Today's Content

(a) Merge

(b) Rearrange Linked List

(c) Cycle detection

- (i) Detect cycle
- (ii) Find start of cycle
- (iii) Remove cycle

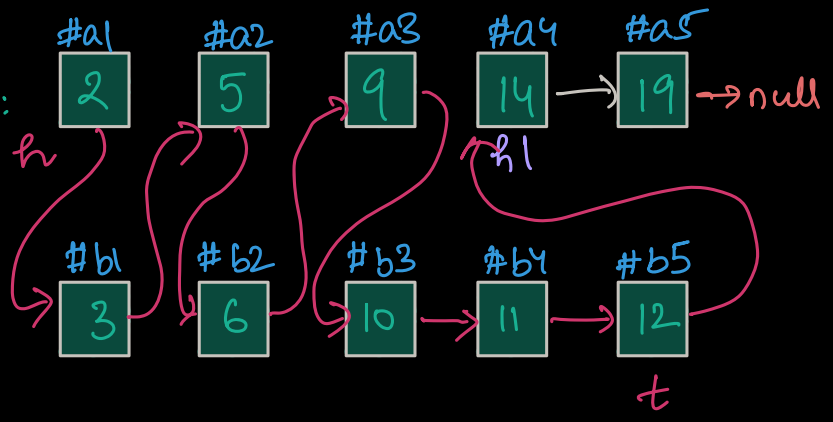
(d) Find intersection of linked list

 ← Copy me

Q1: Given 2 sorted linked list, merge & get final sorted list.

Expected TC: O(N+M), SC: O(1)

Eg 1:

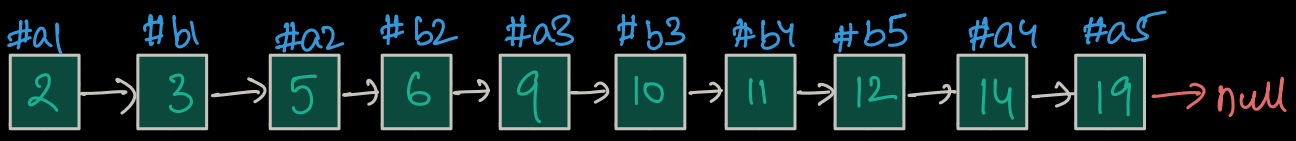


t.next = h1/h2

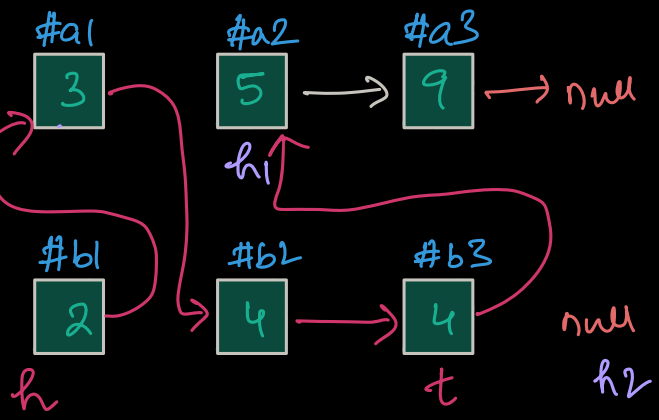
h1/h2.next

t = t.next

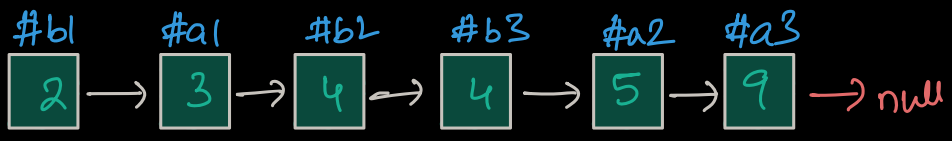
final:



Eg-2:



Final:



Pseudocode:

```
Node merge (Node h1, Node h2) {
    Node h, t
    if (h1 == null) return h2
    if (h2 == null) return h1
    if (h1.data < h2.data) { h = h1, t = h1, h1 = h1.next }
    else { h = h2, t = h2, h2 = h2.next }
    while (h1 != null && h2 != null) {
        if (h1.data < h2.data) {
            t.next = h1
            h1 = h1.next
            t = t.next
        }
        else {
            t.next = h2
            h2 = h2.next
            t = t.next
        }
    }
    // Link left over linked list
    if (h1 != null) { t.next = h1 }
    if (h2 != null) { t.next = h2 }
    return h
}
```

