# DSA

September 7, 2024

Problem 1: Reverse a singly linked list.

Problem 2: Merge two sorted linked lists into one sorted linked list.

Problem 3: Remove the nth node from the end of a linked list.

Problem 4: Find the intersection point of two linked lists.

Problem 5: Remove duplicates from a sorted linked list.

Problem 6: Add two numbers represented by linked lists (where each node contains a single digit).

Problem 7: Swap nodes in pairs in a linked list.

Problem 8: Reverse nodes in a linked list in groups of k.

Problem 9: Determine if a linked list is a palindrome. Input: 1 -> 2 -> 3 -> 4 -> 5 Output: 5 -> 4 -> 3 -> 2 -> 1

Input: List 1: 1 -> 3 -> 5, List 2: 2 -> 4 -> 6 Output: 1 -> 2 -> 3 -> 4 -> 5 -> 6

Input: 1 -> 2 -> 3 -> 4 -> 5, n = 2 Output: 1 -> 2 -> 3 -> 5

Input: List 1: 1 -> 2 -> 3 -> 4, List 2: 9 -> 8 -> 3 -> 4 Output: Node with value 3

Input: 1 -> 1 -> 2 -> 3 -> 3 Output: 1 -> 2 -> 3

Input: List 1: 2 -> 4 -> 3, List 2: 5 -> 6 -> 4 (represents 342 + 465) Output: 7 -> 0 -> 8 (represents 807)

Input: 1 -> 2 -> 3 -> 4 Output: 2 -> 1 -> 4 -> 3

Input: 1 -> 2 -> 3 -> 4 -> 5, k = 3 Output: 3 -> 2 -> 1 -> 4 -> 5

Input: 1 -> 2 -> 2 -> 1 Output: True

[1]:

```python
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def reverse_linked_list(head):
    prev = None
    current = head
    while current:
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev

# Example usage:
# 1 -> 2 -> 3 -> 4 -> 5
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

reversed_head = reverse_linked_list(head)
while reversed_head:
    print(reversed_head.value, end=" -> " if reversed_head.next else "")
    reversed_head = reversed_head.next
```

```
5 -> 4 -> 3 -> 2 -> 1
```

[2]:

```
def merge_two_sorted_lists(l1, l2):
    dummy = ListNode()
    current = dummy
    while l1 and l2:
        if l1.value < l2.value:
            current.next = l1
            l1 = l1.next
        else:
            current.next = l2
            l2 = l2.next
        current = current.next
    if l1:
        current.next = l1
    if l2:
        current.next = l2
    return dummy.next

# Example usage:
# List 1: 1 -> 3 -> 5
# List 2: 2 -> 4 -> 6
l1 = ListNode(1, ListNode(3, ListNode(5)))
l2 = ListNode(2, ListNode(4, ListNode(6)))

merged_head = merge_two_sorted_lists(l1, l2)
while merged_head:
    print(merged_head.value, end=" -> " if merged_head.next else "")
    merged_head = merged_head.next
```

```
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

[3]:

```python
def remove_nth_from_end(head, n):
    dummy = ListNode(0, head)
    first = second = dummy
    for _ in range(n + 1):
        first = first.next
    while first:
        first = first.next
        second = second.next
    second.next = second.next.next
    return dummy.next

# Example usage:
# 1 -> 2 -> 3 -> 4 -> 5, n = 2
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

new_head = remove_nth_from_end(head, 2)
while new_head:
    print(new_head.value, end=" -> " if new_head.next else "")
    new_head = new_head.next
```

```
1 -> 2 -> 3 -> 5
```

[4]:

```python
def get_intersection_node(headA, headB):
    if not headA or not headB:
        return None
    a, b = headA, headB
    while a != b:
        a = a.next if a else headB
        b = b.next if b else headA
    return a

