# Assignment Solutions | Linkedlist - 2 | Week 15

1. You are given the `head` of a linked list. **Delete** the **middle node**, and return *the* `head` *of the modified linked list.* **[Leetcode 2095]**

   The **middle node** of a linked list of size `n` is the `⌊n / 2⌋th` node from the **start** using **0-based indexing**, where `⌊x⌋` denotes the largest integer less than or equal to `x`.

   ○ For `n = 1`, `2`, `3`, `4`, and `5`, the middle nodes are `0`, `1`, `1`, `2`, and `2`, respectively.

   Example 1:
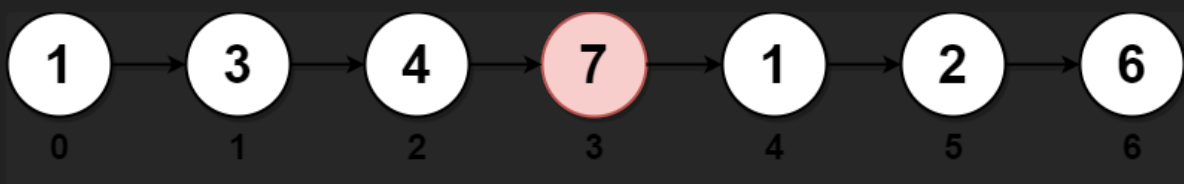
   Input: head = [1,3,4,7,1,2,6]

   Output: [1,3,4,1,2,6]

   Explanation:

   The above figure represents the given linked list. The indices of the nodes are written below.

   Since n = 7, node 3 with value 7 is the middle node, which is marked in red.

   We return the new list after removing this node.

   Example 2:

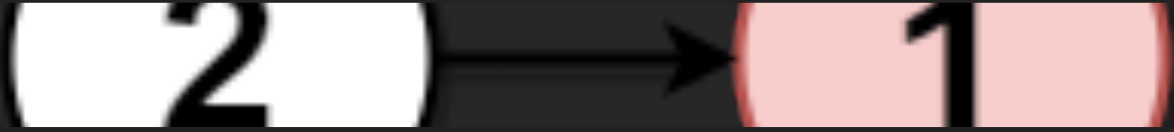   **Input:** head = [1,2,3,4]

   Output: [1,2,4]

   Explanation:

   The above figure represents the given linked list.

   For n = 4, node 2 with value 3 is the middle node, which is marked in red.

Example 3:

Input: head = [2,1]

Output: [2]

Explanation:

The above figure represents the given linked list.

For n = 2, node 1 with value 1 is the middle node, which is marked in red.

Node 0 with value 2 is the only node remaining after removing node 1.

Solution :

```cpp
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if(!head or !head->next)return NULL;

        ListNode *fast = head , *slow = head;

        while(fast and fast->next){
            slow = slow->next;
            fast = fast->next->next;
        }

        ListNode *prev = NULL , *curr = head;

        while(curr != slow){
            prev = curr;
            curr = curr->next;
        }

        prev->next = curr->next;

        return head;
    }
}
```
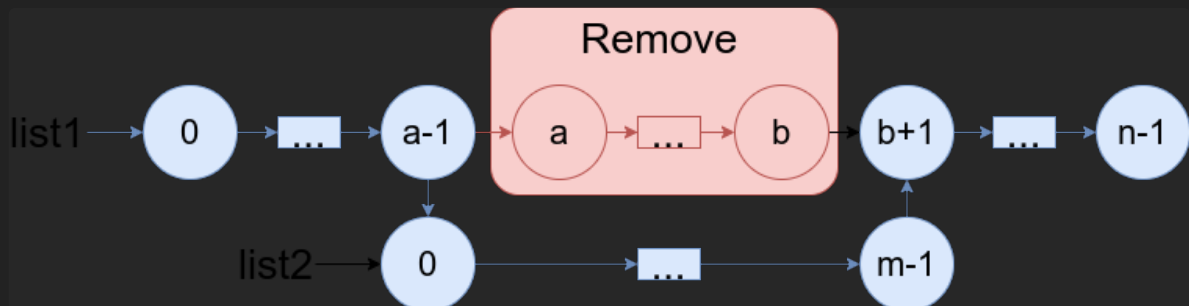
2. You are given two linked lists: `list1` and `list2` of sizes `n` and `m` respectively.
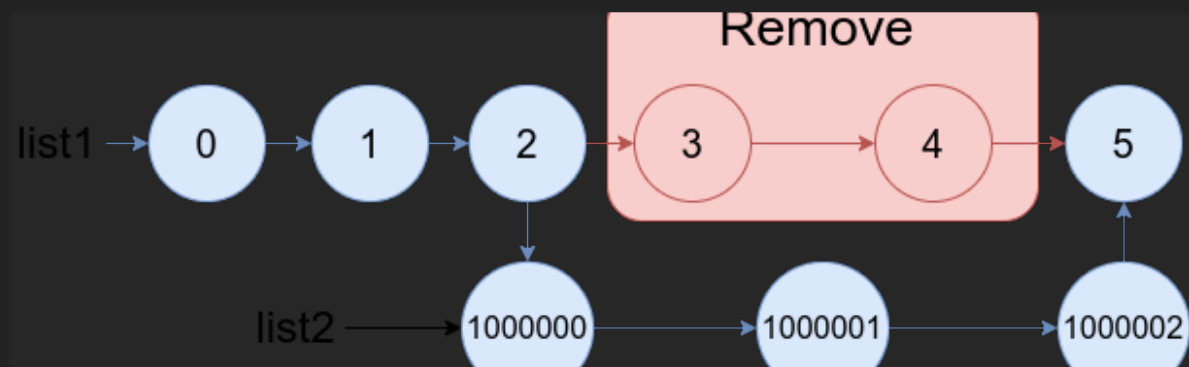   Remove `list1`'s nodes from the `ath` node to the `bth` node, and put `list2` in their place.
   [Leetcode 1669]
   The blue edges and nodes in the following figure indicate the result:

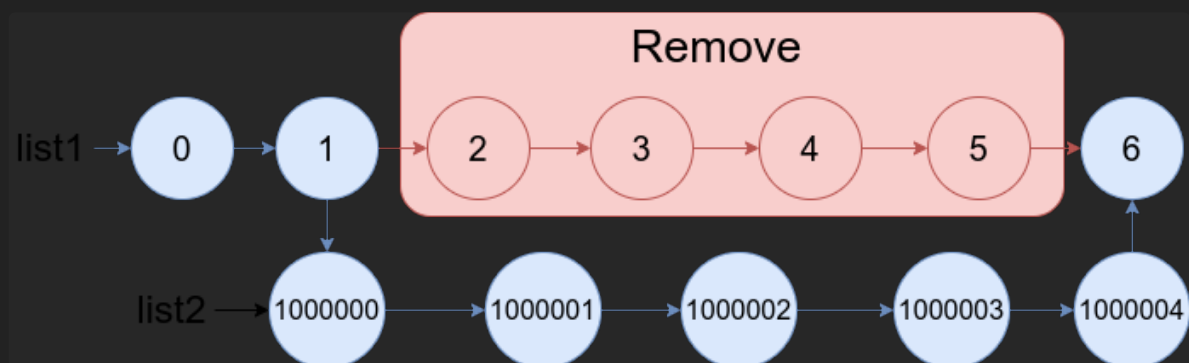*Build the result list and return its head.*

Example 1:

Input: list1 = [0,1,2,3,4,5], a = 3, b = 4, list2 = [1000000,1000001,1000002]

Output: [0,1,2,1000000,1000001,1000002,5]

Explanation: We remove the nodes 3 and 4 and put the entire list2 in their place. The blue edges and nodes in the above figure indicate the result.

Example 2:

Input: list1 = [0,1,2,3,4,5,6], a = 2, b = 5, list2 = [1000000,1000001,1000002,1000003,1000004]

Output: [0,1,1000000,1000001,1000002,1000003,1000004,6]

Explanation: The blue edges and nodes in the above figure indicate the result.

Solution :

```
class Solution {
public:
    ListNode* mergeInBetween(ListNode* list1, int a, int b, ListNode* list2) {
        ListNode *curr = list1;
        a--;
        while(a--){
            curr = curr->next;
        }

        b++;
        ListNode *curr2 = list1;
        while(b--){
            curr2 = curr2->next;
        }

        ListNode *temp = list2;

        while(temp->next)temp = temp->next;
        temp->next = curr2;
        curr->next = list2;

        return list1;
    }
}
```
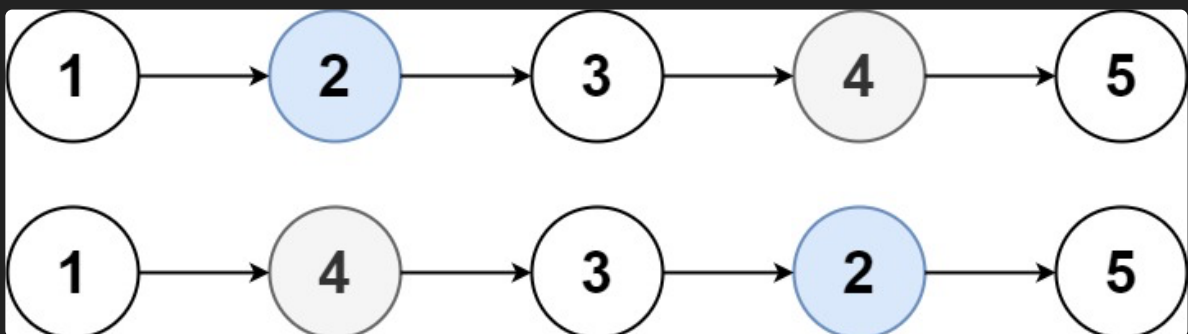
3. You are given the `head` of a linked list, and an integer `k` .