TC: O(n+m)
SC: O(1)

Merge Sort: To Do

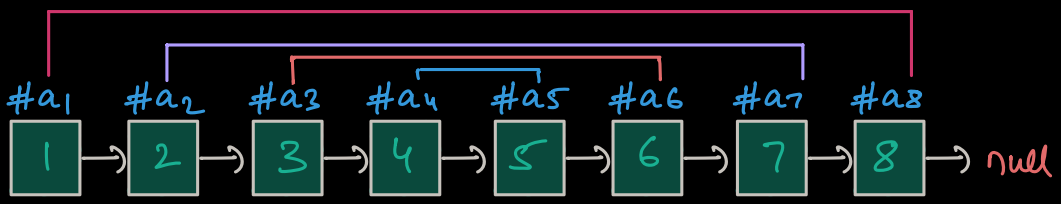
TC: O(n log n)
SC: O(log(n))

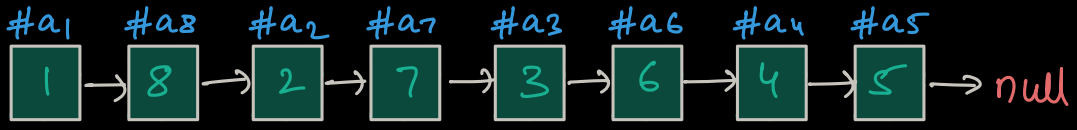
Q3: Silly Qn but asked in interviews

Re-arrange the linked list

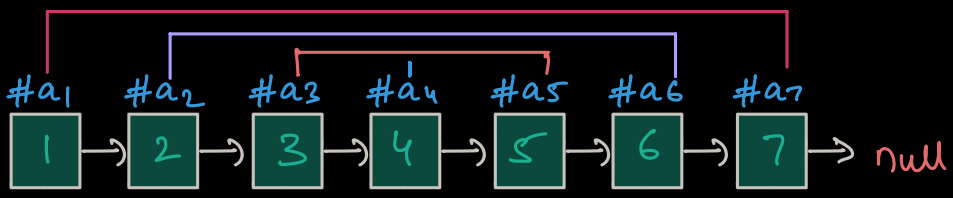
Expected TC: Linear O(N) SC: constant O(1)

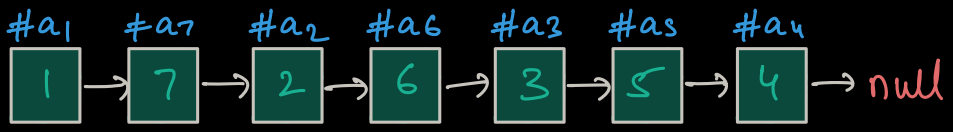
Eg1:



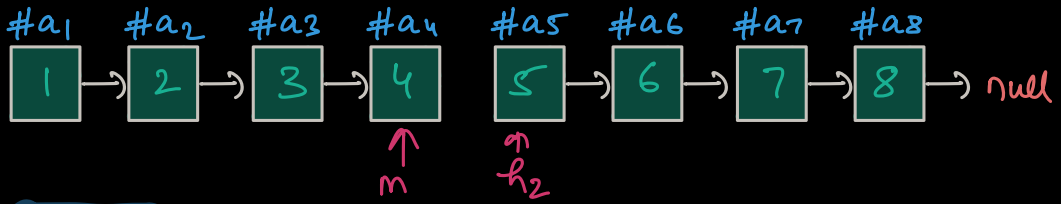


Eg2:

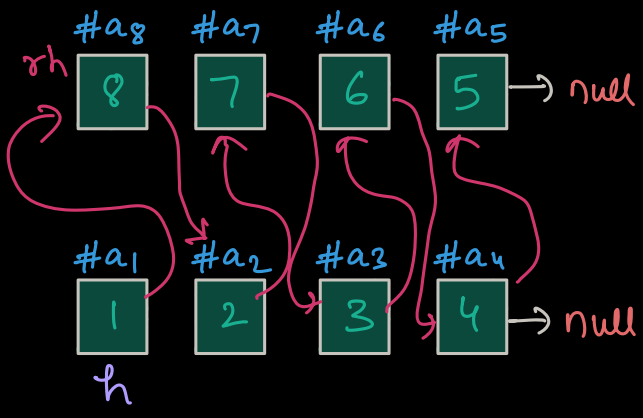




idea: Find the mid of the linked list



Step 2: Reverse the 2nd list (h_2)



