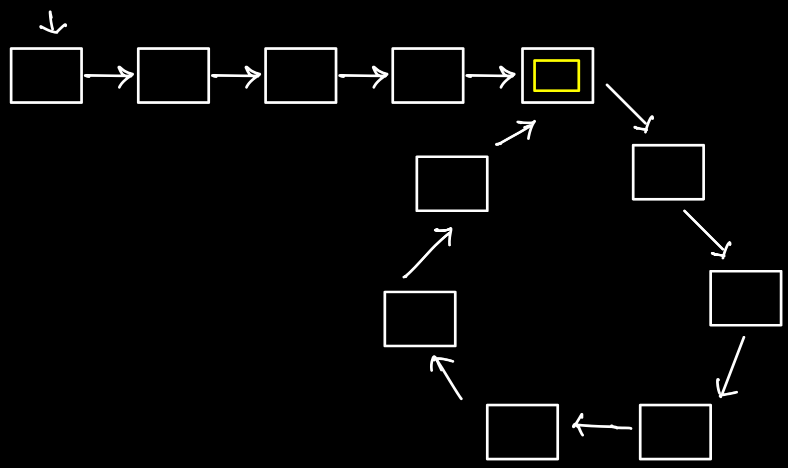
Cycle Detection in L·L

- Detect if there's a cycle in L·L
- find the first node | start node of the cycle.



① HashMap / HashSet.

HashSet<Node> set;

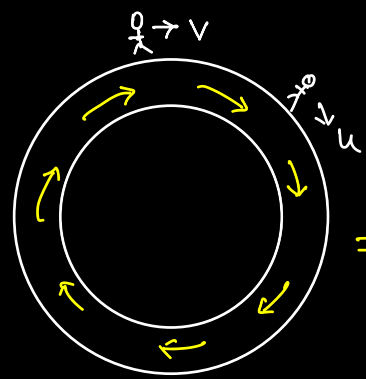
→ Iterate over the LL, if the node is already present in the set then there's a cycle.

Steps: Iterate over the LL:
for every node:
Check if it is present in the set or not ⇒
{ if yes ⇒ return true;
{ else ⇒ insert node in the Set & move to next.

this Node is the start of the cycle.

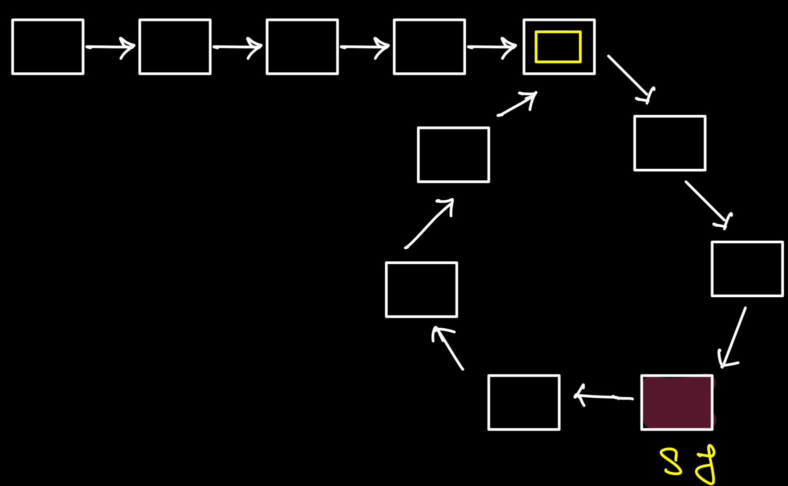
$$\boxed{\begin{array}{l} TC : O(N) \\ SC : O(N) \end{array}}$$

② 2 pointers.

