Linked List Cycle

BRUTE FORCE: Hashing

Keep storing the node in the map if at any point a node already exists in the map then return true otherwise keep iterating the loop until it reaches NULL if the loop is completed and nothing is returned means no cycle was detected so we return false outside the loop.

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(head==NULL || head->next==NULL)
        {
            return false;
        }
        ListNode* fast=head;
        ListNode* slow=head;
        while(fast->next!=NULL && fast->next->next!=NULL)
        {
            slow=slow->next;
            fast=fast->next->next;
            if(fast==slow)
                return true;
        }
        return false;
    }
};
```

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

Optimal Approach :

Use two pointers, slow moves one step whereas fast moves two steps. If at any point slow and fast meet this means cycle is detected because if fast is moving ahead there is no point that fast can come back and be at same position as slow it is only possible when they move circularly. If either fast is the last node or fast is the second last node then the loop will end because there is no way for fast to move ahead.

Edge case to handle :

If the list is empty or contains only 1 node then we need to return false

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if(head==NULL || head->next==NULL)
        {
            return false;
        }
        ListNode* fast=head;
        ListNode* slow=head;
        while(fast->next!=NULL && fast->next->next!=NULL)
        {
            slow=slow->next;
            fast=fast->next->next;
            if(fast==slow)
                return true;
        }
        return false;
    }
};
```

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