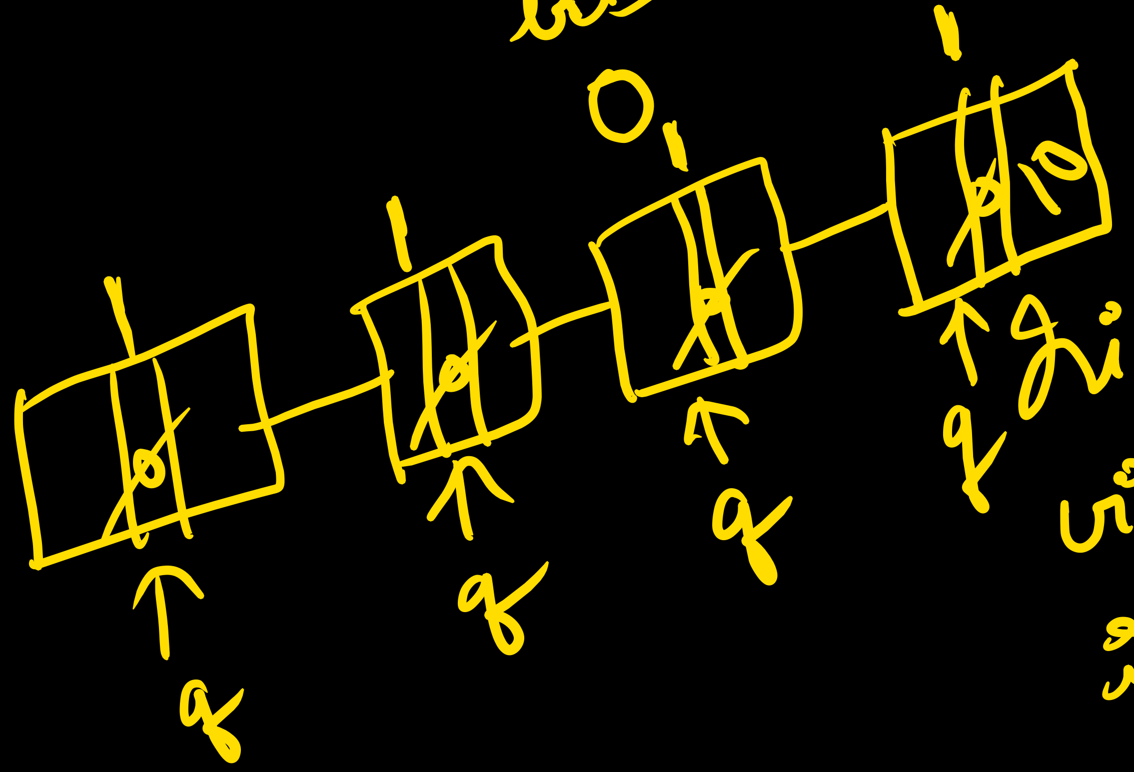
if s == null

if s->next == null.



extra bit

$SC = O(n)$; extra space is required.

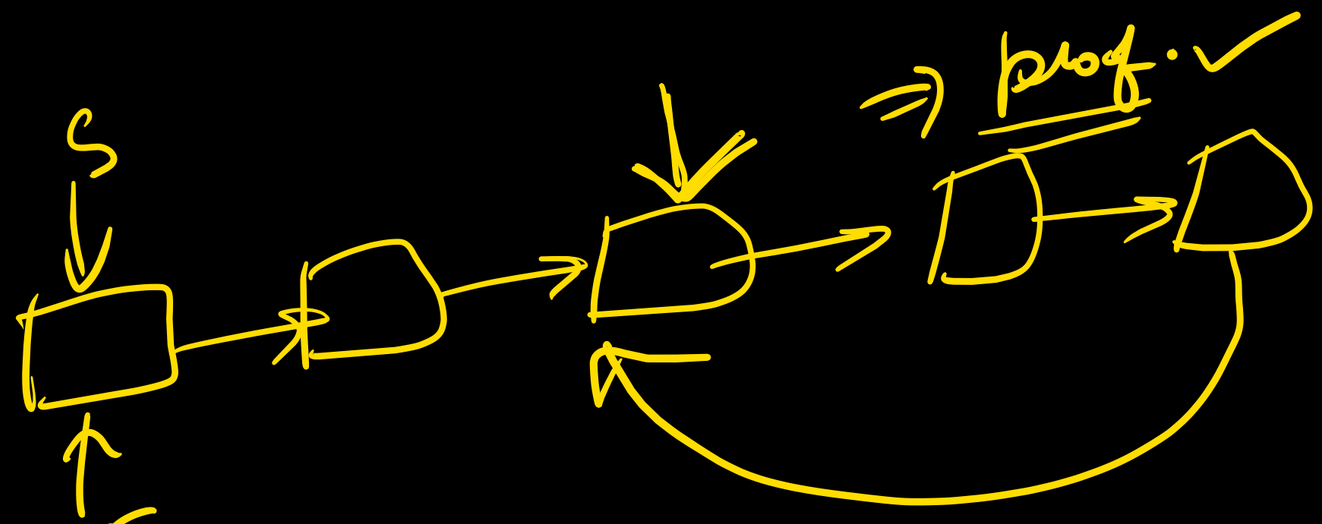


loop.

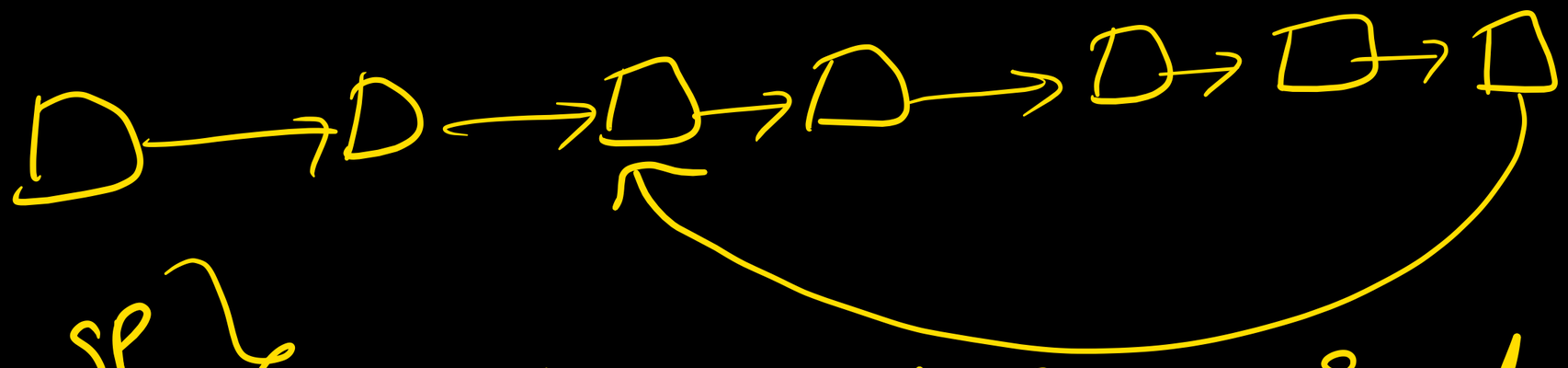
Initially all 0's.

visited all the nodes and make 0's as 1's

if you come across 1, loop is there.



