Todays Content

1. Cycle Detection
2. If cycle Exists remove cycle
3. Intersection of 2 Linked List
4. Proof for cycle detection

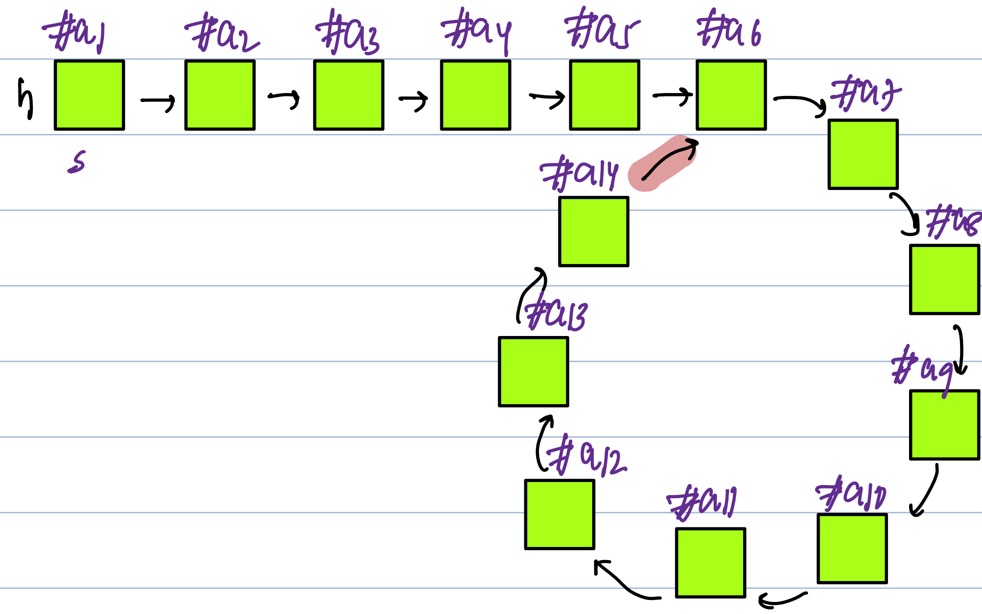
Q1 Given a head node of Linked List, check for cycle detection?

#Note1: When last node points to any prev node, that's when cycle formed

#Note2: If no cycle return null.

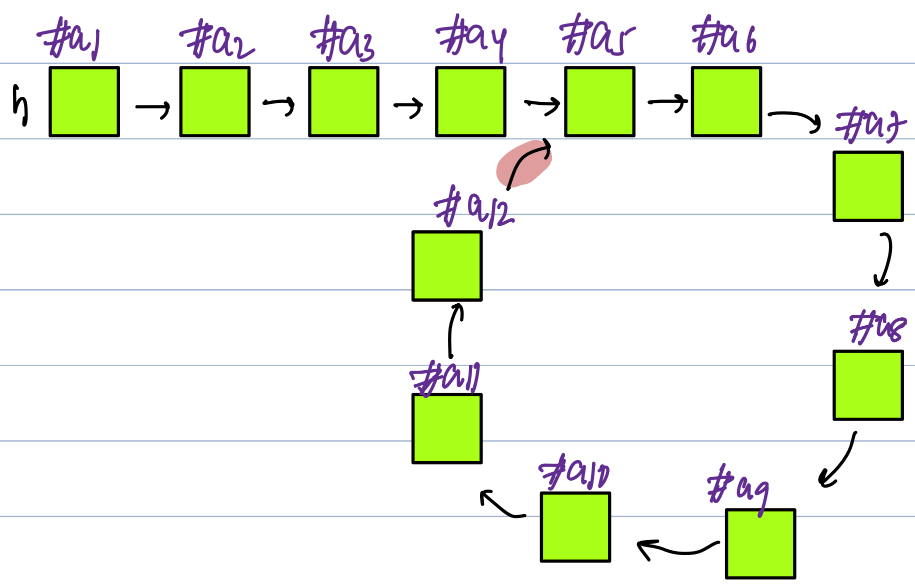
#Note3: If cycle is there, remove cycle {last node → nullptr}
Return start/first node of cycle.