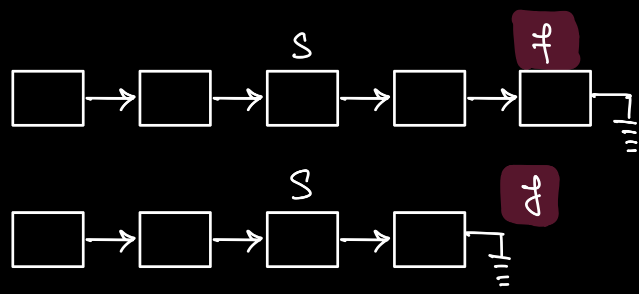


$V > u$

⇒ Runners will cross eachother

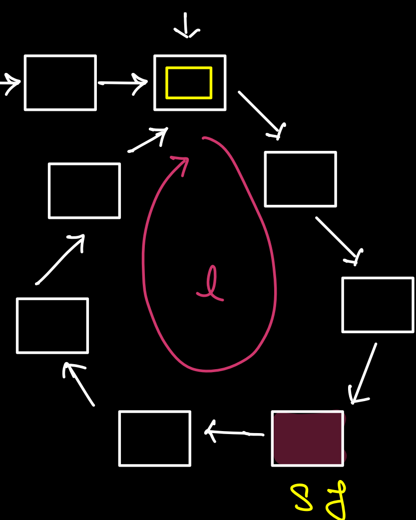


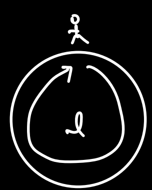

```
slow = head;
fast = head;
while( fast != Null && fast.next != null ){
    slow = slow.next
    fast = fast.next.next
    if (slow == fast) return true;
}
return false;
```



find the start of the cycle :-

l: length of the cycle

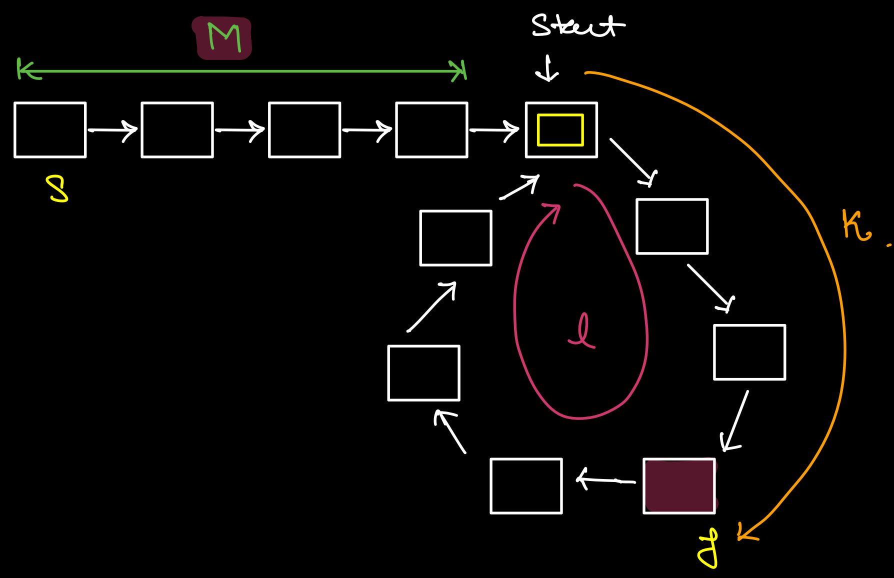




If the length of the cycle is l & we start with the first node of cycle then :

after	l	iterations	→	we'll	reach	start	again
"	$2l$	"	→	"	"	"	"
"	$9l$	"	→	"	"	"	"
"	xl	"	→	"	"	"	"

x ↑ int



$$\boxed{\begin{aligned} dist(fast) &= M + x\ell + k \\ dist(slow) &= M + y\ell + k \end{aligned}}$$

$$dist(fast) = 2 * dist(slow)$$

$$M + x\ell + k = 2(M + y\ell + k)$$

$$M + x\ell + k = 2M + 2y\ell + 2k$$

$$(\underbrace{x - 2y})\ell = M + k$$

$$P * \ell = M + k$$

$\underbrace{M+k}$ ⇒ integer multiple of ℓ.

