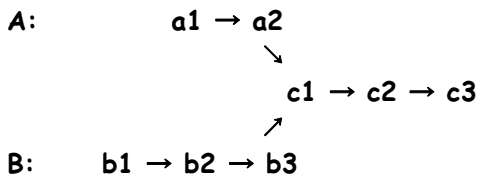
DAY 12

INTERVIEW BIT PROBLEMS :

1. Intersection of Linked Lists

Write a program to find the node at which the intersection of two singly linked lists begins.
For example, the following two linked lists:



begin to intersect at node c1.

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

CODE :

PYTHON

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    # @param A : head node of linked list
    # @param B : head node of linked list
    # @return the head node in the linked list
    def getIntersectionNode(self, A, B):
        c1=0
        c2=0
        head1=A
        head2=B

        while A is not None:
            A=A.next
            c1+=1

        while B is not None:
            B=B.next
```

```
            c2+=1

        A=head1
        B=head2
        if c1>c2:
            for i in range(c1-c2):
                A=A.next
        else:
            for i in range(c2-c1):
                B=B.next

        for i in range(min(c1,c2)):
            if A is B:
                return A
            A=A.next
            B=B.next

         return None
```

C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *      int val;
 *      ListNode *next;
 *      ListNode(int x) : val(x), next(NULL) {}
 * };
 */
ListNode* Solution::getIntersectionNode(ListNode* A, ListNode* B) {
    if(A==NULL || B==NULL){
        return NULL;
    }
    ListNode *head1=A;
    ListNode *head2=B;

    while(head1!=head2){
        if(head1==NULL){
            head1=B;
        }
        else{
            head1=head1->next;
        }
        if(head2==NULL){
            head2=A;
        }
        else{
            head2=head2->next;
        }
    }
```

```
    return head1;
}
```

2. Reverse Linked List

Reverse a linked list. Do it in-place and in one-pass.
For example:
Given 1->2->3->4->5->NULL,
return 5->4->3->2->1->NULL.

CODE :

PYTHON

```
# Definition for singly-linked list.
# class ListNode:
#       def __init__(self, x):
#           self.val = x
#           self.next = None

class Solution:
    # @param A : head node of linked list
    # @return the head node in the linked list
    def reverseList(self, A):
        previous=None
        next_one=None
        current=A
        if current is None:
            return None
        while current is not None:
            next_one=current.next
            current.next=previous
            previous=current
            current=next_one
        return previous
```

C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *      int val;
 *      ListNode *next;
 *      ListNode(int x) : val(x), next(NULL) {}
 * };
 */
ListNode* Solution::reverseList(ListNode* A) {
    ListNode *previous=NULL;
    ListNode *current=A;
    ListNode *next_node=NULL;
```

```
    if(current==NULL){
        return NULL;
    }
    while(current!=NULL){
        next_node=current->next;
        current->next=previous;
        previous=current;
        current=next_node;
    }
    return previous;
}
```

3. K reverse linked list

Given a singly linked list and an integer K, reverses the nodes of the list K at a time and returns modified linked list.

NOTE : The length of the list is divisible by K

Example :
Given linked list 1 -> 2 -> 3 -> 4 -> 5 -> 6 and K=2,
You should return 2 -> 1 -> 4 -> 3 -> 6 -> 5
Try to solve the problem using **constant extra space.**

CODE :

PYTHON

```
# Definition for singly-linked list.
# class ListNode:
#   def __init__(self, x):
#       self.val = x
#       self.next = None

class Solution:
    # @param A : head node of linked list
    # @param B : integer
    # @return the head node in the linked list
    def reverseList(self, A, B):
        current=A
        previous=None
        next_node=None
        i=0

        while(current is not None and i<B):
            i+=1
            next_node=current.next
            current.next=previous
            previous=current
            current=next_node
```

```
        if next_node is not None:
            A.next=self.reverseList(next_node,B)
        return previous
```

4. Palindrome List

Given a singly linked list, determine if its a palindrome. Return 1 or 0 denoting if its a palindrome or not, respectively.
For example,
List 1-->2-->1 is a palindrome.
List 1-->2-->3 is not a palindrome.

CODE :

C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *      int val;
 *      ListNode *next;
 *      ListNode(int x) : val(x), next(NULL) {}
 * };
 */
int Solution::lPalin(ListNode* A) {
    ListNode *head=A;
    ListNode *runner=A;
    stack<int>pal_stack;

    while(runner!=NULL){
        pal_stack.push(runner->val);
        runner=runner->next;
    }
    while(head!=NULL){
        int check=pal_stack.top();
        pal_stack.pop();
        if(head->val!=check){
            return false;
        }
        head=head->next;
    }
    return true;
}
```

PYTHON

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
```

```
#         self.next = None

class Solution:
    # @param A : head node of linked list
    # @return an integer
    def lPalin(self, A):
        l=0
        head=A
        previous=None
        next_node=None

        while A:
            l+=1
            A=A.next

        A=head
        first_half=l//2
        while first_half>0:
            first_half-=1
            next_node=A.next
            A.next=previous
            previous=A
            A=next_node

        if l%2==1:
            A=A.next

        while A:
            if previous.val!=A.val:
                return 0
            A=A.next
            previous=previous.next

        return 1
```

5. Remove Duplicates from Sorted List

Given a sorted linked list, delete all duplicates such that each element appear only once.
For example,
Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

CODE :

PYTHON

```
class Solution:
    # @param A : head node of linked list
    # @return the head node in the linked list
    def deleteDuplicates(self, A):
```

```
        temp=A
        if temp is None:
            return
        while temp.next is not None:
            if temp.val==temp.next.val:
                insert=temp.next.next
                temp.next=None
                temp.next=insert
            else:
                temp=temp.next
        return A
```

6. Remove Duplicates from Sorted List II

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.
For example,
Given 1->2->3->3->4->4->5, return 1->2->5.
Given 1->1->1->2->3, return 2->3.

CODE:

PYTHON

```
class Solution:
    # @param A : head node of linked list
    # @return the head node in the linked list
    def deleteDuplicates(self, A):
        current=A
        head=previous=ListNode(-1)
        head.next=current
        while current and current.next:
            if current.val==current.next.val:
                while(current and current.next and current.val==current.next.val):
                    current=current.next
                current=current.next
                previous.next=current
            else:
                previous=previous.next
                current=current.next
        return head.next
```

(OR)

```
class Solution:
    # @param A : head node of linked list
    # @return the head node in the linked list
    def deleteDuplicates(self, A):
        my_dict={}
        while A:
            if(my_dict.get(A.val) is None):
                my_dict[A.val]=1
```

```
        else:
            my_dict[A.val]+=1
        A=A.next
head=ListNode(-1)
final=head
for element,count in sorted(my_dict.items()):
        if count==1:
            head.next=ListNode(element)
            head=head.next
return final.next
```