Ex1:



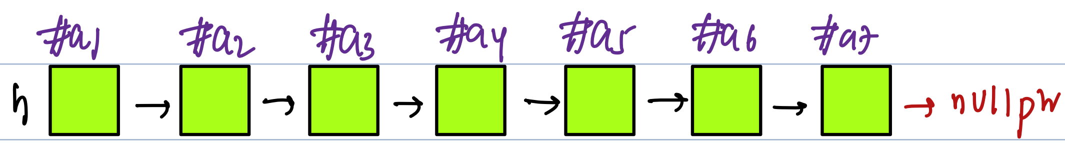
Cycle exist
1. break cycle
2. return a6

Ex2:



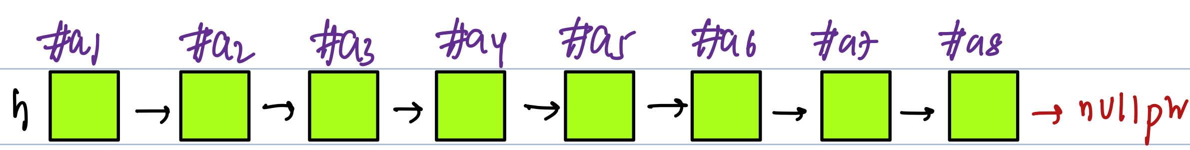
Cycle exist
1. break cycle
2. return a5

Ex3:



return nullptr

Ex4:



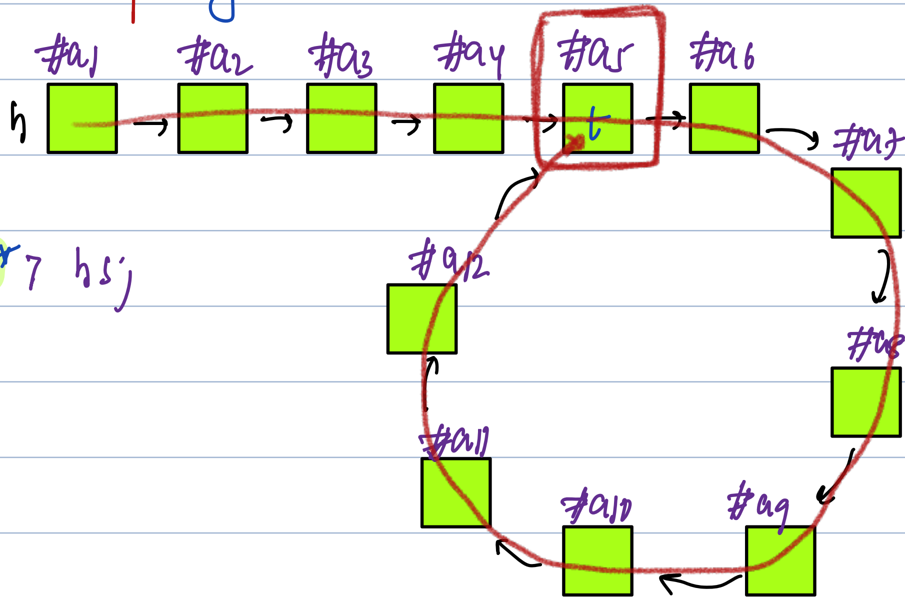
return nullptr

Idea1: Create a hashset<Node*> TC: O(N) SC: O(N) → HashSet

Iterate on LinkedList & Insert each node address.