TC: O(n)

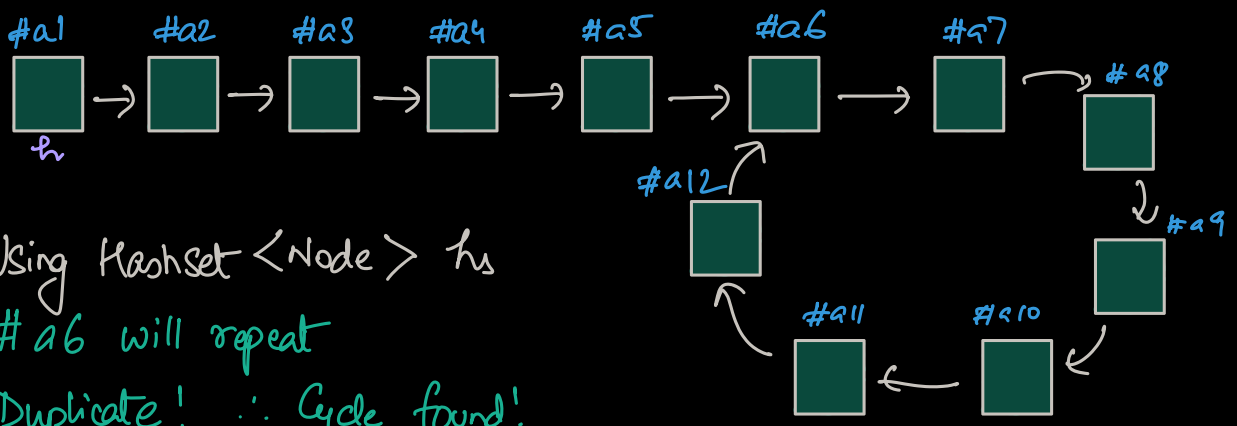
SC: O(1)

Code: To Do

Step 3: Rearrange the list st we take one node from h & other from rh

Q4: Given head node of linked list, check for cycle detection?

Eg1

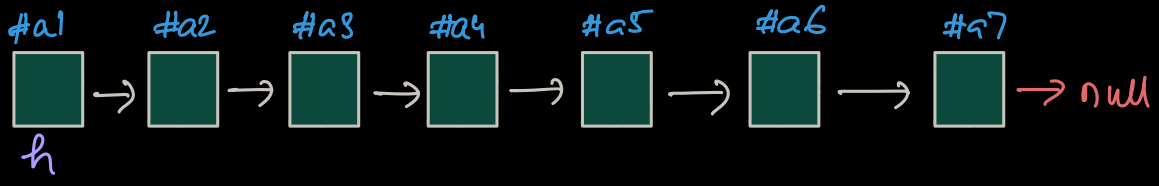


idea1: Using Hashset<Node> hs

#a6 will repeat

⇒ Duplicate! ∴ Cycle found!

