

```

1 //Easy Problems
2
3 // Node class for singly linked list
4 class ListNode {
5     int data;
6     ListNode next;
7     ListNode(int data) { this.data = data; }
8 }
9
10 // Node class for doubly linked list
11 class DoublyListNode {
12     int data;
13     DoublyListNode next, prev;
14     DoublyListNode(int data) { this.data = data; }
15 }
16
17 public class LinkedListProblems {
18
19     // 1. Middle of a linked list
20     public ListNode findMiddle(ListNode head) {
21         if (head == null) return null;
22         ListNode slow = head, fast = head;
23         while (fast != null && fast.next != null) {
24             slow = slow.next;
25             fast = fast.next.next;
26         }
27         return slow; // middle node
28     }
29
30     // 2. Reverse a singly linked list
31     public ListNode reverseList(ListNode head) {
32         ListNode prev = null, curr = head;
33         while (curr != null) {
34             ListNode nextNode = curr.next;
35             curr.next = prev;
36             prev = curr;
37             curr = nextNode;
38         }
39         return prev;
40     }
41
42     // 3. Reverse a doubly linked list
43     public DoublyListNode reverseDoublyList(DoublyListNode head) {
44         DoublyListNode curr = head, prev = null;
45         while (curr != null) {
46             DoublyListNode nextNode = curr.next;
47             curr.next = prev;
48             curr.prev = nextNode;
49             prev = curr;
50             curr = nextNode;
51         }
52         return prev;
53     }
54
55     // 4. Rotate a linked list by k nodes
56     public ListNode rotateList(ListNode head, int k) {
57         if (head == null || k == 0) return head;
58         ListNode curr = head;
59         int len = 1;
60         while (curr.next != null) {
61             curr = curr.next;
62             len++;
63         }
64         k = k % len;
65         if (k == 0) return head;
66
67         curr.next = head; // make circular
68         ListNode newTail = head;
69         for (int i = 0; i < len - k - 1; i++) newTail = newTail.next;

```

```

70     ListNode newHead = newTail.next;
71     newTail.next = null;
72     return newHead;
73 }
74
75 // 5. Nth node from End
76 public ListNode nthFromEnd(ListNode head, int n) {
77     ListNode first = head, second = head;
78     for (int i = 0; i < n; i++) {
79         if (first == null) return null;
80         first = first.next;
81     }
82     while (first != null) {
83         first = first.next;
84         second = second.next;
85     }
86     return second;
87 }
88
89 // 6. Delete Last Occurrence of key
90 public ListNode deleteLastOccurrence(ListNode head, int key) {
91     ListNode curr = head, last = null;
92     while (curr != null) {
93         if (curr.data == key) last = curr;
94         curr = curr.next;
95     }
96     if (last == null) return head; // key not found
97     if (last == head && head.next == null) return null;
98     curr = head;
99     ListNode prev = null;
100    while (curr != last) {
101        prev = curr;
102        curr = curr.next;
103    }
104    if (prev != null) prev.next = curr.next;
105    else head = curr.next; // last is head
106    return head;
107 }
108
109 // 7. Delete middle node (given head)
110 public ListNode deleteMiddle(ListNode head) {
111     if (head == null || head.next == null) return null;
112     ListNode slow = head, fast = head, prev = null;
113     while (fast != null && fast.next != null) {
114         prev = slow;
115         slow = slow.next;
116         fast = fast.next.next;
117     }
118     prev.next = slow.next;
119     return head;
120 }
121
122 // 8. Merge alternate positions of two lists
123 public ListNode mergeAlternate(ListNode first, ListNode second) {
124     ListNode head1 = first, head2 = second;
125     while (head1 != null && head2 != null) {
126         ListNode next1 = head1.next, next2 = head2.next;
127         head1.next = head2;
128         if (next1 == null) break;
129         head2.next = next1;
130         head1 = next1;
131         head2 = next2;
132     }
133     return first;
134 }
135
136 // 9. Circular list traversal
137 public void traverseCircularList(ListNode head) {
138     if (head == null) return;