# Example usage:
# List 1: 1 -> 2 -> 3 -> 4
# List 2: 9 -> 8 -> 3 -> 4
common = ListNode(3, ListNode(4))
list1 = ListNode(1, ListNode(2, common))
list2 = ListNode(9, ListNode(8, common))

intersection = get_intersection_node(list1, list2)
print(intersection.value if intersection else "No intersection")
```

```
3
```

[5]:

```python
def remove_duplicates(head):
    current = head
    while current and current.next:
        if current.value == current.next.value:
            current.next = current.next.next
        else:
            current = current.next
    return head

# Example usage:
# 1 -> 1 -> 2 -> 3 -> 3
head = ListNode(1, ListNode(1, ListNode(2, ListNode(3, ListNode(3)))))

new_head = remove_duplicates(head)
while new_head:
    print(new_head.value, end=" -> " if new_head.next else "")
    new_head = new_head.next
```

```
1 -> 2 -> 3
```

[6]:

```
def add_two_numbers(l1, l2):
    dummy = ListNode()
    p, q, current = l1, l2, dummy
    carry = 0
    while p or q or carry:
        x = p.value if p else 0
        y = q.value if q else 0
        total = x + y + carry
        carry = total // 10
        current.next = ListNode(total % 10)
        current = current.next
        if p: p = p.next
        if q: q = q.next
    return dummy.next

# Example usage:
# List 1: 2 -> 4 -> 3 (represents 342)
# List 2: 5 -> 6 -> 4 (represents 465)
l1 = ListNode(2, ListNode(4, ListNode(3)))
l2 = ListNode(5, ListNode(6, ListNode(4)))

sum_head = add_two_numbers(l1, l2)
while sum_head:
    print(sum_head.value, end=" -> " if sum_head.next else "")
    sum_head = sum_head.next
```

```
7 -> 0 -> 8
```

[7]:

```
def swap_pairs(head):
    dummy = ListNode(0)
    dummy.next = head
```

```python
    prev = dummy
    while head and head.next:
        first = head
        second = head.next
        prev.next = second
        first.next = second.next
        second.next = first
        prev = first
        head = first.next
    return dummy.next

# Example usage:
# 1 -> 2 -> 3 -> 4
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4))))

swapped_head = swap_pairs(head)
while swapped_head:
    print(swapped_head.value, end=" -> " if swapped_head.next else "")
    swapped_head = swapped_head.next
```

```
2 -> 1 -> 4 -> 3
```

[8]:

```python
def reverse_k_group(head, k):
    def reverse_linked_list(head, k):
        prev, curr = None, head
        while k:
            next_node = curr.next
            curr.next = prev
            prev = curr
            curr = next_node
            k -= 1
        return prev

    def get_kth_node(head, k):
        curr = head
        while k and curr:
            curr = curr.next
            k -= 1
        return curr

    dummy = ListNode(0)
    dummy.next = head
    prev_group_end = dummy
    while True:
        kth_node = get_kth_node(head, k)
        if not kth_node:
            break
        group_start = head
        next_group_start = kth_node.next
        kth_node.next = None
        prev_group_end.next = reverse_linked_list(group_start, k)
        group_start.next = next_group_start
        prev_group_end = group_start
        head = next_group_start
    return dummy.next

# Example usage:
# 1 -> 2 -> 3 -> 4 -> 5, k = 3
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

reversed_head = reverse_k_group(head, 3)
while reversed_head:
    print(reversed_head.value, end=" -> " if reversed_head.next else "")
    reversed_head = reversed_head.next
```

```
3 -> 2 -> 1 -> 5
```

[9]:

```
def is_palindrome(head):
    def find_middle(head):
        slow = fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        return slow

    def reverse_list(head):
        prev = None
        while head:
            next_node = head.next
            head.next = prev
            prev = head
            head = next_node
        return prev

    middle = find_middle(head)
    second_half = reverse_list(middle)
    first_half = head
    while second_half:
        if first_half.value != second_half.value:
            return False
        first_half = first_half.next
        second_half = second_half.next
    return True
```

