**Merge In Between Linked Lists**

class Solution {

public ListNode mergeInBetween(ListNode list1, int a, int b, ListNode list2) {

int size = 0;

ListNode p1 = list1;

while (size < a - 1) {

p1 = p1.next;

size++;

}

ListNode p2 = p1;

while (size < b + 1) {

p2 = p2.next;

size++;

}

p1.next = list2;

ListNode l2p = list2;

while (l2p.next != null) {

l2p = l2p.next;

}

l2p.next = p2;

return list1;

}

}

**Swapping Nodes in a Linked List**

class Solution {

public ListNode swapNodes(ListNode head, int k) {

if (head == null || head.next == null) return head;

ListNode p1 = head;

ListNode p = head;

int cur = 1;

int size = 1;

while (p.next != null) {

if (size == k) {

p1 = p;

cur = size;

}

size++;

p = p.next;

}

if (size == k) {

p1 = p;

cur = size;

}

int mid = size / 2;

if (cur < mid) {

p = p1;

for (int i = 0; i < size - cur - k + 1; i++) {

p = p.next;

}

} else {

p = head;

for (int i = 0; i < size - cur; i++) {

p = p.next;

}

}

int temp = p1.val;

p1.val = p.val;

p.val = temp;

return head;

}

}

**Remove Linked List Elements**

class Solution {

public ListNode removeElements(ListNode head, int val) {

if (head == null) return head;

ListNode front = head;

while (front != null && front.val == val) {

front = front.next;

head = front;

}

if (front == null) return head;

ListNode back = front.next;

while (back != null) {

if (back.val == val) {

back = back.next;

front.next = back;

continue;

}

front = front.next;

if (front != null) back = front.next;

}

return head;

}

}

**Delete N Nodes After M Nodes of a Linked List**

class Solution {

public ListNode deleteNodes(ListNode head, int m, int n) {

if (head == null) return head;

ListNode p = head;

while (p != null && p.next != null) {

int cur = 1;

while (cur < m) {

if (p == null) return head;

p = p.next;

cur++;

}

if (p == null) return head;

ListNode p2 = p.next;

cur = 0;

while (cur < n) {

if (p2 == null) {

p.next = p2;

return head;

}

cur++;

p2 = p2.next;

}

p.next = p2;

p = p.next;

}

return head;

}

}

**Design and implement a Double Link list**

public class Main {

public static void main(String[] args) {

ListNode ln = new ListNode(2);

doubleLinkedList obj = new doubleLinkedList(ln);

System.out.println("Size: " + obj.getSize());

obj.addAtHead(3);

obj.addAtTail(4);

System.out.println("Size: " + obj.getSize());

obj.printList();

}

}

class ListNode {

int val;

ListNode next;

ListNode prev;

public ListNode() {}

public ListNode(int val) { this.val = val; }

public ListNode(int val, ListNode next) { this.val = val; this.next = next; }

public ListNode(int val, ListNode next, ListNode prev) { this.val = val; this.next = next; this.prev = prev; }

}

class doubleLinkedList {

private ListNode head;

private ListNode tail;

private int size;

public doubleLinkedList() {}

public doubleLinkedList(ListNode head) {

this.head = head;

this.tail = head;

size += 1;

}

public int getSize() {

return size;

}

public void printList() {

ListNode p = head;

for (int i = 0; i < size; i++) {

System.out.print(p.val + ", ");

p = p.next;

}

}

/\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/

public void addAtHead(int val) {

size++;

ListNode newNode = new ListNode(val);

newNode.next = head;

newNode.next.prev = newNode;

head = newNode;

newNode.prev = head;

}

/\*\* Append a node of value val to the last element of the linked list. \*/

public void addAtTail(int val) {

size++;

ListNode newNode = new ListNode(val);

tail.next = newNode;

newNode.prev = tail;

}

}