obs1 # If a node address already exists: cycle exists

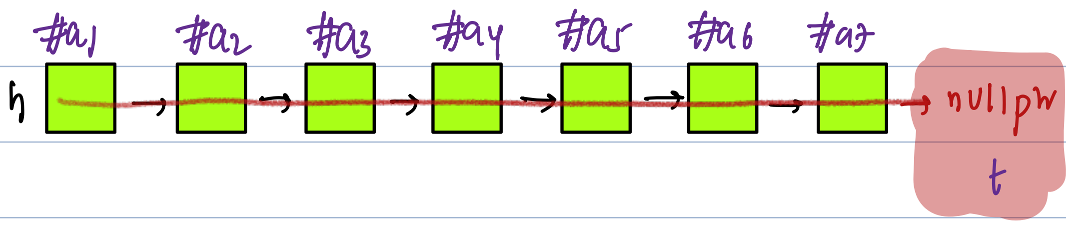
obs2 # If t == nullptr cycle doesn't exist



unordered_set<Node*> hs;

#a1	#a2	#a3
#a4	#a5	#a6
#a7	#a8	#a9
#a10	#a11	#a12

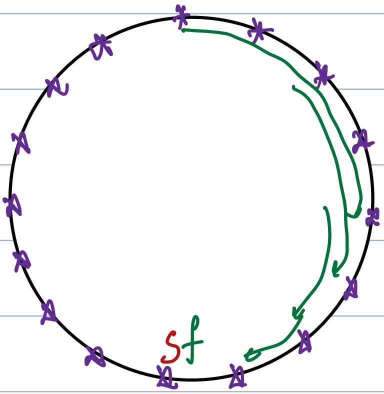
Ex2:



#a1	#a2	#a3
#a4	#a5	#a6
#a7		

```
Node* cycle(Node* h) {

}
```

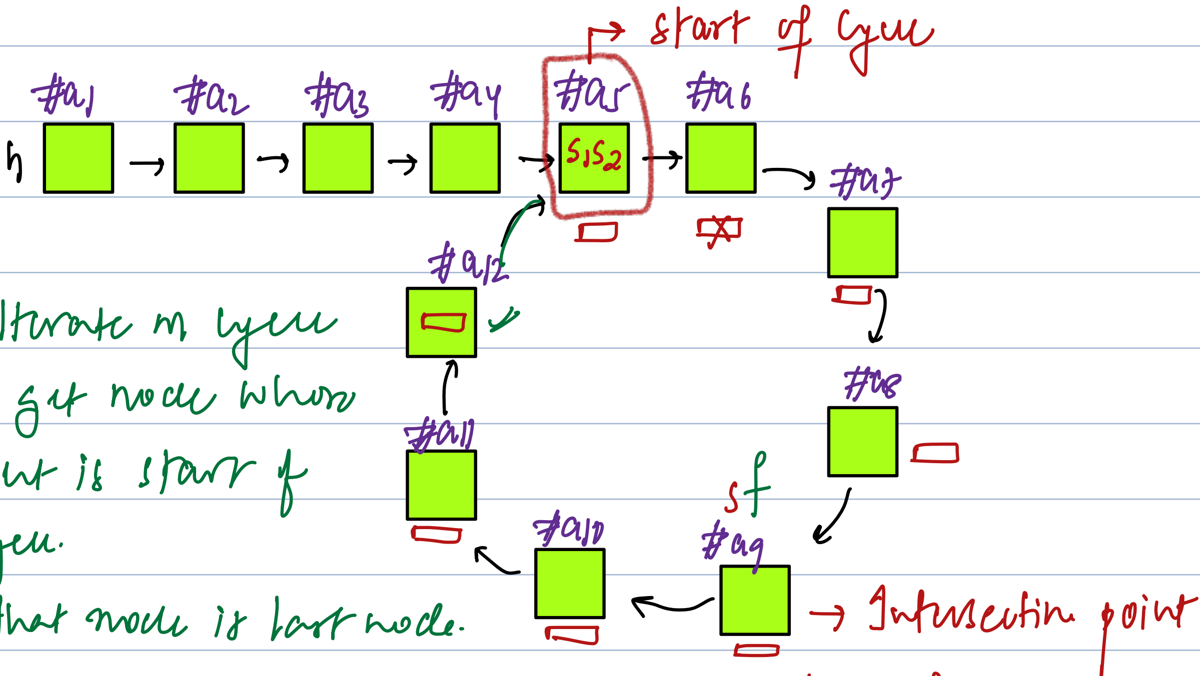
Idea2:



