Linked List Cycle-II

BRUTE FORCE :

Using hashing we store the entire node in the map and check if the node already exist then we continue in the loop as soon as a node is found we return that node.

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        unordered_map<ListNode*,int> m;
        ListNode* temp=head;
        while(temp!=NULL)
        {
            if(m.find(temp)!=m.end())
                return temp;
            m[temp]++;
            temp=temp->next;
        }
        return NULL;
    }
};
```

- Time Complexity : O(N)
- Space Complexity : O(N)

Optimal Approach :

Using two pointers fast and slow if a cycle exist then fast and slow will point to one element at some point. fast will move 2 position but slow will change 1 point.

if fast→next≠NULL && fast→next→next≠NULL we need to check for cycle these consitions make sure that if cycle does not exist then NULL is returned when loop ends.

But if cycle is there then colliding point will be when fast==slow

Next we will position fast to head and keep moving both pointers one step only until they collide again the point at which they collide again will be the starting of cycle.

Intuition for this is :

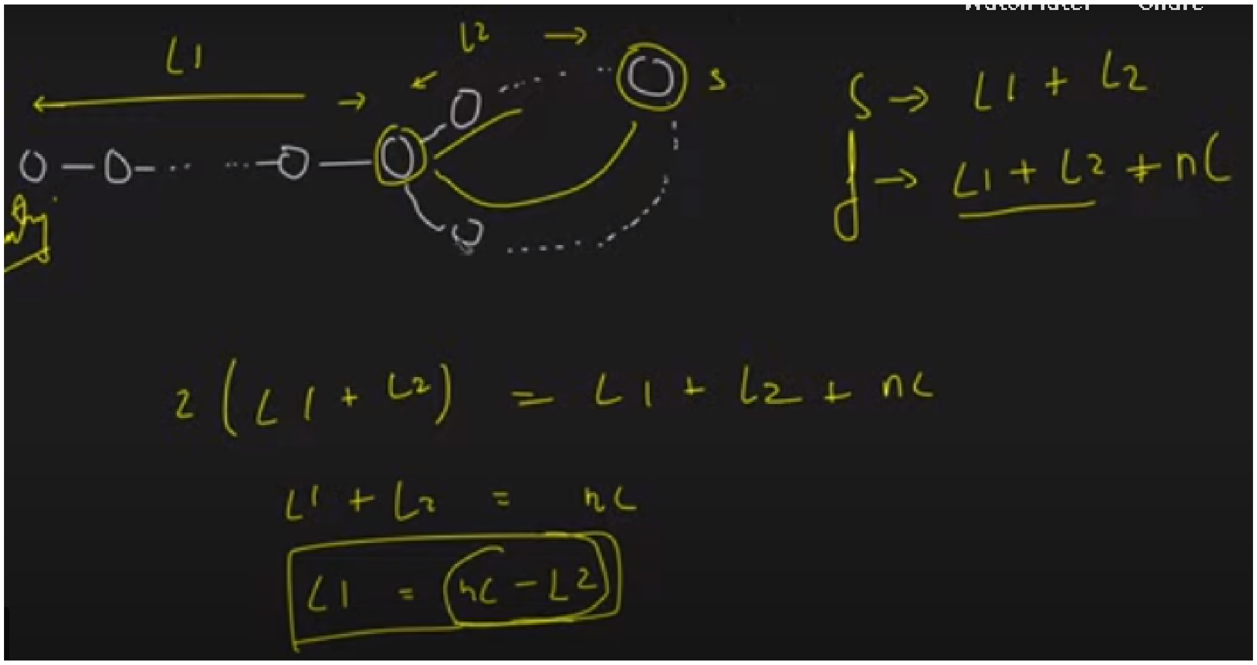
Let us consider that distance from head to cycle starting is l1 and cycle-start to first colliding point be l2.

so the total distance travelled by slow is (l1+l2) but fast travels l1+l2+nC where nC is the number of cycles that fast pointer takes to meet the slow pointer.

Now we know that fast pointer travells 2(l1+l2) distance than slow pointer which makes the eqn:

2(l1+l2) = l1+l2+nC ⇒ l1+l2=nC ⇒ l1=nC-l2

which gives the cycle start equal to cycles- colliding point distance when we place fast at head then when slow traverses nC-l2 it will reach to the starting point of the cycle.



```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        if(head==NULL || head->next==NULL)
            return NULL;
        ListNode* slow=head;
        ListNode* fast=head;
        int found=0;
        while(fast->next!=NULL && fast->next->next!=NULL)
        {
            slow=slow->next;
            fast=fast->next->next;
            if(fast==slow)
            {
                fast=head;
                while(slow!=fast)
                {
                    slow=slow->next;
                    fast=fast->next;
                }
                return slow;
            }
        }

        return NULL;
    }
};
```

- Time Complexity : O(N)
- Space Complexity : O(1)