```
# Example usage:
# 1 -> 2 -> 2 -> 1
head = ListNode(1, ListNode(2, ListNode(2, ListNode(1))))

print(is_palindrome(head))
```

```
True
```

Problem 10: Rotate a linked list to the right by k places.

Problem 11: Flatten a multilevel doubly linked list.

Problem 12: Rearrange a linked list such that all even positioned nodes are placed at the end.

Problem 13: Given a non-negative number represented as a linked list, add one to it.

Problem 14: Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be inserted.

Problem 15: Find the minimum element in a rotated sorted array.

Problem 16: Search for a target value in a rotated sorted array.

Problem 17: Find the peak element in an array. A peak element is greater than its neighbors.

Problem 18: Given a m x n matrix where each row and column is sorted in ascending order, count the number of negative numbers. Input: 1 -> 2 -> 3 -> 4 -> 5, k = 2 Output: 4 -> 5 -> 1 -> 2 -> 3

Input: 1 <-> 2 <-> 3 <-> 7 <-> 8 <-> 11 -> 12, 4 <-> 5 -> 9 -> 10, 6 -> 13 Output: 1 <-> 2 <-> 3 <-> 4 <-> 5 <-> 6 <-> 7 <-> 8 <-> 9 <-> 10 <-> 11 <-> 12 <-> 13

Input: 1 -> 2 -> 3 -> 4 -> 5 Output: 1 -> 3 -> 5 -> 2 -> 4

Input: 1 -> 2 -> 3 (represents the number 123) Output: 1 -> 2 -> 4 (represents the number 124)

Input: nums = [1, 3, 5, 6], target = 5 Output: 2

Input: [4, 5, 6, 7, 0, 1, 2] Output: 0

Input: nums = [4, 5, 6, 7, 0, 1, 2], target = 0 Output: 4

Input: nums = [1, 2, 3, 1] Output: 2 (index of peak element)

Input: grid = [[4, 3, 2, -1], [3, 2, 1, -1], [1, 1, -1, -2], [-1, -1, -2, -3]] Output: 8

[10]:

```
def rotate_right(head, k):
    if not head or k == 0:
        return head

    # Compute the length of the linked list
    length = 1
    old_tail = head
    while old_tail.next:
        old_tail = old_tail.next
        length += 1
    # Make it a circular linked list
    old_tail.next = head

    # Find the new tail and new head
    k = k % length
    steps_to_new_head = length - k
    new_tail = old_tail
    for _ in range(steps_to_new_head):
        new_tail = new_tail.next

    new_head = new_tail.next
    new_tail.next = None
    return new_head

# Example usage:
# 1 -> 2 -> 3 -> 4 -> 5, k = 2
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

rotated_head = rotate_right(head, 2)
while rotated_head:
    print(rotated_head.value, end=" -> " if rotated_head.next else "")
    rotated_head = rotated_head.next
```

```
4 -> 5 -> 1 -> 2 -> 3
```

[11]:

```
class Node:
    def __init__(self, value=0, next=None, prev=None, child=None):
        self.value = value
        self.next = next
        self.prev = prev
        self.child = child

def flatten(head):
    def flatten_dfs(node):
        current = node
        tail = node
        while current:
            if current.child:
                next_node = current.next
                child_tail = flatten_dfs(current.child)

                current.next = current.child
                current.child.prev = current
                current.child = None

                if next_node:
                    child_tail.next = next_node
                    next_node.prev = child_tail

                tail = child_tail
            else:
                tail = current
            current = current.next
        return tail

    flatten_dfs(head)
    return head

# Example usage:
# Creating the multilevel doubly linked list
head = Node(1, Node(2, Node(3)))
head.next.child = Node(7, Node(8, Node(9, Node(10))))
head.next.child.child = Node(11, Node(12))

flattened_head = flatten(head)
while flattened_head:
    print(flattened_head.value, end=" -> " if flattened_head.next else "")
    flattened_head = flattened_head.next
```