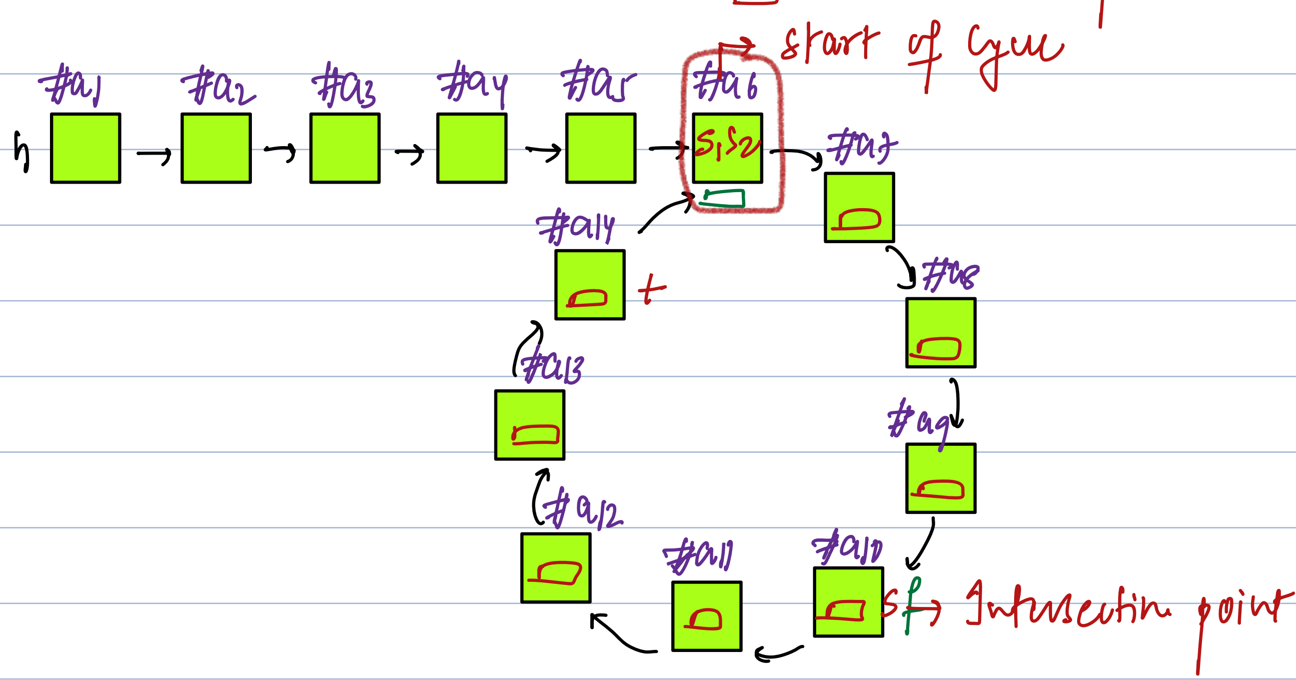
Distance between them

Initial: 4 → 3 → 2 → 1 → 0

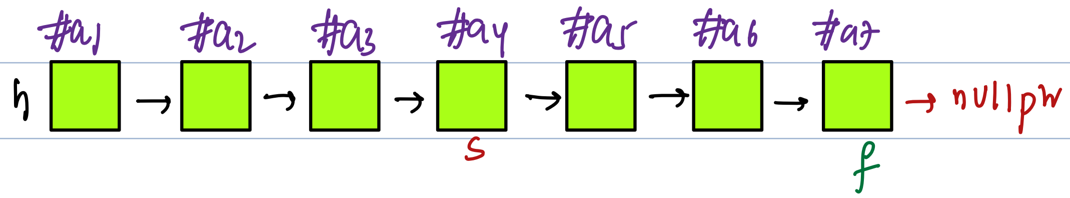
In a circle, if 1: step : 2 steps, irrespective of initial distance, they will meet, because distance between will decrease by 1 step after each iteration in circle {d=0, meeting each other}



#Iterate on cycle & get node whose next is start of cycle.
That node is last node.