Eg2

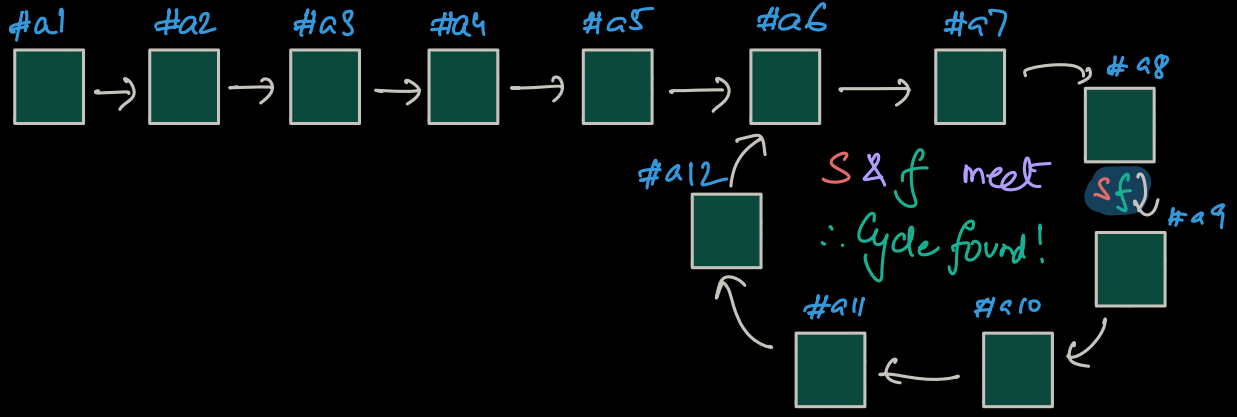


idea 1: Use Extra space

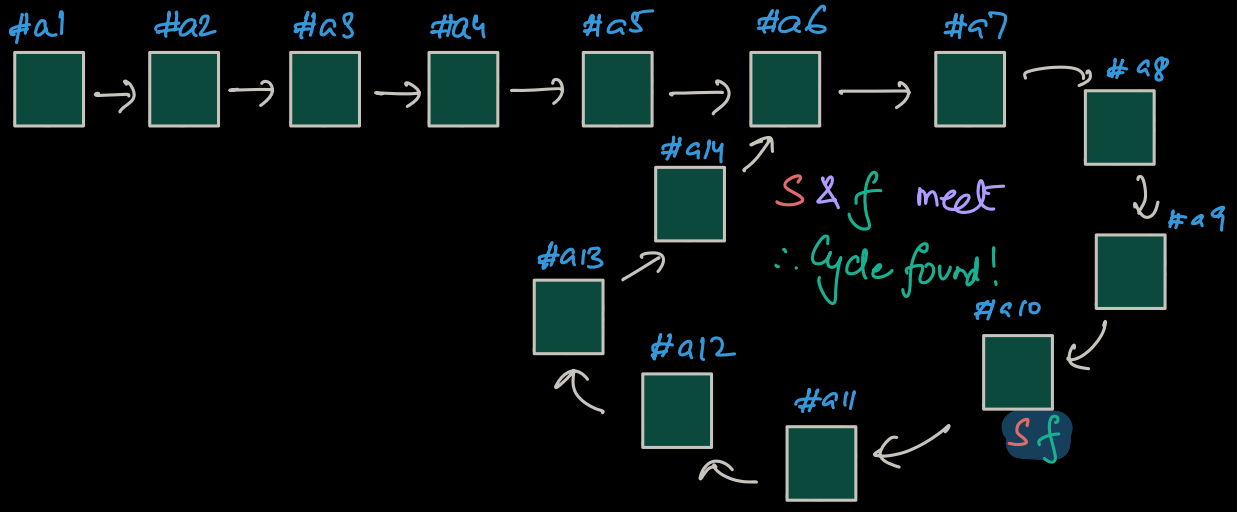
* Iterate on LL & store address in a Hashset<Node>
* If address is present in hs ⇒ Cycle found!
  Else if we reach null ⇒ cycle not found!

idea 2: W/o Extra Space [Floyd's Cycle Detection Algorithm]

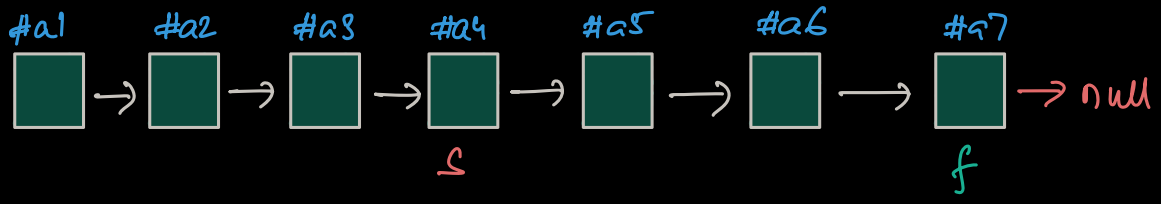
Eg 1:



Eg 2:



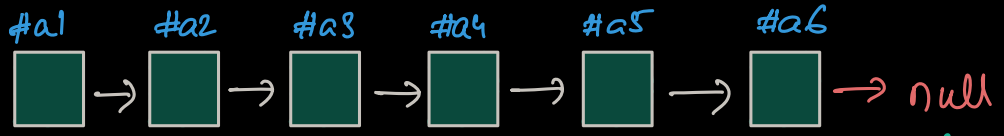
Eg 3:



f.next == null

STOP!