Slow pointer and fast pointer ⟹ will they always meet?

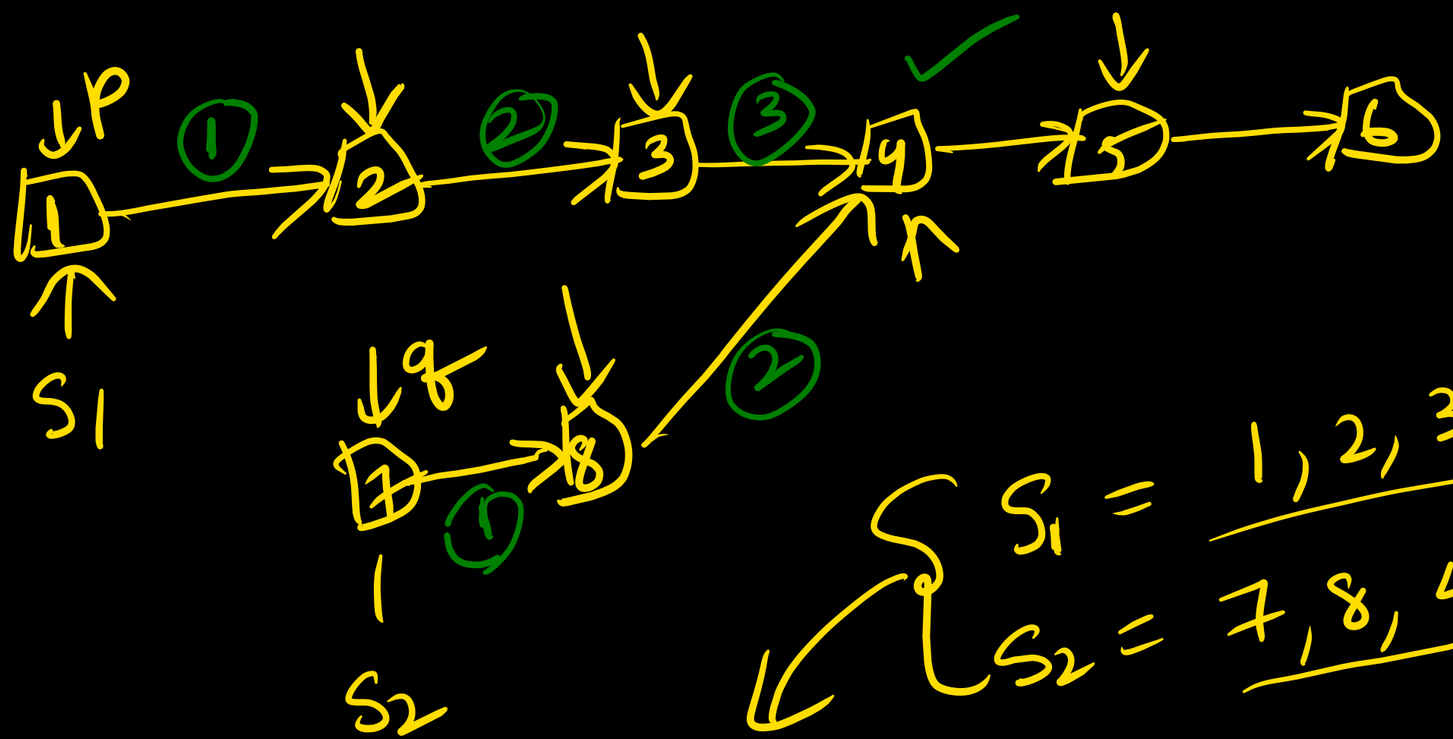


1 step — SP
2 steps — FP
and
→ definitely meet if there is a loop.

→ proof on internet.

3 steps & 4 steps

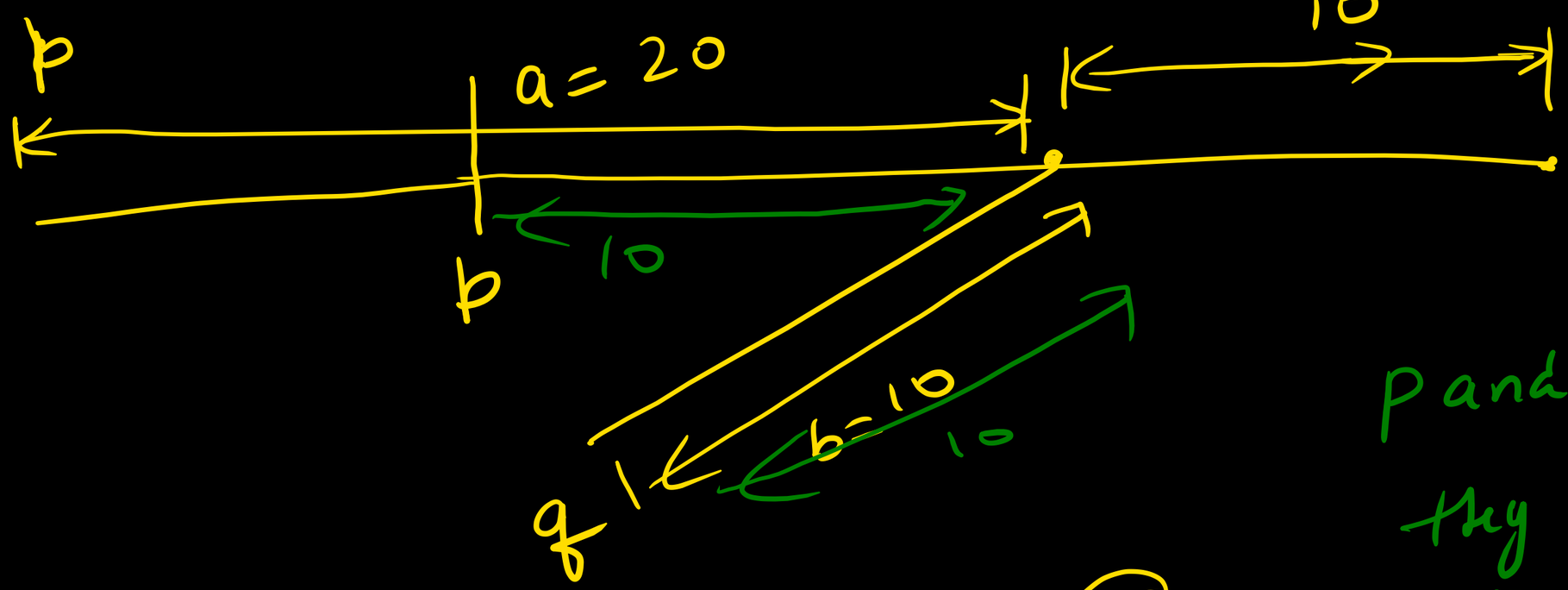
Find the address of I intersecting node.