```
1 -> 2 -> 7 -> 11 -> 12 -> 8 -> 9 -> 10 -> 3
```

[12]:

```
def rearrange_even_odd(head):
    if not head:
        return None

    odd_head = odd = ListNode(0)
    even_head = even = ListNode(0)
    index = 1

    while head:
        if index % 2 == 1:
            odd.next = head
            odd = odd.next
        else:
            even.next = head
            even = even.next
        head = head.next
        index += 1

    even.next = None
    odd.next = even_head.next
    return odd_head.next
```

```
# Example usage:
# 1 -> 2 -> 3 -> 4 -> 5
head = ListNode(1, ListNode(2, ListNode(3, ListNode(4, ListNode(5)))))

rearranged_head = rearrange_even_odd(head)
while rearranged_head:
    print(rearranged_head.value, end=" -> " if rearranged_head.next else "")
    rearranged_head = rearranged_head.next
```

```
1 -> 3 -> 5 -> 2 -> 4
```

[13]:

```
def add_one(head):
    def reverse_list(node):
        prev = None
        while node:
            next_node = node.next
            node.next = prev
            prev = node
            node = next_node
        return prev

    def add_one_to_list(node):
        carry = 1
        prev = None
        while node:
            total = node.value + carry
            carry = total // 10
            node.value = total % 10
            prev = node
            node = node.next
        if carry:
            prev.next = ListNode(carry)

    head = reverse_list(head)
    add_one_to_list(head)
    return reverse_list(head)

# Example usage:
# 1 -> 2 -> 3 (represents the number 123)
head = ListNode(1, ListNode(2, ListNode(3)))

new_head = add_one(head)
while new_head:
    print(new_head.value, end=" -> " if new_head.next else "")
    new_head = new_head.next
```

```
1 -> 2 -> 4
```

[14]:

```python
def search_insert_position(nums, target):
    left, right = 0, len(nums)
    while left < right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid
    return left

# Example usage:
# nums = [1, 3, 5, 6], target = 5
print(search_insert_position([1, 3, 5, 6], 5))  # Output: 2
```

```
2
```

[15]:

```python
def find_min(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > nums[right]:
            left = mid + 1
        else:
            right = mid
    return nums[left]

# Example usage:
# nums = [4, 5, 6, 7, 0, 1, 2]
print(find_min([4, 5, 6, 7, 0, 1, 2]))  # Output: 0
```

```
0
```

[16]:

```python
def search_in_rotated_sorted_array(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1
    return -1

# Example usage:
# nums = [4, 5, 6, 7, 0, 1, 2], target = 0
print(search_in_rotated_sorted_array([4, 5, 6, 7, 0, 1, 2], 0))  # Output: 4
```

```
4
```

[17]:

```
def find_peak_element(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > nums[mid + 1]:
            right = mid
        else:
            left = mid + 1
    return left

# Example usage:
# nums = [1, 2, 3, 1]
print(find_peak_element([1, 2, 3, 1]))  # Output: 2 (index of peak element)
```

```
2
```

[18]:

```
def count_negatives(grid):
    count = 0
    rows, cols = len(grid), len(grid[0])
    r, c = rows - 1, 0

    while r >= 0 and c < cols:
        if grid[r][c] < 0:
            count += (cols - c)
            r -= 1
        else:
            c += 1

    return count

# Example usage:
# grid = [[4, 3, 2, -1], [3, 2, 1, -1], [1, 1, -1, -2], [-1, -1, -2, -3]]
print(count_negatives([[4, 3, 2, -1], [3, 2, 1, -1], [1, 1, -1, -2], [-1, -1,␣
 ↪-2, -3]])) # Output: 8
```

```
8
```

```
[ ]:
```