Eg 4:

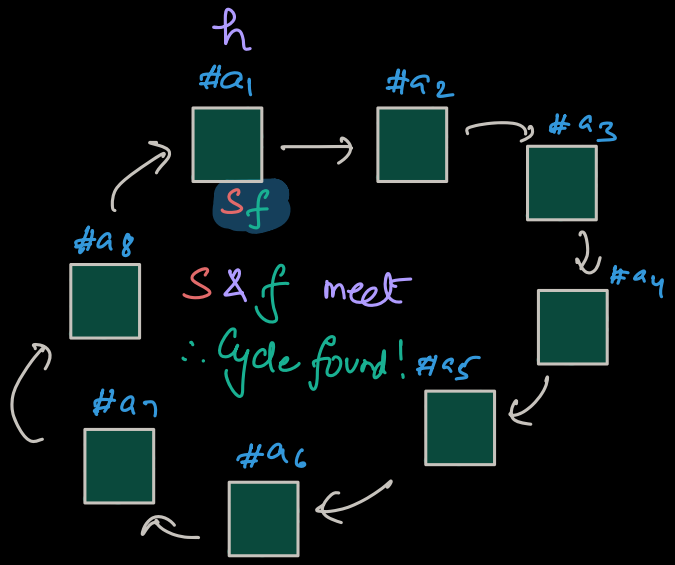


Cycle not found!

f == null

STOP!

Eg 5:



idea2: Take slow & fast pointers & initialize to head.

s = s.next

f = f.next.next

Both meet / intersect ⇒ Cycle found!

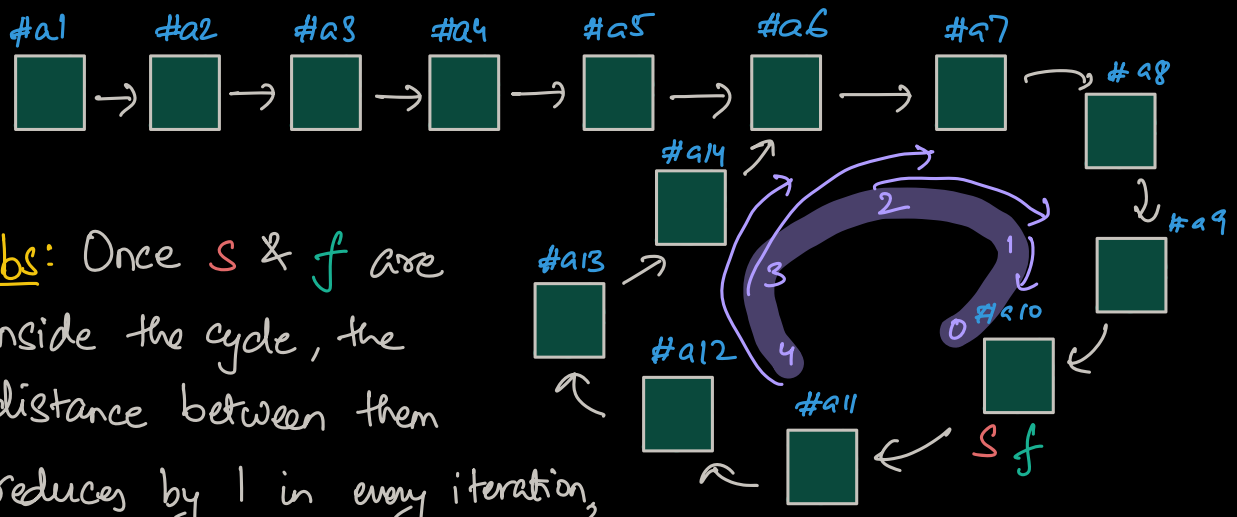
Pseudocode:

```
bool checkCycle(Node h) {
    Node s = h
    Node f = h
    bool isCycle = false
    while(f != null && f.next != null) {
        s = s.next
        f = f.next.next
        if(s == f) {
            isCycle = true
            break
        }
    }
    return isCycle
}
```

TC : O(n)

SC : O(1)

Q: Why does this work?