$S_1 = 1, 2, 3, 4, 5, 6$

$S_2 = 7, 8, 4, 5, 6$

intersecting I node of intersection = 4 ✓

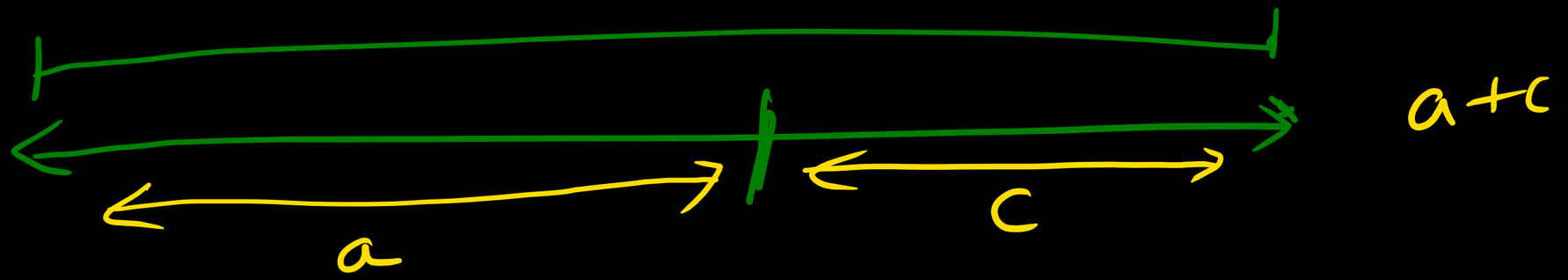


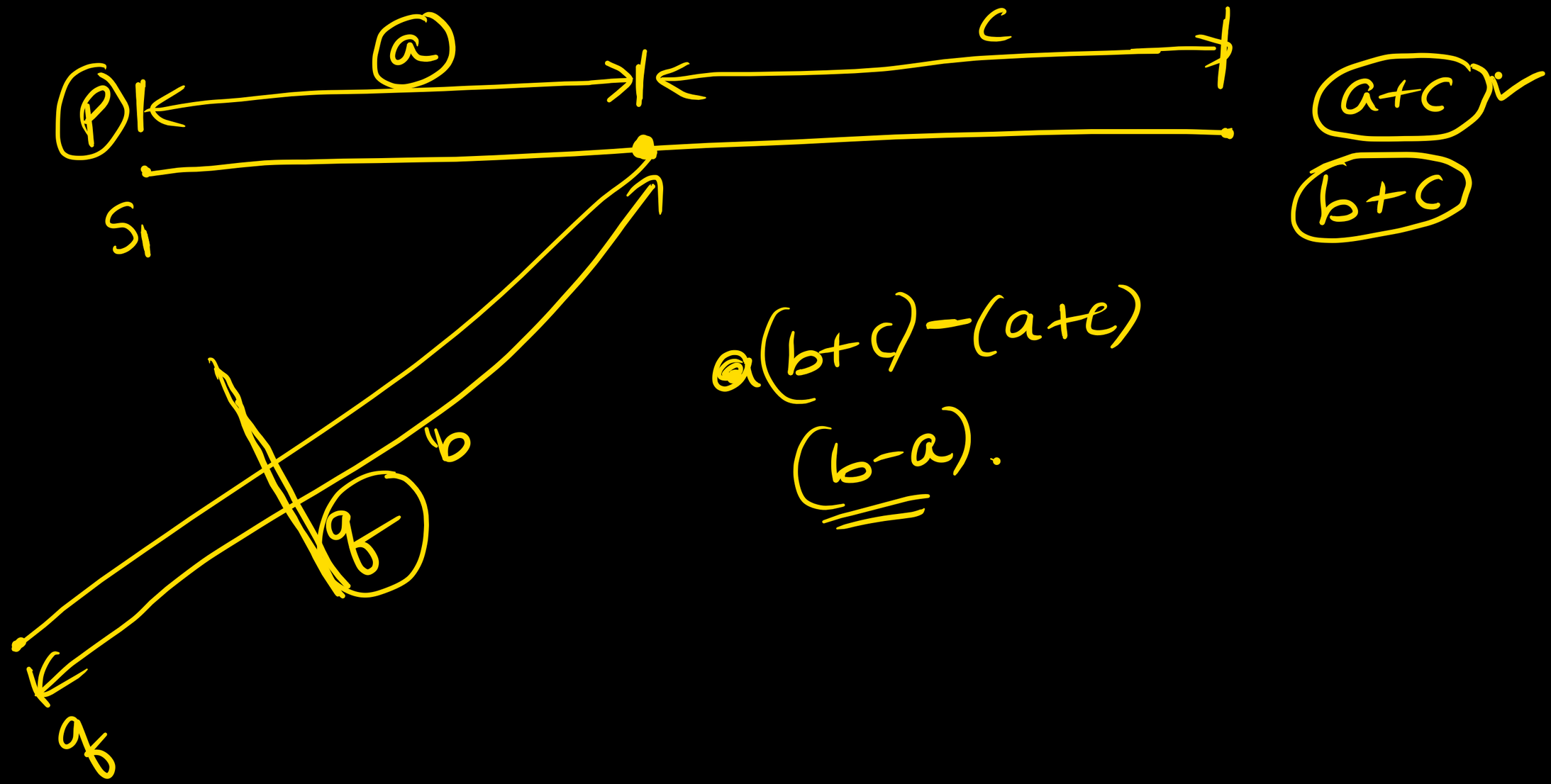
p and q simultaneously.
they will meet at
intersection point