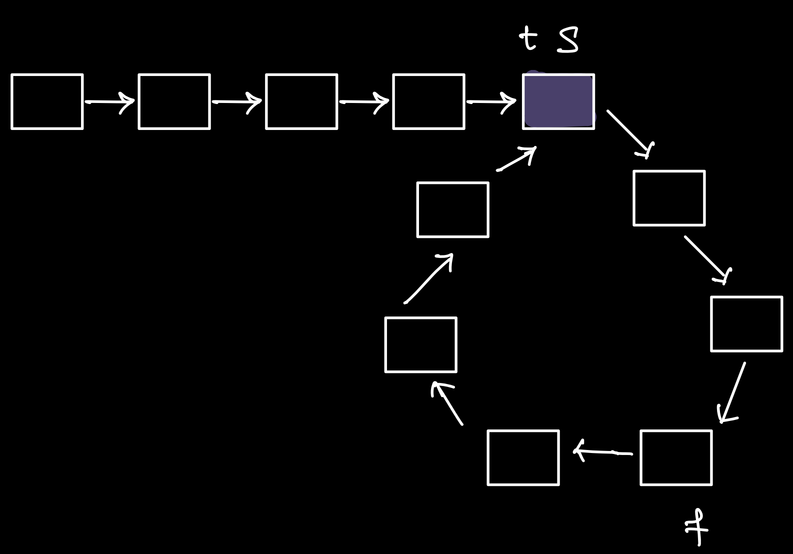
⇒ If we perform M+K iterations from the start node of the cycle then we'll end up at start node again, because M+K is an integer multiple of l.

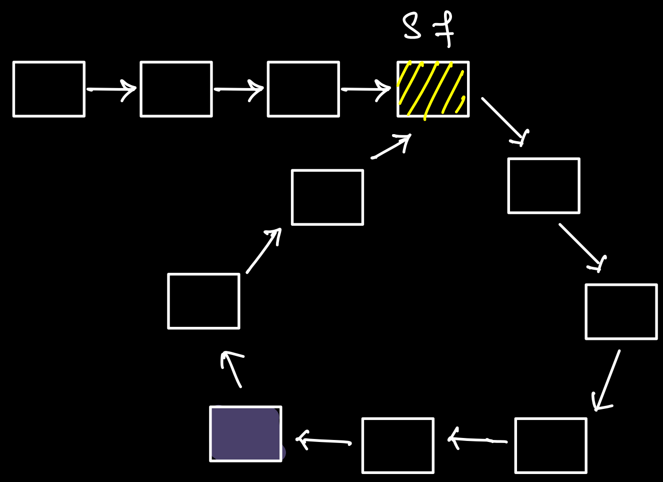
⇒ The meeting point is at K distance from start node, so M iterations from the meeting point will take us to the start node.

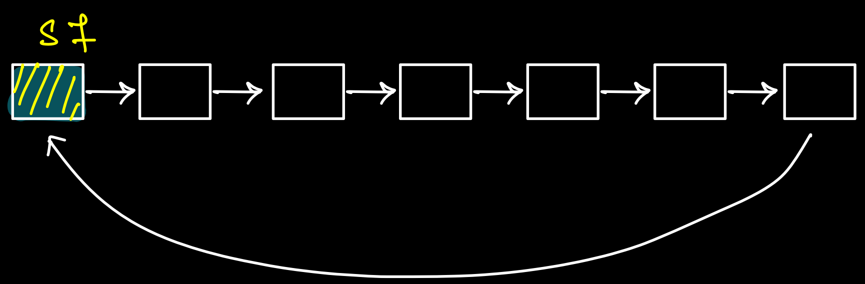
⇒ To do M iterations from the meeting point,

Start one node from the head & other from the meeting point and move them by one pointer each

⇒ These pointers will meet at start node of the cycle.







TC : O(N)

SC : O(1)

Floyd's Cycle Detection Algorithm.

Spotify : 10M songs.

⇒ Caching (LRU Cache Implement)

Cache
↳ Small piece of memory which is very fast to access.
→ Cache H/w is very cost.
→ It reduces query time significantly.