Obs: Once s & f are inside the cycle, the distance between them reduces by 1 in every iteration, until they meet & distance becomes 0.

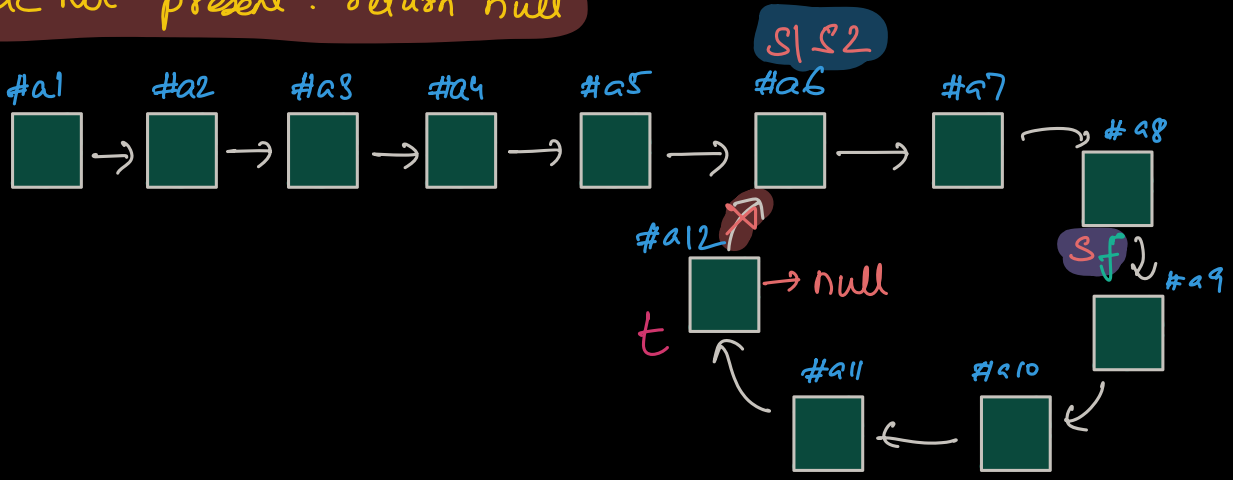
Qn: Given a linked list

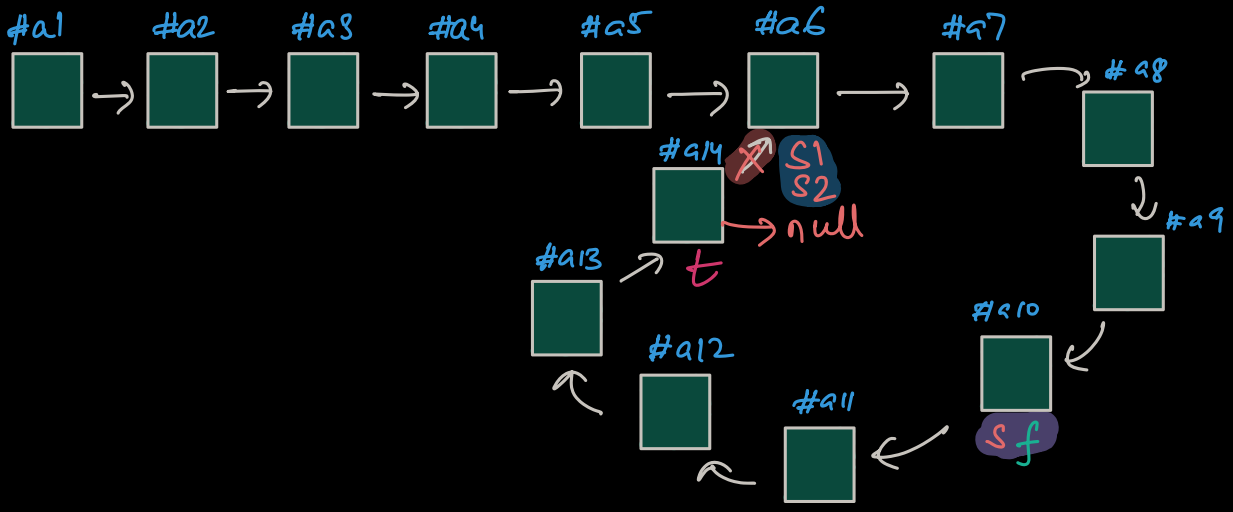
: if cycle is present :