Cannot find a and b

$a - b =$ 10

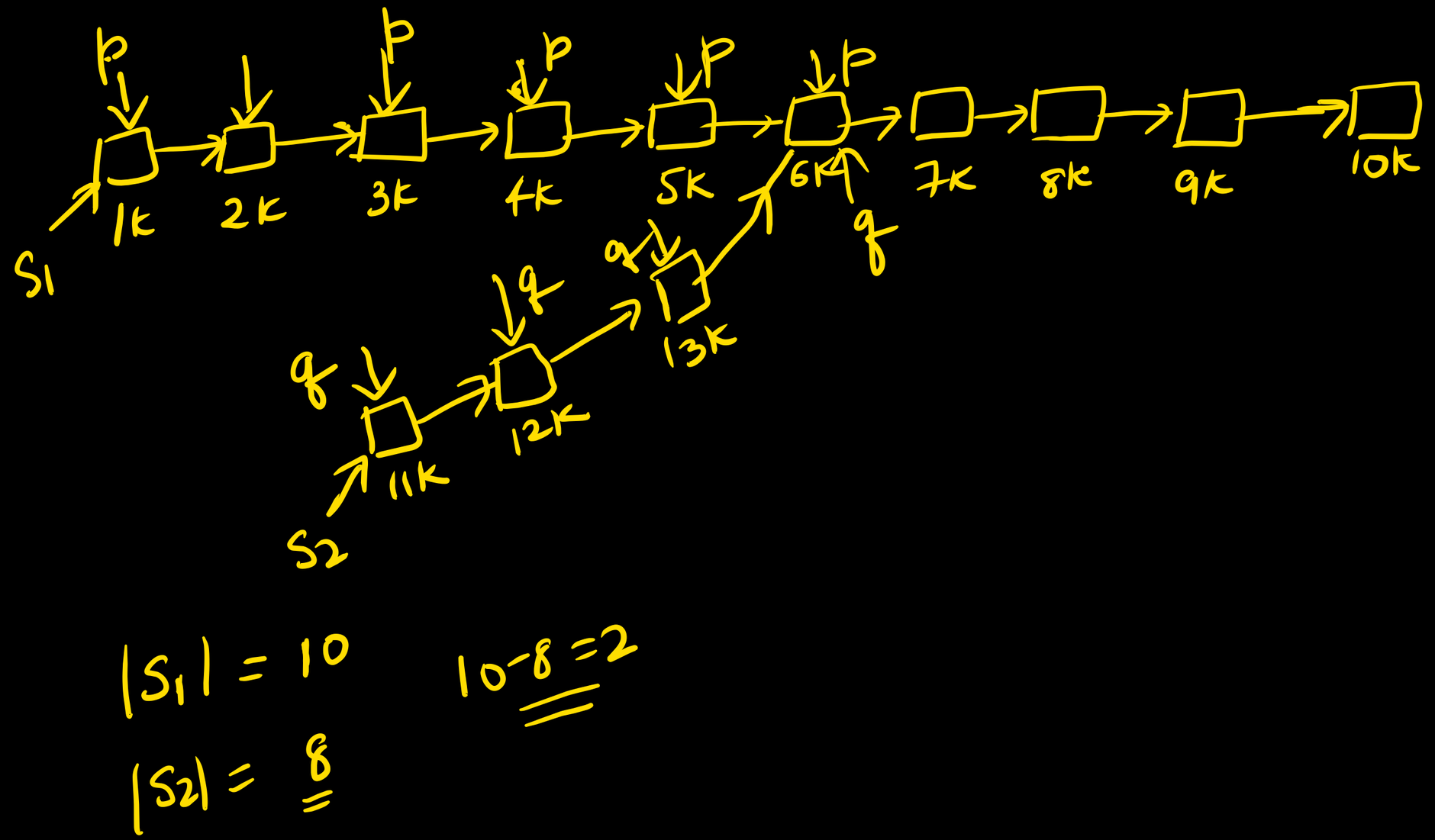
move p by a − b


a+c
a
c



$$a(b+c)-(a+c)$$

$$(b-a).$$



$|S_1| = 10$ $\quad 10-8=2$

$|S_2| = 8$

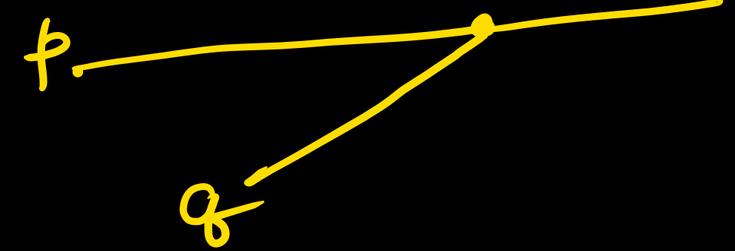
Check which LL is longer

m = size of LL1 , n = size of LL2

D = |m - n|

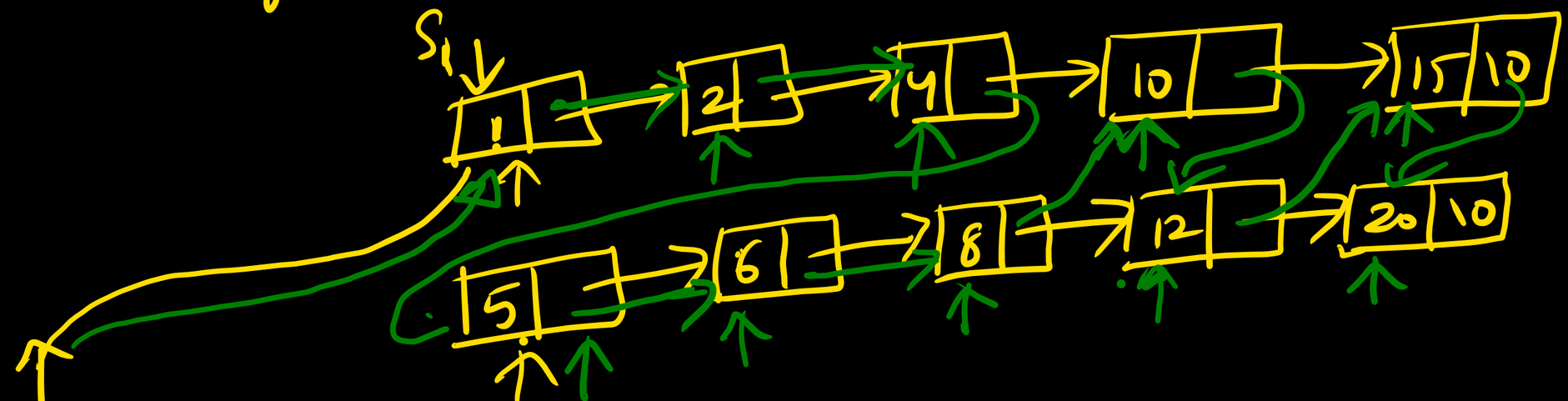
// move the pointer on the longer LL

```
while (D != 0)
{   p = p->next
    D = D - 1
}
while (p != q)
{   p = p->next ; q = q->next ; }
```