```

```

139     ListNode curr = head;
140     do {
141         System.out.print(curr.data + " ");
142         curr = curr.next;
143     } while (curr != head);
144     System.out.println();
145 }
146
147 // 10. Queue using linked list
148 class QueueLL {
149     ListNode front, rear;
150
151     public void enqueue(int x) {
152         ListNode node = new ListNode(x);
153         if (rear != null) rear.next = node;
154         rear = node;
155         if (front == null) front = rear;
156     }
157
158     public int dequeue() {
159         if (front == null) throw new RuntimeException("Queue empty");
160         int val = front.data;
161         front = front.next;
162         if (front == null) rear = null;
163         return val;
164     }
165
166     public boolean isEmpty() { return front == null; }
167 }
168
169 // 11. Stack using singly linked list
170 class StackLL {
171     ListNode top;
172     public void push(int x) {
173         ListNode node = new ListNode(x);
174         node.next = top;
175         top = node;
176     }
177     public int pop() {
178         if (top == null) throw new RuntimeException("Stack empty");
179         int val = top.data;
180         top = top.next;
181         return val;
182     }
183     public boolean isEmpty() { return top == null; }
184 }
185
186 // 12. Pairwise swap nodes
187 public ListNode pairwiseSwap(ListNode head) {
188     if (head == null || head.next == null) return head;
189     ListNode prev = head, curr = head.next;
190     head = curr; // new head
191     while (true) {
192         ListNode next = curr.next;
193         curr.next = prev;
194         if (next == null || next.next == null) {
195             prev.next = next;
196             break;
197         }
198         prev.next = next.next;
199         prev = next;
200         curr = prev.next;
201     }
202     return head;
203 }
204
205 // 13. Count occurrences of a key
206 public int countOccurrences(ListNode head, int key) {
207     int count = 0;

```

```

208     while (head != null) {
209         if (head.data == key) count++;
210         head = head.next;
211     }
212     return count;
213 }
214 }
215
216
217 //Medium Problems
218
219 // Node class for singly linked list
220 class ListNode {
221     int data;
222     ListNode next;
223     ListNode(int data) { this.data = data; }
224 }
225
226 // Node class for doubly linked list
227 class DoublyListNode {
228     int data;
229     DoublyListNode next, prev;
230     DoublyListNode(int data) { this.data = data; }
231 }
232
233 public class LinkedListMediumProblems {
234
235     // 1. Detect Loop in Linked List
236     public boolean detectLoop(ListNode head) {
237         ListNode slow = head, fast = head;
238         while (fast != null && fast.next != null) {
239             slow = slow.next;
240             fast = fast.next.next;
241             if (slow == fast) return true;
242         }
243         return false;
244     }
245
246     // 2. Length of the Loop
247     public int loopLength(ListNode head) {
248         ListNode slow = head, fast = head;
249         while (fast != null && fast.next != null) {
250             slow = slow.next;
251             fast = fast.next.next;
252             if (slow == fast) {
253                 int length = 1;
254                 ListNode temp = slow;
255                 while (temp.next != slow) {
256                     length++;
257                     temp = temp.next;
258                 }
259                 return length;
260             }
261         }
262         return 0;
263     }
264
265     // 3. Reverse in groups of k
266     public ListNode reverseInGroups(ListNode head, int k) {
267         if (head == null) return null;
268         ListNode curr = head, prev = null, next = null;
269         int count = 0;
270         while (curr != null && count < k) {
271             next = curr.next;
272             curr.next = prev;
273             prev = curr;
274             curr = next;
275             count++;
276         }

