```
package algorithm.list;

/**
 * @author xiaobaoqiu  Date: 16-7-5 Time:  下午11:23
 */
public class ListNode {
    int val;
    ListNode next;

    ListNode(int x) {
        val = x;
        next = null;
    }

    public void print() {
        ListNode node = this;
        while (node != null) {
            System.out.print(node.val + "-->");
            node = node.next;
        }
    }
}
```

delete-node-in-a-linked-list

```
package algorithm.list;

/**
 * https://leetcode.com/problems/delete-node-in-a-linked-list/
 *
 Write a function to delete a node (except the tail) in a singly linked lis

 Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third n

 * @author xiaobaoqiu  Date: 16-5-19 Time:  下午10:15
 */
public class DeleteNodeInALinkedList {
    public static void main(String[] args) {

    }

    /**
     * 思路：将下一个节点的值复制到当前节点，然后删除下一个节点
     *
     * 1 ms
     * Your runtime beats 2.72% of java submissions
     */
    public void deleteNode(ListNode node) {
        node.val = node.next.val;
        node.next = node.next.next;
    }
}
```

intersection-of-two-linked-lists

```
package algorithm.list;

/**
 * https://leetcode.com/problems/intersection-of-two-linked-lists/
 *
 Write a program to find the node at which the intersection
 of two singly linked lists begins.


 For example, the following two linked lists:

 A:
      a1 → a2
                ↘
                  c1 → c2 → c3
                ↗
 b1 → b2 → b3
 B:

 begin to intersect at node c1.

 Notes:

 If the two linked lists have no intersection at all, return null.
 The linked lists must retain their original structure after the function r
 You may assume there are no cycles anywhere in the entire linked structure
 Your code should preferably run in O(n) time and use only O(1) memory.

 * @author xiaobaoqiu  Date: 16-7-6 Time:  下午11:10
 */
public class IntersectionOfTwoLinkedLists {
    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(3);
        ListNode node4 = new ListNode(4);
        ListNode node5 = new ListNode(5);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = node5;
        node1.print();
        System.out.println();

        ListNode node6 = new ListNode(6);
        ListNode node7 = new ListNode(7);
        node6.next = node7;
        node7.next = node4;
        node6.print();
        System.out.println();

        ListNode i = getIntersectionNode(node1, node6);
        System.out.println(i == null ? null : i.val);
    }
```

```
    /**
     * 思路：Hash存已经遍历的节点，如果是同一个节点则返回
     * 时间:O(n)
     * 空间:O(n)
     */
//    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
//        return null;
//    }

    /**
     * 思路：先得到两个链表的长度，设为  a = A.length, b = B.length, 这里假设 a >
     * 先让第一个链表先走  a-b 个节点，再开始逐步比较是否相同
     *
     * 时间:O(n)
     * 空间:O(1)
     *
     * 2 ms
     * Your runtime beats 34.83% of java submissions
     */
    public static ListNode getIntersectionNode(ListNode headA, ListNode hea
        if (headA == null || headB == null) return null;
        int a = 0, b = 0;
        ListNode A = headA, B = headB;
        while (A.next != null) {A = A.next; ++a;}
        while (B.next != null) {B = B.next; ++b;}

        if (a > b) {
            while(a > b) {headA = headA.next; --a;}
        } else {
            while(a < b) {headB = headB.next; --b;}
        }

        while (headA != null) {
            if (headA == headB) return headA;
            headA = headA.next;
            headB = headB.next;
        }
        return null;
    }
}
```

linked-list-cycle

```
package algorithm.list;

/**
 * https://leetcode.com/problems/linked-list-cycle/
 * <p/>
 * Given a linked list, determine if it has a cycle in it.
 * <p/>
 * Follow up:
 * Can you solve it without using extra space?
 *
 * @author xiaobaoqiu  Date: 16-5-27 Time:  下午11:35
 */
public class LinkedListCycle {

    public static void main(String[] args) {

    }

    /**
     * 1 ms
     * Your runtime beats 9.18% of java submissions
     */
    public boolean hasCycle(ListNode head) {
        ListNode slow = head, fast = head;
        while(slow != null && fast != null) {
            slow = slow.next;
            fast = fast.next;
            if (fast == null) return false;
            fast = fast.next;
            if (fast == slow) return true;
        }
        return false;
    }
}
```