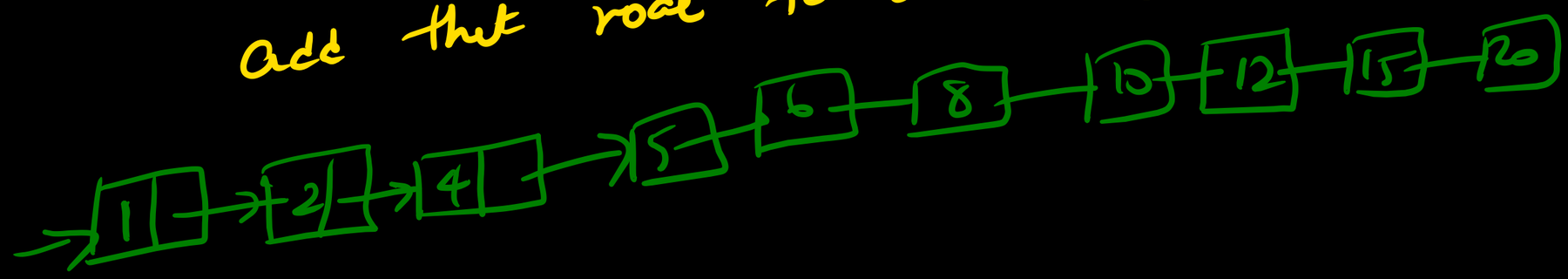


(Corner cases.)

merge two sorted LL into one linkedlist.



Compare $S_1 \rightarrow$ data with S_2 data, whichever is smaller, add that node to the end of resultant LL(R)



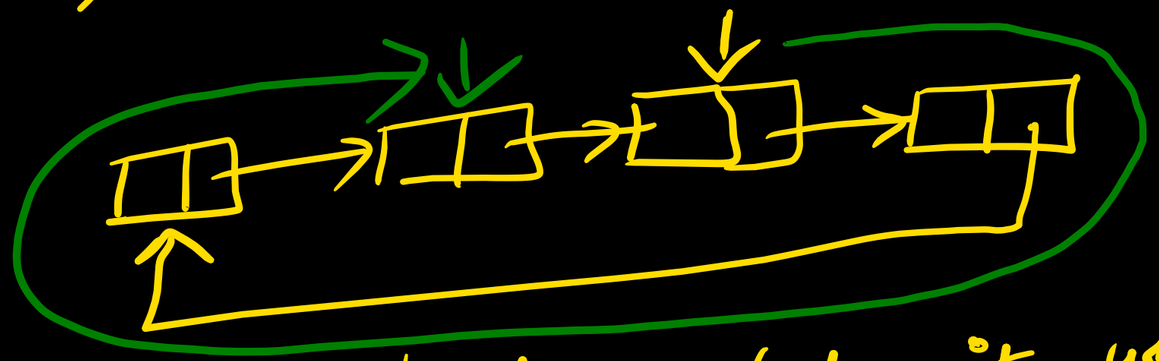
~~About~~ Drawbacks of SLL:

1) we cant go back
2) last node next pointer is always null.

So to eliminate these 2 drawbacks, we are going to Circular Single linked list.

C - SLL :-

→ In C-SLL, last node next part is 1st node address.