```

```

277     if (next != null) head.next = reverseInGroups(next, k);
278     return prev;
279 }
280
281 // 4. Intersection Point of two linked lists
282 public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
283     if (headA == null || headB == null) return null;
284     ListNode a = headA, b = headB;
285     while (a != b) {
286         a = (a == null) ? headB : a.next;
287         b = (b == null) ? headA : b.next;
288     }
289     return a;
290 }
291
292 // 5. Delete a node without head pointer
293 public void deleteWithoutHead(ListNode node) {
294     if (node == null || node.next == null) return;
295     node.data = node.next.data;
296     node.next = node.next.next;
297 }
298
299 // 6. Merge two sorted linked lists
300 public ListNode mergeSorted(ListNode l1, ListNode l2) {
301     ListNode dummy = new ListNode(0), tail = dummy;
302     while (l1 != null && l2 != null) {
303         if (l1.data < l2.data) {
304             tail.next = l1; l1 = l1.next;
305         } else {
306             tail.next = l2; l2 = l2.next;
307         }
308         tail = tail.next;
309     }
310     tail.next = (l1 != null) ? l1 : l2;
311     return dummy.next;
312 }
313
314 // 7. Sort a List of 0s, 1s, and 2s
315 public ListNode sort012(ListNode head) {
316     int[] count = new int[3];
317     ListNode curr = head;
318     while (curr != null) {
319         count[curr.data]++;
320         curr = curr.next;
321     }
322     curr = head;
323     for (int i = 0; i < 3; i++) {
324         while (count[i]-- > 0) {
325             curr.data = i;
326             curr = curr.next;
327         }
328     }
329     return head;
330 }
331
332 // 8. Check Palindrome Linked List
333 public boolean isPalindrome(ListNode head) {
334     if (head == null) return true;
335     ListNode slow = head, fast = head;
336     while (fast != null && fast.next != null) {
337         slow = slow.next;
338         fast = fast.next.next;
339     }
340     ListNode secondHalf = reverseList(slow);
341     ListNode firstHalf = head;
342     while (secondHalf != null) {
343         if (firstHalf.data != secondHalf.data) return false;
344         firstHalf = firstHalf.next;
345         secondHalf = secondHalf.next;

```

```

346     }
347     return true;
348 }
349
350 private ListNode reverseList(ListNode head) {
351     ListNode prev = null, curr = head;
352     while (curr != null) {
353         ListNode nextNode = curr.next;
354         curr.next = prev;
355         prev = curr;
356         curr = nextNode;
357     }
358     return prev;
359 }
360
361 // 9. Remove all occurrences of a given value
362 public ListNode removeAll(ListNode head, int key) {
363     ListNode dummy = new ListNode(0);
364     dummy.next = head;
365     ListNode prev = dummy, curr = head;
366     while (curr != null) {
367         if (curr.data == key) prev.next = curr.next;
368         else prev = curr;
369         curr = curr.next;
370     }
371     return dummy.next;
372 }
373
374 // 10. Split a Circular Linked List into two halves
375 public ListNode[] splitCircularList(ListNode head) {
376     if (head == null || head.next == head) return new ListNode[]{head, null};
377     ListNode slow = head, fast = head;
378     while (fast.next != head && fast.next.next != head) {
379         slow = slow.next;
380         fast = fast.next.next;
381     }
382     ListNode head1 = head;
383     ListNode head2 = slow.next;
384     slow.next = head1;
385     ListNode temp = head2;
386     while (temp.next != head) temp = temp.next;
387     temp.next = head2;
388     return new ListNode[]{head1, head2};
389 }
390
391 // 11. Pair Sum in Doubly Linked List
392 public void pairSum(DoublyListNode head, int sum) {
393     DoublyListNode first = head;
394     DoublyListNode second = head;
395     while (second.next != null) second = second.next;
396     while (first != null && second != null && first != second && second.next != first) {
397         int s = first.data + second.data;
398         if (s == sum) {
399             System.out.println("(" + first.data + "," + second.data + ")");
400             first = first.next; second = second.prev;
401         } else if (s < sum) first = first.next;
402         else second = second.prev;
403     }
404 }
405
406 // 12. Add two numbers represented by linked lists
407 public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
408     ListNode dummy = new ListNode(0), curr = dummy;
409     int carry = 0;
410     while (l1 != null || l2 != null || carry != 0) {
411         int sum = carry;
412         if (l1 != null) { sum += l1.data; l1 = l1.next; }
413         if (l2 != null) { sum += l2.data; l2 = l2.next; }