merge-two-sorted-lists

```
package algorithm.list;

/**
 * https://leetcode.com/problems/merge-two-sorted-lists/
 *
 Merge two sorted linked lists and return it as a new list.
 The new list should be made by splicing together the nodes of the first tw

 * @author xiaobaoqiu  Date: 16-5-27 Time:  下午11:46
 */
public class MergeTwoSortedLists {

    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(3);
        node1.next = node2;
        node2.next = node3;
        node3.next = null;
        node1.print();
        System.out.println();

        ListNode node0 = new ListNode(0);
        node0.print();
        System.out.println();

        ListNode ret = mergeTwoLists(node1, node0);
        ret.print();
    }

    /**
     * 1 ms
     * Your runtime beats 11.81% of java submissions
     */
    public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;
        ListNode head = null, tail = null;
        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                if (tail != null) tail.next = l1;
                tail = l1;
                l1 = l1.next;
            } else {
                if (tail != null) tail.next = l2;
                tail = l2;
                l2 = l2.next;
            }
            if (head == null) head = tail;
        }

        while (l1 != null) {
            tail.next = l1;
            tail = l1;
```

```
            l1 = l1.next;
        }
        while (l2 != null) {
            tail.next = l2;
            tail = l2;
            l2 = l2.next;
        }

        return head;
    }
}
```

palindrome-linked-list

```
package algorithm.list;

import java.util.LinkedList;

/**
 * https://leetcode.com/problems/palindrome-linked-list/
 *

 Given a singly linked list, determine if it is a palindrome.

 Follow up:
 Could you do it in O(n) time and O(1) space?

 * @author xiaobaoqiu  Date: 16-7-11 Time:  下午9:56
 */
public class PalindromeLinkedList {
    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(3);
        ListNode node4 = new ListNode(4);
        ListNode node5 = new ListNode(5);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = node5;

        ListNode node = findMidNode(node1);
        System.out.println(node.val);

        ListNode head = reverseList(node1);
        head.print();
    }

    /**
     * 思路：快慢指针，让快指针先到末尾，则慢指针指向中间，翻转后半个链表，再比较
     *
     * 2 ms
     * Your runtime beats 33.31% of java submissions
     */
    public static boolean isPalindrome(ListNode head) {
        if (head == null || head.next == null) return true;

        //1.find mid node
        ListNode mid = findMidNode(head);

        //2.reverse later half list
        mid = reverseList(mid);

        //3.compare
        while(mid != null) {
            if (head.val != mid.val) return false;
            head = head.next;
            mid = mid.next;
```

```
        }
        return true;
    }
    /**
     * 总数为 2n，则找到第 n + 1 个
     * 总数为 2n + 1，则找到 n + 2
     */
    private static ListNode findMidNode(ListNode head) {
        ListNode quick = head, slow = head;
        while (true) {
            if (quick == null) break;
            if (quick.next == null) {
                quick = quick.next;
            } else {
                quick = quick.next.next;
            }
            slow = slow.next;
        }
        return slow;
    }
    private static ListNode reverseList(ListNode head) {
        ListNode pre = null, cur = head, next = cur.next;
        while (next != null) {
            cur.next = pre;
            pre = cur;
            cur = next;
            next = cur.next;
        }
        cur.next = pre;

        return cur;
    }
}
```

remove-duplicates-from-sorted-list

```
package algorithm.list;

/**
 * https://leetcode.com/problems/remove-duplicates-from-sorted-list/
 * <p/>
 * Given a sorted linked list, delete all duplicates such that each element
 * <p/>
 * For example,
 * Given 1->1->2, return 1->2.
 * Given 1->1->2->3->3, return 1->2->3.
 *
 * @author xiaobaoqiu  Date: 16-5-27 Time:  下午10:31
 */
public class RemoveDuplicatesFromSortedList {
    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(2);
        ListNode node4 = new ListNode(2);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = null;

        printList(node1);
        ListNode node = deleteDuplicates(node1);
        System.out.println();
        printList(node);
    }

    /**
     * 1 ms
     * Your runtime beats 25.39% of java submissions
     */
    public static ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) return head;

        ListNode node = head;
        while(node.next != null) {
            if (node.val == node.next.val) node.next = node.next.next;
            else node = node.next;
        }
        return head;
    }

    private static void printList(ListNode head) {
        ListNode node = head;
        while(node != null) {
            System.out.print(node.val + "-->");
            node = node.next;
        }
    }
}
```

remove-linked-list-elements

```
package algorithm.list;

/**
 * https://leetcode.com/problems/remove-linked-list-elements/
 *
 Remove all elements from a linked list of integers that have value val.

 Example
 Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6
 Return: 1 --> 2 --> 3 --> 4 --> 5

 * @author xiaobaoqiu  Date: 16-7-7 Time:  下午11:40
 */
public class RemoveLinkedListElements {
    public static void main(String[] args) {
        //1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node6 = new ListNode(6);
        ListNode node3 = new ListNode(3);
        ListNode node4 = new ListNode(4);
        ListNode node5 = new ListNode(5);
        ListNode node6_1 = new ListNode(6);

//        node1.next = node2;
//        node2.next = node6;
//        node6.next = node3;
//        node3.next = node4;
//        node4.next = node5;
//        node5.next = node6_1;
//        node1.print();
//        System.out.println();
////        ListNode head = removeElements(node1, 6);
//        ListNode head = removeElements(node6, 6);

//        node3.next = node4;
//        node4.next = node5;
//        node3.print();
//        System.out.println();
//        ListNode head = removeElements(node3, 6);

        node6.next = node6_1;
        node6.print();
        System.out.println();
        ListNode head = removeElements(node6, 6);

        if (head == null) System.out.println("null");
        else head.print();
    }

    /**
     * 1 ms
     * Your runtime beats 87.03% of java submissions
     */
```

```
    public static ListNode removeElements(ListNode head, int val) {
        while (head != null && head.val == val) {
            head = head.next;
        }
        if (head == null) return null;

        ListNode pre = head, node = head.next;
        while(node != null) {
            if (node.val == val) {
                pre.next = node.next;
                node = pre.next;
            } else {
                pre = node;
                node = node.next;
            }
        }
        return head;
    }
}
```