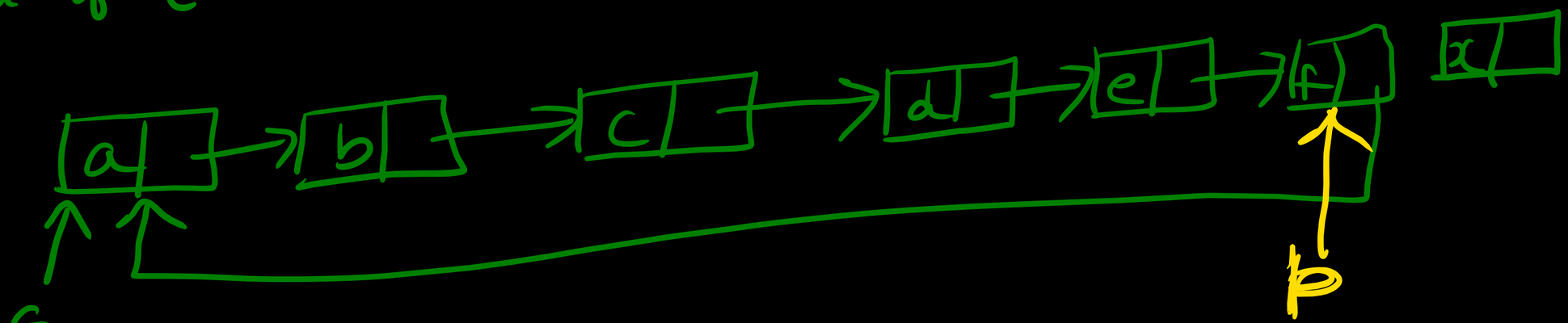


→ we can go back direction, but it will take O(n) time

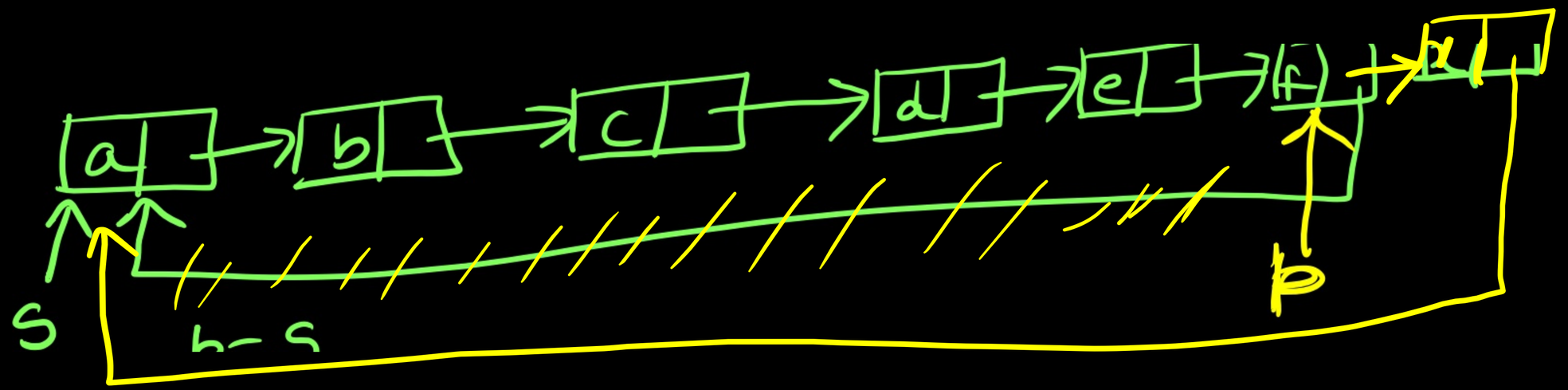
O(n) time

→ we are utilising last node next part efficiently

Write an algo to insert a node with data 'x' at the end of C-SLL.




```
p = S
while(p->next != S)
    p = p->next
q = (struct node *) malloc (size of(struct node));
q->data = x
```

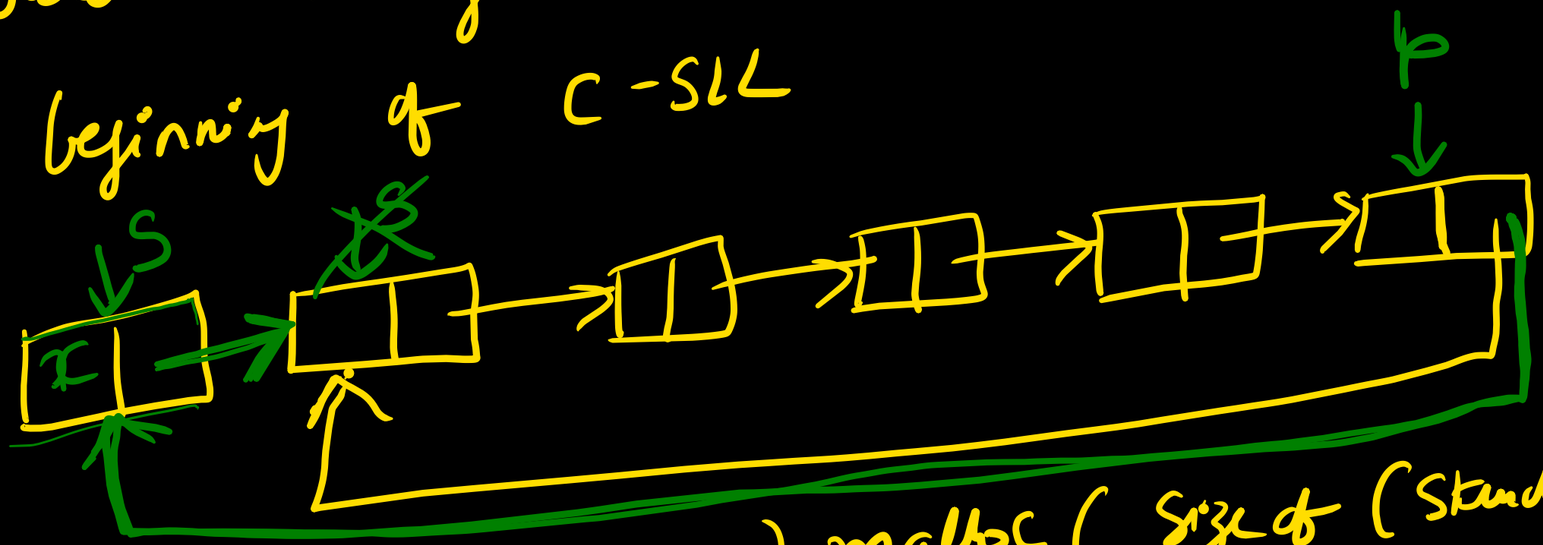


$p \rightarrow next = q$

$q \rightarrow next = s$

return (s).

Write an algo to add an element of data 'x' at the beginning of C-SLL



edge cases.

if (S == null)

if (S->next == null)

q = (struct node *) malloc (sizeof (struct node))

q -> data = x.

while p = S

while (p -> next ! S)

p = p -> next

p -> next = q.

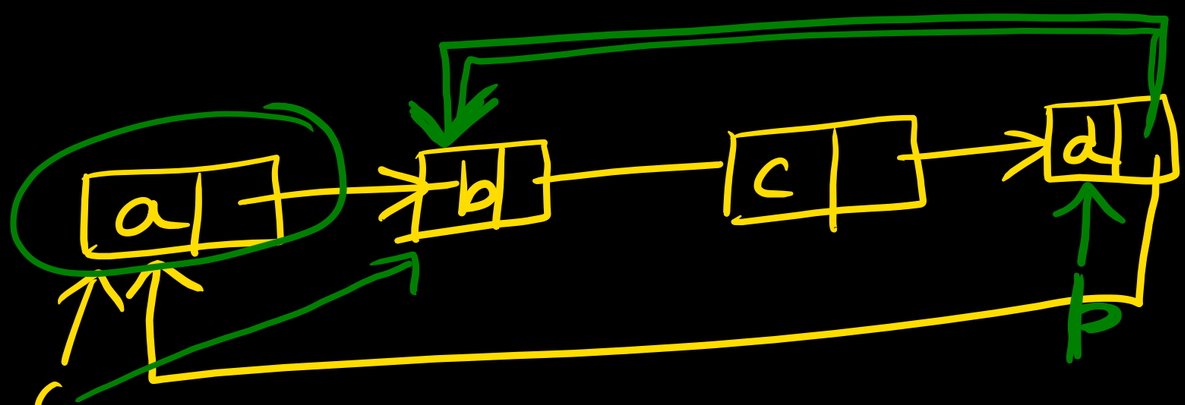
q -> next = S

S = q

return (S).