```

```

414     carry = sum / 10;
415     curr.next = new ListNode(sum % 10);
416     curr = curr.next;
417 }
418 return dummy.next;
419 }
420
421 // 13. Multiply two numbers represented by linked lists
422 public ListNode multiplyTwoNumbers(ListNode l1, ListNode l2) {
423     int num1 = 0, num2 = 0;
424     ListNode temp = l1;
425     while (temp != null) { num1 = num1 * 10 + temp.data; temp = temp.next; }
426     temp = l2;
427     while (temp != null) { num2 = num2 * 10 + temp.data; temp = temp.next; }
428     int product = num1 * num2;
429     if (product == 0) return new ListNode(0);
430     ListNode dummy = new ListNode(0), curr = null;
431     while (product > 0) {
432         ListNode node = new ListNode(product % 10);
433         node.next = curr;
434         curr = node;
435         product /= 10;
436     }
437     return curr;
438 }
439
440 // 14. Swap Kth nodes from beginning and end
441 public ListNode swapKthNode(ListNode head, int k) {
442     int len = 0;
443     ListNode curr = head;
444     while (curr != null) { len++; curr = curr.next; }
445     if (k > len) return head;
446     if (2*k - 1 == len) return head; // same node
447     ListNode prevX = null, currX = head;
448     for (int i = 1; i < k; i++) { prevX = currX; currX = currX.next; }
449     ListNode prevY = null, currY = head;
450     for (int i = 1; i < len - k + 1; i++) { prevY = currY; currY = currY.next; }
451     if (prevX != null) prevX.next = currY; else head = currY;
452     if (prevY != null) prevY.next = currX; else head = currX;
453     ListNode temp = currX.next;
454     currX.next = currY.next;
455     currY.next = temp;
456     return head;
457 }
458
459 // 15. Rotate Doubly Linked List by N nodes
460 public DoublyListNode rotateDoublyList(DoublyListNode head, int n) {
461     if (head == null || n == 0) return head;
462     DoublyListNode tail = head;
463     int len = 1;
464     while (tail.next != null) { tail = tail.next; len++; }
465     n = n % len;
466     if (n == 0) return head;
467     DoublyListNode newTail = head;
468     for (int i = 1; i < n; i++) newTail = newTail.next;
469     DoublyListNode newHead = newTail.next;
470     newTail.next = null;
471     newHead.prev = null;
472     tail.next = head;
473     head.prev = tail;
474     return newHead;
475 }
476
477 // 16. Binary Tree to Doubly Linked List
478 class TreeNode {
479     int val;
480     TreeNode left, right;
481     TreeNode(int val) { this.val = val; }
482 }

```

```
483 DoublyListNode headDLL = null, prevDLL = null;
484 public void btreeToDLL(TreeNode root) {
485     if (root == null) return;
486     btreeToDLL(root.left);
487     DoublyListNode node = new DoublyListNode(root.val);
488     if (prevDLL == null) headDLL = node;
489     else { prevDLL.next = node; node.prev = prevDLL; }
490     prevDLL = node;
491     btreeToDLL(root.right);
492 }
493
494 // 17. Linked List from 2D matrix
495 public ListNode linkedListFromMatrix(int[][] mat) {
496     ListNode dummy = new ListNode(0), tail = dummy;
497     for (int[] row : mat) {
498         for (int val : row) {
499             tail.next = new ListNode(val);
500             tail = tail.next;
501         }
502     }
503     return dummy.next;
504 }
505
506 // 18. Reverse a Sublist (positions m to n)
507 public ListNode reverseSublist(ListNode head, int m, int n) {
```