* Remove the cycle
* Return start of cycle

if cycle not present : return null

Eg 1:





* Find Start of cycle [proof : in doubt session]

Take 2 slow pointers s1, s2

Keep updating them one by one

Their meeting point / point of intersection is start of cycle

$s_1 = h$

$s_2 = s/f$

* Remove the cycle

Take a temp pointer & initialize it to start of cycle or s/f

Keep iterating until t.next = start of cycle

then make t.next = null.

```
bool checkCycle(Node h) {
    Node s = h
    Node f = h
    bool isCycle = false
    while(f != null && f.next != null) {
        s = s.next
        f = f.next.next
        if(s == f) {
            isCycle = true
            break
        }
    }
    if(isCycle == false) { return null }
    // Find start of cycle
    Node s1 = h, s2 = s // or f (both are same)
    while(s1 != s2) {
        s1 = s1.next
        s2 = s2.next
    }
    return s1/s2 // Start of cycle = s1
    Node t = s1
    while(t.next != s1) { t = t.next }
    t.next = null
    return s1
}
```

Detect cycle

Find start of cycle

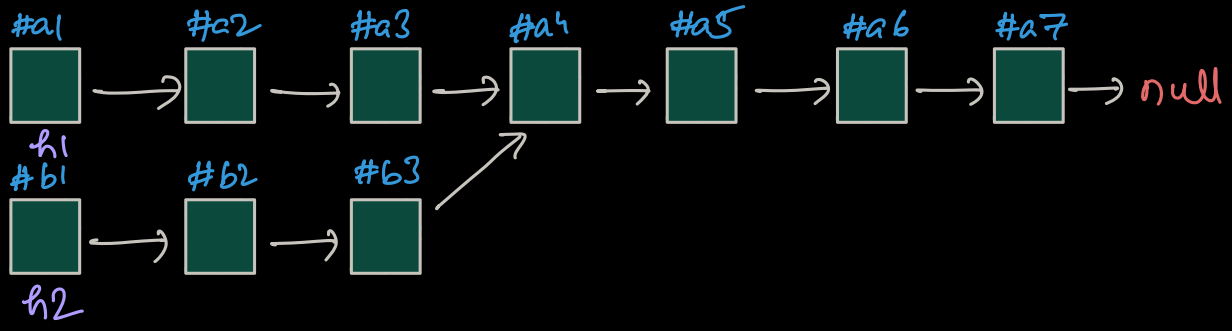
Remove cycle

TC: O(n)

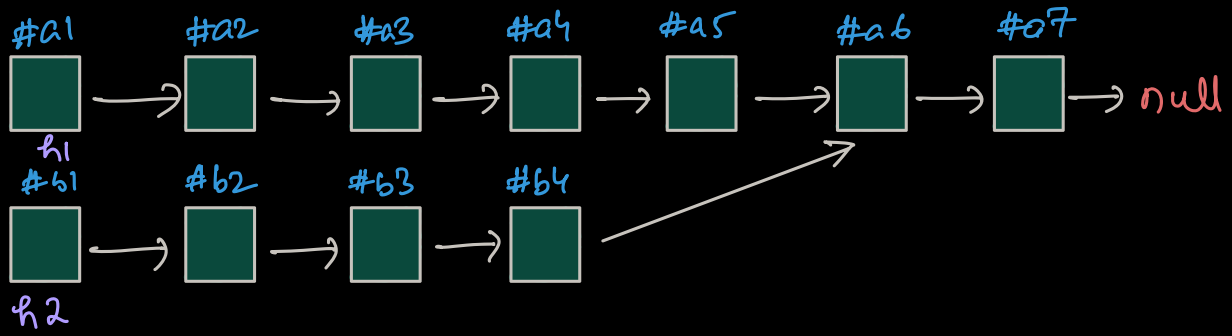
SC: O(1)

Last Qn: Find intersection point of 2 linked lists.