Gate

In a C SLL delete I node



p = s

while (p → next != s)

① p = p → next;

p → next = s → next

free (s); s = p → next;

→ blank

Correct code

if (s == null)

(s → next != s) == 1)

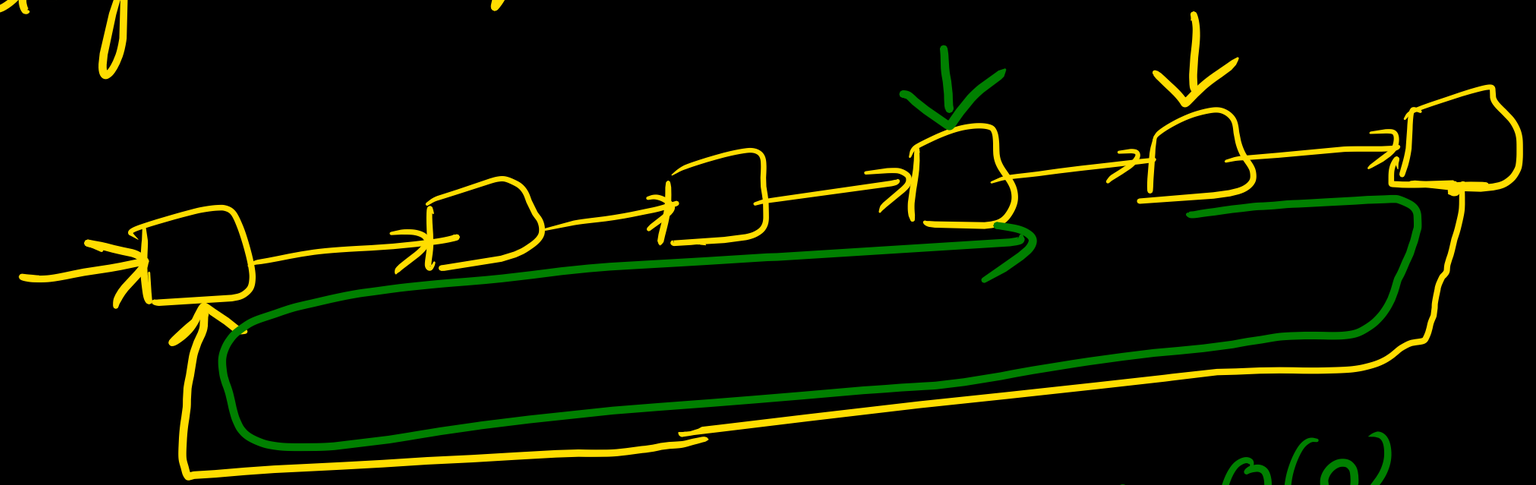
a) b) c) d).

Double linked ~~first~~ list :-

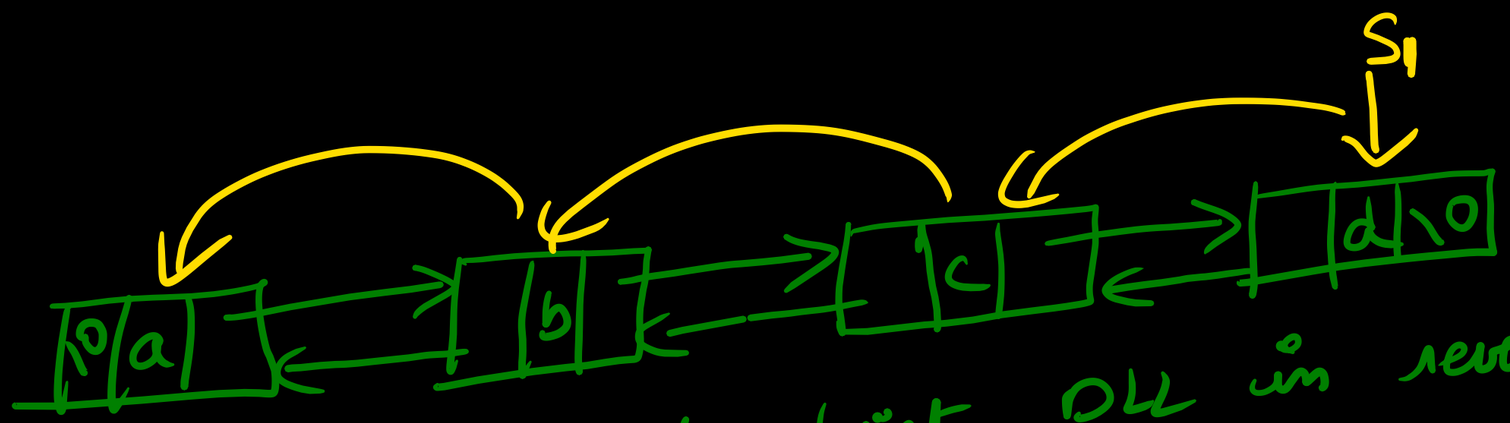
```
Struct node
{
    struct node * prev;
    int data;
    Struct node * next
};
```