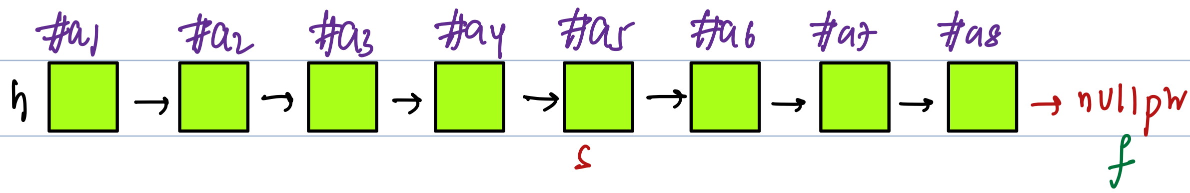


Ex3:



#Con1: if (f→next == nullptr) { No cycle }

Ex4:



#Con2: if (f == nullptr) { No cycle }

Include both Con1 & Con2, because we are not sure of even or odd length

Steps:

1.

2.

3.

Q) Given a head node of Linked List, check for cycle detection?

#Note1: When last node points to any prev node, That's when cycle formed

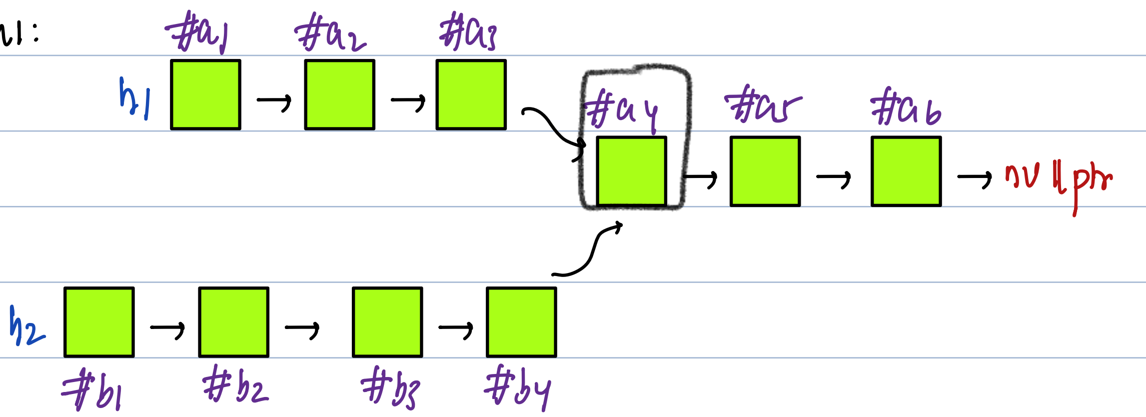
#Note2: If no cycle return null.

#Note3: If cycle is there, remove cycle {last node → nullptr}
Return start/first node of cycle.

```
Node* detectCycle(Node *h) {   TC: O(N)   SC: O(1)

    Node *s = h, *f = h;                              # cycle detection
    bool cycleExist = false;
    while(f != nullptr && f->next != nullptr){
        s = s->next
        f = f->next->next;
        if(s == f){
            cycleExist = true;
            break;
        }
    }
    if( cycleExist == false){
        return nullptr;
    }

    Node *s1 = h, *s2 = s;                            # start of cycle
    while( s1 != s2){
        s1 = s1->next;
        s2 = s2->next;
    }

    Node *start = s1;                                 # breaking cycle
    Node *temp = start;
    while( temp->next != start){
        temp = temp->next;
    }
    temp->next = nullptr;

    return start;
}
```