Eg 1:



Eg 2:



idea 1: Use a hashset <Node> hs TC: O(n+m), SC: O(n)

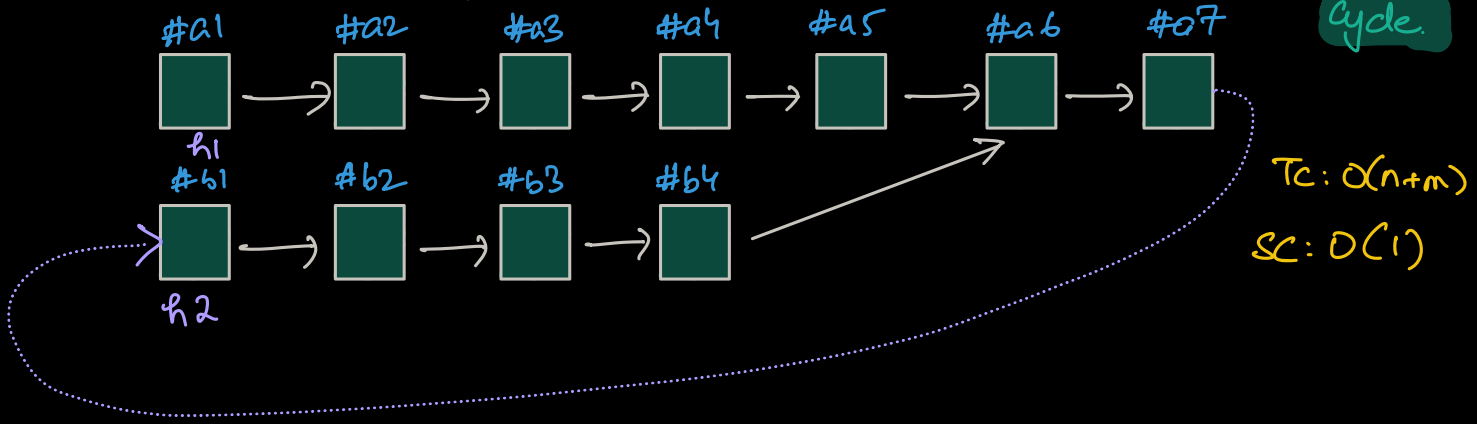
* Iterate over h_1 & store in hs
* Iterate over h_2 & if duplicate found ⇒ Intersection pt.

idea 2: Use a Hashmap <Node, int> TC: O(n+m), SC: O(n)

* Iterate over h_1 & store in hm
* Iterate over h_2 & if freq == 2

duplicate found ⇒ Intersection pt

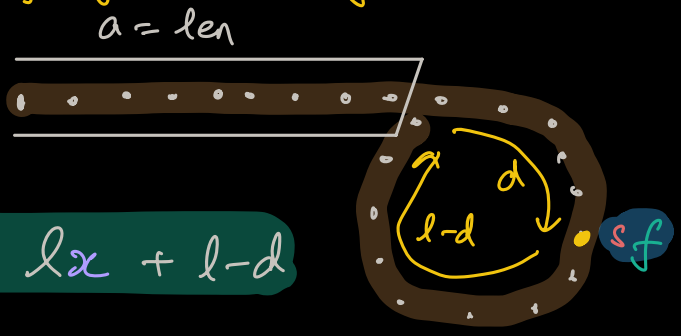
idea 3: Join end of one list with start of another. Find start of cycle.



TC: O(n+m)

SC: O(1)

Proof for start cycle {doubt session}



length of cycle = l

$a = lx + l - d$

s & f intersect at d.

remaining length = $l - d$.

d_s = distance travelled by s = $a + C_s * l + d$

d_f = distance travelled by f = $a + C_f * l + d$

$$d_f = 2 \times d_s$$

$$a + C_f * l + d = 2 \times [a + C_s * l + d]$$

$$\cancel{d} + C_f l + \cancel{d} = 2a + 2C_s l + 2d$$

$$0 = a + d + 2C_s l - C_f l$$

Add & Subtract l

$$0 = a + d + 2C_s l - C_f l + l - l$$

$$a = C_f l - 2C_s l - d + l - l$$

$$a = l[C_f - 2C_s - 1] + l - d$$

$C_f - 2C_s - 1 = x$

$$\Rightarrow \quad a = lx + l - d$$