Drawback of CSL :-

Going back required O(n) time



going back seq requires O(n) time

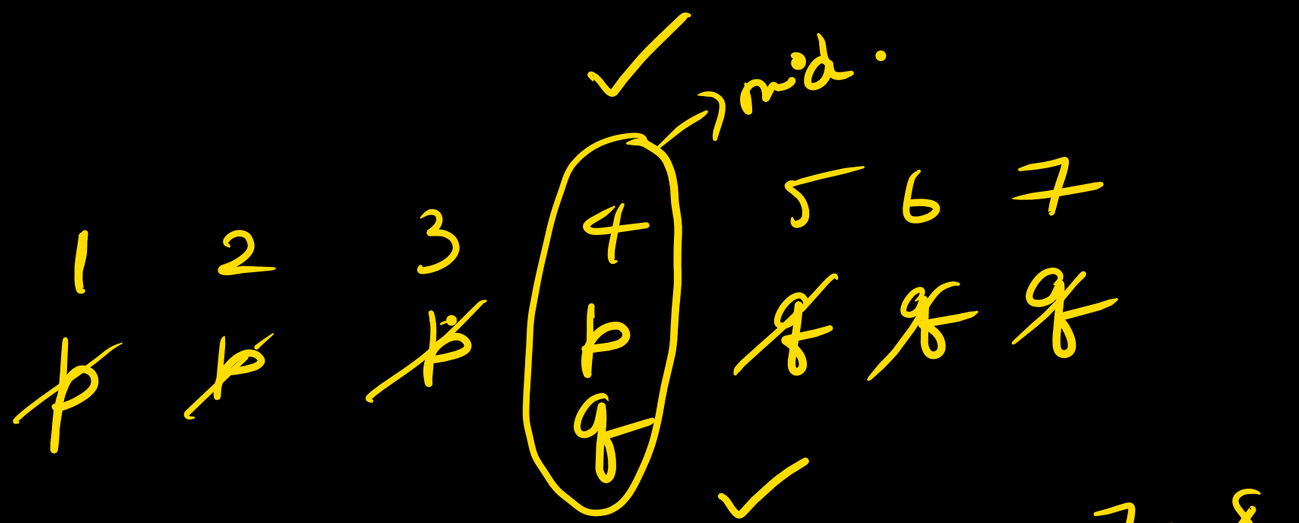


Write an algo to print DLL in reverse order

```
S1 = S
while (S1 -> next != null)
    S1 = S1 -> next
while (S1 != null)
{   pf (S1 -> data
    S = S1 -> prev
```

Write an algo to find mid element of DLL.

1) use two pointers from the beginning and increment one pointer by one and other by 2.
2) Count the no of nodes and find the middle.
3) Start one pointer from beginning and other pointer from end and increment by one step.


mid.
1 2 3 4 5 6 7
p p p p q q q
1 2 3 4 5 6 7 8
p p p p q q q q
p->next = q
two mid values.

① move q to end of DLL

② move p to start of DLL

③ while (p != q || p->next != q)

odd number

even

{

p = p->next;

q = q->prev;

}

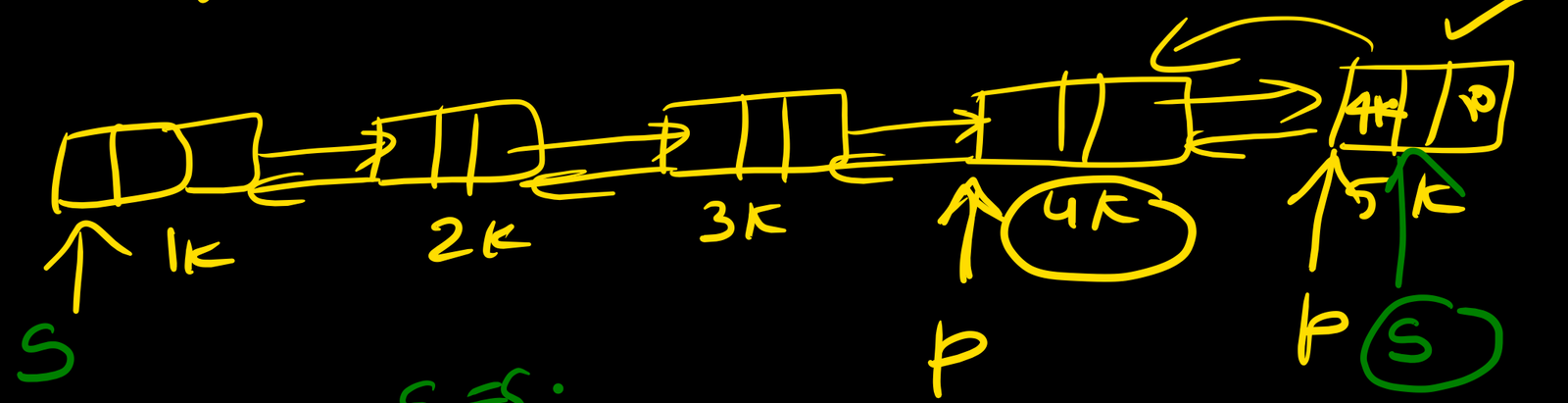
Corner cases

if

S == null

S->next == null

write an algo to find 2nd last node address in DLL



$S_1 = S$.

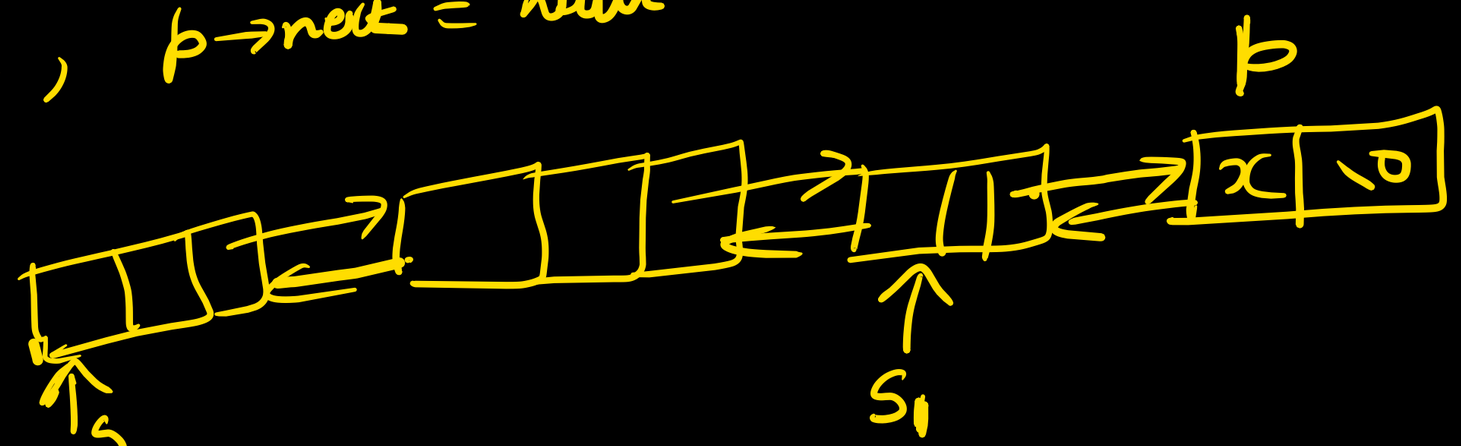
```
while ( S1 -> next -> next != null )
    S1 = S1 -> next
Pf ( S1 -> pw ).
```

```
while ( S1 -> next -> next != null )
    S1 = S1 -> next;
Print(S1);
```

Write an algo to addnode with data x at end of DLL

p=(struct node *) malloc (sizeof(struct node))

p→data = x , p→next = null



S1=S

while (S1→next != null)

{ S = S→next }

S1→next=p p→prev=S1