**MARKING RUBRIC**

| **Criteria** | **Description** | **Marks** |
| --- | --- | --- |
| | **1. Function Initialization** | | --- |  |  | | --- | | Correct function signature, handling base cases (e.g., empty list, k=0). | **2** |
| | **2. Length and Tail** | | --- | | Correctly calculate the list length and identify the tail node. | **3** |
| **3. Normalizing k** | Correct calculation of k%length, handling k = 0 | **2** |
| | **4. New Tail and Head** | | | --- | --- | |  | | Accurate identification of the new tail and head positions | **3** |
| | **5. Breaking and Reconnecting** | | | --- | --- | |  | | Correctly break the lines at the new tail and head positions | **3** |
| | **6. Output and Readability** | | | --- | --- | |  | | Produces correct output, clean code | **2** |

**PYTHON**

**Solution of set-A:**

def rotate\_doubly\_linked\_list(head, k):

    if not head or not head.next or k == 0:

        return head

    length = 1

    tail = head

    while tail.next:

        tail = tail.next

        length += 1

    k = k % length

    if k == 0:

        return head

    new\_tail\_pos = length - k - 1

    new\_head\_pos = length - k

    new\_tail = head

    for \_ in range(new\_tail\_pos):

        new\_tail = new\_tail.next

    new\_head = new\_tail.next

    new\_tail.next = None

    new\_head.prev = None

    tail.next = head

    head.prev = tail

    return new\_head

**Solution of set-B:**

def rotate\_doubly\_linked\_list(head, k):

    if not head or not head.next or k == 0:

        return head

    length = 1

    tail = head

    while tail.next:

        tail = tail.next

        length += 1

    k = k % length

    if k == 0:

        return head

    new\_tail\_pos = k - 1

    new\_tail = head

    for \_ in range(new\_tail\_pos):

        new\_tail = new\_tail.next

    new\_head = new\_tail.next

    new\_tail.next = None

    new\_head.prev = None

    tail.next = head

    head.prev = tail

    return new\_head

**JAVA**

**Set-A [There can be other valid solution as well]**

**public static Node rotateDoublyLinkedList(Node head, int k) {**

**if (head == null || head.next == null || k == 0) {**

**return head;**

**}**

**int length = 1;**

**Node tail = head;**

**while (tail.next != null) {**

**tail = tail.next;**

**length++;**

**}**

**k = k % length;**

**if (k == 0) {**

**return head;**

**}**

**int newTailPos = length - k - 1;**

**int newHeadPos = length - k;**

**Node newTail = head;**

**for (int i = 0; i < newTailPos; i++) {**

**newTail = newTail.next;**

**}**

**Node newHead = newTail.next;**

**newTail.next = null;**

**newHead.prev = null;**

**tail.next = head;**

**head.prev = tail;**

**return newHead;**

**}**

**Set-B [There can be other valid solution as well]**

**public static Node rotateDoublyLinkedList(ListNode head, int k) {**

**if (head == null || head.next == null || k == 0) {**

**return head;**

**}**

**int length = 1;**

**Node tail = head;**

**while (tail.next != null) {**

**tail = tail.next;**

**length++;**

**}**

**k = k % length;**

**if (k == 0) {**

**return head;**

**}**

**int newTailPos = k - 1;**

**Node newTail = head;**

**for (int i = 0; i < newTailPos; i++) {**

**newTail = newTail.next;**

**}**

**Node newHead = newTail.next;**

**newTail.next = null;**

**if (newHead != null) {**

**newHead.prev = null;**

**}**

**tail.next = head;**

**if (head != null) {**

**head.prev = tail;**

**}**

**return newHead;**

**}**