Spotify.

Naatu

Cache

295
Calm down
Enemy
Bones

→ Cache Eviction Policy.

- Least Recently Used (LRU)
- Least frequently Used

⋮

Cache

MRP	Naatu
	Calm down
	Bones
LRP	Enemy

LRU.

~~295~~

- Search (x)
- insert (x) at Most recent Posⁿ
- delete ()

	Array	LL	DLL + HM
Search	O(N)	O(N) → O(1) HM/set	O(1)
delete	O(N) (Shifting)	O(1) (Search is already done)	O(1)
insert	O(1)	O(1)	O(1)

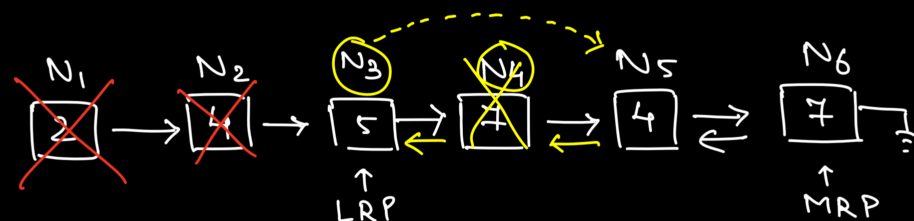
Cache_capacity = 3

delete at LRU
insert at MRU.

HashMap
<int, node>

~~2 : N_1~~
~~4 : N_2~~
5 : N_3
~~7 : N_4~~
4 : N_5
7 : N_6

2 4 5 7 4 7



⇒ Doubly Linked List. (DLL)

x

Search (x)

Not found

Cache Miss

if (size == capacity)

Yes

- Remove from LRU
- Insert at MRU.

NO

- Insert at MRU

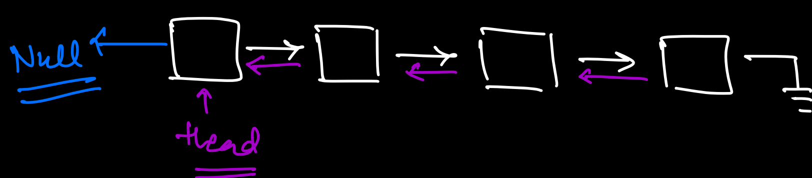
found

Cache Hit

- Remove x from current Pos^n
- Insert x at MRU.

⇒ DLL + HashMap : All operations are supported in O(1)

```
Class DLL {
    int data;
    DLL next;
    DLL prev;
    DLL(x) {
        data = x;
        next = NULL;
        prev = NULL;
    }
}
```

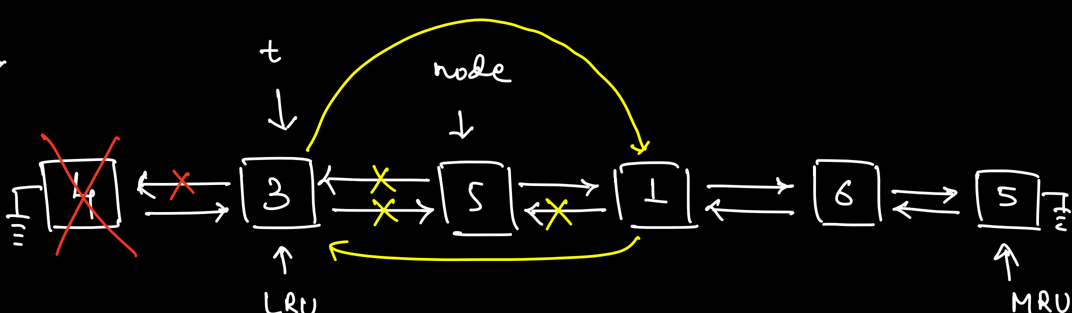


Dry-run

Capacity = 4   4   3   5   1   6   5   1

HM
<int, DLL>

~~4 : N1~~
3 : N_2
5 : ~~N_3~~ N_6
1 : N_4
6 : N_5



—— * ——