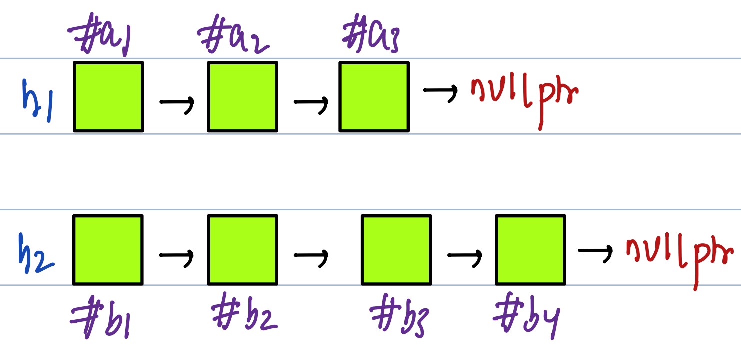
28 Given the heads of two singly linked-lists
Return nodes at which the two lists intersect.
If the two linked list have no intersection at all, return null.

Ex1:



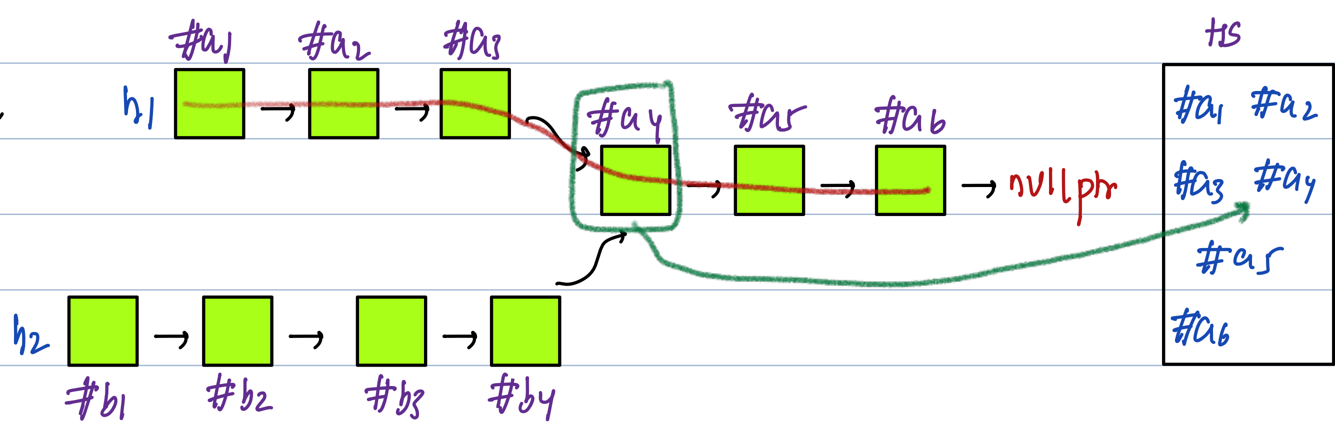
Output
a4

Ex2:



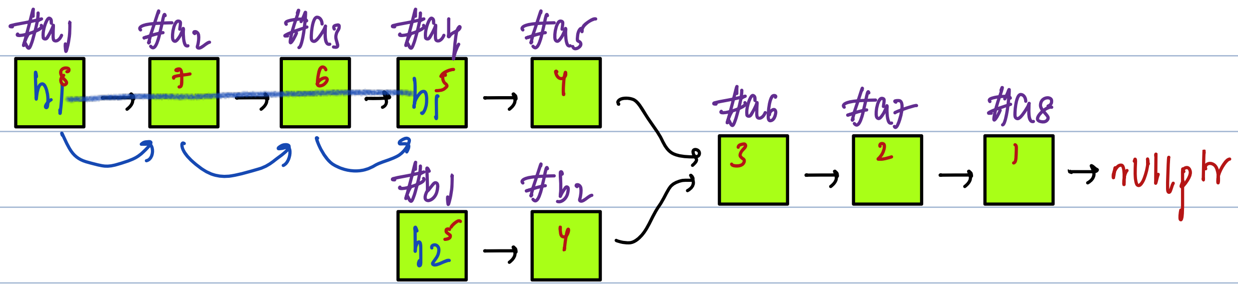
Output
nullptr

Idea:



1. Iterate on h1 & Insert all address in hashset<Node*> hs
2. Iterate on h2:
   if node exists in hs: Return node
   if not, goto next node.

