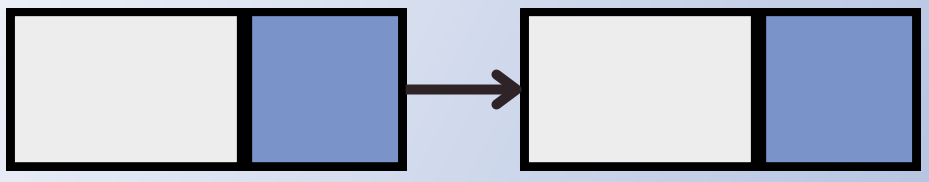
BHUSHAN KALE

BASIC TECHINQUES TO APPROACH

LINKED LIST



BHUSHAN KALE

REMOVE DUPLICATES FROM SORTED LIST

We can just solve it like in an array using another index to collect the valid nodes

Iterative

```
ListNode* deleteDuplicates(ListNode* head) {
  ListNode* dummy = new ListNode(0);
  dummy->next = head;
  ListNode* cur = dummy;
  int duplicate;
  while (cur->next && cur->next->next) {
    if (cur->next->val == cur->next->next->val) {
      duplicate = cur->next->val;
      while (cur->next && cur->next->val == duplicate)
        cur->next = cur->next->next;
    }
    else cur = cur->next;
  }
  return dummy->next;
}
```

BHUSHAN KALE

PALINDROME LINKED LIST

This can be solved by converting this palindrome into a normal vector or else with the help of linked list as follows

```
bool isPalindrome(ListNode* head) {
    if(!head || !head->next) return true;
    ListNode slow = head,fast = head->next;
    while(fast && fast->next) {//split into two halves while the
first half can be one-node longer;
        slow = slow->next;
        fast = fast->next->next;
    }
    fast = slow->next;
    slow->next = NULL;
    ListNode newHead(0); //reverse the second half;
    ListNode next = NULL,p = fast;
    while(p) {
        next = p->next;
        p->next = newHead.next;
        newHead.next = p;
        p = next;
    }
    fast = newHead.next; //compare the two lists;
    while(fast) {
        if(fast->val != head->val) return false;
        fast = fast->next;
        head = head->next;
    }
    return fast == NULL;
}
```

BHUSHAN KALE

ROTATE LIST

For Example:
Given 1->2->3->4->5->NULL and k = 2,
return 4->5->1->2->3->NULL.

```
ListNode* rotateRight(ListNode* head, int k) {
    if(!head) return head;
    int len = 1;
    ListNode *p = head;
    while(p->next) { len++; p = p->next; }
    p->next = head;
    if(k %= len)
        for(int i = 0; i < len-k; ++i, p=p->next) ;
    ListNode* newHead = p->next;
    p->next = NULL;
    return newHead;
}
```

BHUSHAN KALE

ADD TWO NUMBERS

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

```
ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    int c = 0;
    ListNode newHead(0);
    ListNode *t = &newHead;
    while(c || l1 || l2) {
        c += (l1? l1->val : 0) + (l2? l2->val : 0);
        t->next = new ListNode(c%10);
        t = t->next;
        c /= 10;
        if(l1) l1 = l1->next;
        if(l2) l2 = l2->next;
    }
    return newHead.next;
}
```

BHUSHAN KALE

REVERSE LINKED LIST II

Reverse a linked list from position m to n in O(n) time

```cpp
ListNode* reverseBetween(ListNode* head, int m, int n) {
    ListNode newHead(0);
    newHead.next = head;
    ListNode *pre = &newHead, *cur = head, *next = NULL;
    int i = 1;
    while(i < n) {
        if(i++ < m) { pre = cur; cur = cur->next; }
        else {
            next = cur->next;
            cur->next = cur->next->next;
            next->next = pre->next;
            pre->next = next;
        }
    }
    return newHead.next;
}
```

BHUSHAN KALE

LINKED LIST CYCLE II

Given a linked list, return the node where the cycle begins. If there is no cycle, return null. Note: Do not modify the linked list.

```
ListNode *detectCycle(ListNode *head) {
    ListNode *slow = head, *fast = head;
    while(fast && fast->next) {
        slow = slow->next;
        fast = fast->next->next;
        if(slow == fast) break;
    }
    if(slow != fast) return NULL;
    fast = head;
    while(fast && fast->next) {
        if(slow == fast) return slow;
        slow = slow->next;
        fast = fast->next;
    }
    return NULL;
}
```

BHUSHAN KALE

REVERSE NODES IN K-GROUP

Given a linked list, return the node where the cycle begins.
If there is no cycle, return null

```
ListNode* reverseKGroup(ListNode* head, int k) {
  if(!head || !head->next) return head;
  ListNode newHead(0);
  ListNode *pre = &newHead, *cur = head, *next = NULL;
  newHead.next = head;
  int len = 0;
  for(ListNode *p = head; p; p = p->next) len++;
  int times = len/k;
  while(times) {
    for(int i = 1; i < k; ++i) {
      next = cur->next;
      cur->next = cur->next->next;
      next->next = pre->next;
      pre->next = next;
      if(i == k-1) {
        pre = cur;
        cur = cur->next;
      }
    }
    times--;
  }
  return newHead.next;
}
```

BHUSHAN KALE

THANK YOU!!

CONNECT WITH BHUSHAN KALE FOR MORE SUCH AMAZING POSTS!