    Return *the head of the linked list after **swapping** the values of the* `kth` *node from the beginning and the* `kth` *node from the end (the list is **1-indexed**).*   **[Leetcode 1721]**

    Example 1:

    Input: head = [1,2,3,4,5], k = 2

    Output: [1,4,3,2,5]

    Example 2:

    Input: head = [7,9,6,6,7,8,3,0,9,5], k = 5

    Output: [7,9,6,6,8,7,3,0,9,5]

    Solution :

```
class Solution {
public:
    ListNode* swapNodes(ListNode* head, int k) {
        ListNode *temp = head;
        k--;
        while(k--)temp = temp->next;
        ListNode *p1 = temp->next , *p2 = head;

        while(p1){
            p1 = p1->next;
            p2 = p2->next;
        }

        swap(temp->val , p2->val);
        return head;
    }
};
```
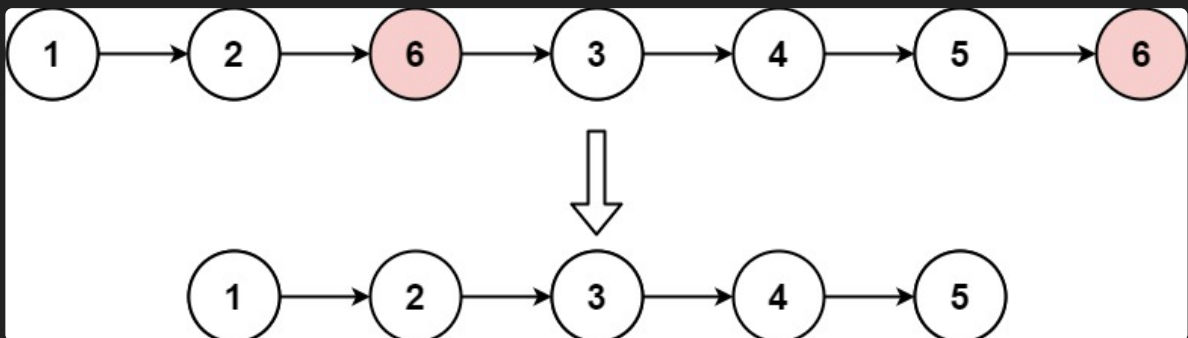
4. Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head*.

Example 1:

Input: head = [1,2,6,3,4,5,6], val = 6

Output: [1,2,3,4,5]

Example 2:

Input: head = [], val = 1

Output: []

Example 3:

Input: head = [7,7,7,7], val = 7

Output: []

Solution :

```cpp
class Solution {
public:
    ListNode* removeElements(ListNode* head, int val) {
        ListNode *curr = head;

        while(curr and curr->val == val){
            curr = curr->next;
        }

        head = curr;

        while(curr){
            if(curr->next and curr->next->val == val)
                curr->next = curr->next->next;
            else curr = curr->next;
        }
        return head;
    }
};
```

5. Find the length of loop in Cycle of Linked List.

Solution :

```cpp
#include<bits/stdc++.h>

using namespace std;

class node{
    public :
        int data;
        node *next;
        node(int n){
            data = n;
            next = NULL;
        }

};

class linkedlist{
    public:
        node *head,*tail;
        linkedlist(){
            head = NULL;
            tail = NULL;
        }
        void display(){
            node *temp = head;
```

```cpp
        while(temp){
        cout<<temp->data<<" ";
        temp = temp->next;
        }
        cout<<endl;
    }
    void addFirst(int val){
        node *temp = new node(val);
        if(head == NULL)head = temp;
        else {
        temp->next = head;
        head = temp;
        }
        if(tail == NULL)tail = head;
        }
        void addCycle(int idx){
        node *temp = head;
        idx--;
        while(idx--){
        temp = temp->next;
        }
        temp->next->next = head->next;
    }
    int findLength(){
        node *fast = head->next;
        node *slow = head;
        int fl = 0;
        while(fast and fast->next){
        if(fast == slow){
        fl = 1;
        break;
        }
        fast = fast->next->next;
        slow = slow->next;
        }
        if(fl == 0)return 0;
        int cnt = 1;
        slow = slow->next;
```

```cpp
            while(slow != fast){
            cnt++;
            slow = slow->next;
            }
            return cnt;
        }
};
int main(){
    linkedlist ll;
    ll.addFirst(1);
    ll.addFirst(2);
    ll.addFirst(3);
    ll.addFirst(4);
    ll.addFirst(5);
    ll.addFirst(6);
    ll.addCycle(4);
    cout<<ll.findLength()<<endl;

}
```