Ex2:



Idea: TC: O(N+M) SC: O(1)

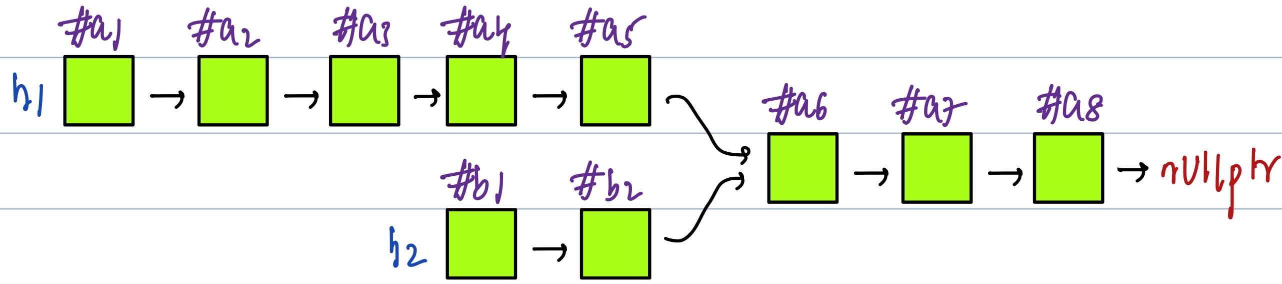
Step1:

```
l1 = size(h1), l2 = size(h2);
if( l1 > l2 ){ # Extra nodes l1-l2
    # update h1 (l1-l2) times
}
else if( l2 > l1 ){ # Extra nodes l2-l1
    # update h2 (l2-l1) times
}
```

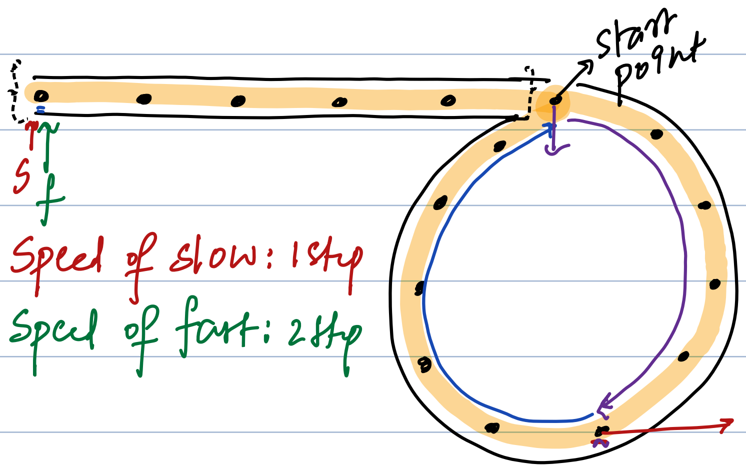
Step2:

```
while(h1 != nullptr){
    if( h1 == h2 ){ return h1; }
    h1 = h1 -> next;
    h2 = h2 -> next;
}

return nullptr;
```

#Ideas:



Cycle Detection: Proof



Speed of slow: 1step
Speed of fast: 2step

l = length of cycle

Note: Both s & f move same amount of time.

d_s: $a + kl + d$ // k: No: of times s pointer iterating in cycle
d_f: $a + ml + d$ // m: No: of times f pointer iterating in cycle

$$d_f = 2 * d_s;$$

$$a + ml + d = 2[a + kl + d]$$

$$\cancel{a} + ml + \cancel{d} = \cancel{2}a + 2kl + \cancel{2}d$$

$$ml = a + 2kl + d$$

$$ml - 2kl - d = a$$

$$\boxed{a = l[m - 2k] - d;}$$

$$\boxed{a = ly - d;}$$