remove-nth-node-from-end-of-list

```
package algorithm.list;

/**
 * https://leetcode.com/problems/remove-nth-node-from-end-of-list/
 *
 Given a linked list, remove the nth node from the end of list
 and return its head.

 For example,

 Given linked list: 1->2->3->4->5, and n = 2.

 After removing the second node from the end, the linked list becomes 1->2·

 Note:
 Given n will always be valid.
 Try to do this in one pass.

 * @author xiaobaoqiu  Date: 16-7-5 Time:  下午11:21
 */
public class RemoveNthNodeFromEndOfList {
    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(3);
        ListNode node4 = new ListNode(4);
        ListNode node5 = new ListNode(5);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = node5;
        node1.print();
        System.out.println();

//        ListNode head = removeNthFromEnd(node1, 2);
        ListNode head = removeNthFromEnd_1(node1, 5);
        head.print();
        System.out.println();
    }

    /**
     * 两趟遍历的方法：先遍历获取总 size，再删除第 size - n 个
     * 2 ms
     * Your runtime beats 3.79% of java submissions
     */
    public static ListNode removeNthFromEnd(ListNode head, int n) {
        int size = 0;
        ListNode cur = head;
        while (cur != null) {
            cur = cur.next;
            size++;
        }
        return removeNth(head, size - n + 1);
    }
```

```
    public static ListNode removeNth(ListNode head, int n) {
        if (n == 1) return head.next;
        ListNode pre = head, cur = head.next;
        while (--n > 1) {
            pre = cur;
            cur = cur.next;
        }
        pre.next = cur.next;
        return head;
    }

    /**
     * 思路：两个指针 p，q，让p先走n个节点，再一直往下走知道 q到末尾，则p指向了要删除的
     * 1 ms
     * Your runtime beats 8.35% of java submissions
     */
    public static ListNode removeNthFromEnd_1(ListNode head, int n) {
        ListNode p = head, q = head;
        while (n-- > 0) {
            q = q.next;
        }
        if (q == null) return head.next;
        while (q.next != null) {
            p = p.next;
            q = q.next;
        }
        p.next = p.next.next;
        return head;
    }
}
```

reverse-linked-list

```
package algorithm.list;

/**
 * https://leetcode.com/problems/reverse-linked-list/
 *
 Reverse a singly linked list.

 * @author xiaobaoqiu  Date: 16-5-25 Time:  下午8:49
 */
public class ReverseLinkedList {
    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(3);
        ListNode node4 = new ListNode(4);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = null;

        printList(node1);
        ListNode newHead = reverseList(node1);
        printList(newHead);
    }

    /**
     * 0 ms
     * Your runtime beats 38.64% of java submissions
     */
    public static ListNode reverseList(ListNode head) {
        if (head == null || head.next == null) return head;

        ListNode pre = head, cur = head.next, next = head.next.next;
        head.next = null;
        while (next != null) {
            cur.next = pre;
            pre = cur;
            cur = next;
            next = next.next;
        }
        cur.next = pre;

        return cur;
    }

    private static void printList(ListNode head) {
        ListNode node = head;
        while(node != null) {
            System.out.print(node.val + "-->");
            node = node.next;
        }
    }
}
```

swap-nodes-in-pairs

```
package algorithm.list;

/**
 * https://leetcode.com/problems/swap-nodes-in-pairs/
 *
 Given a linked list, swap every two adjacent nodes and return its head.

 For example,
 Given 1->2->3->4, you should return the list as 2->1->4->3.

 Your algorithm should use only constant space.
 You may not modify the values in the list, only nodes itself can be change
 * @author xiaobaoqiu  Date: 16-5-28 Time:  上午12:05
 */
public class SwapNodesInPairs {
    public static void main(String[] args) {
        ListNode node1 = new ListNode(1);
        ListNode node2 = new ListNode(2);
        ListNode node3 = new ListNode(3);
        ListNode node4 = new ListNode(4);
        ListNode node5 = new ListNode(5);
        node1.next = node2;
        node2.next = node3;
        node3.next = node4;
        node4.next = node5;

        printList(node1);
        ListNode head = swapPairs(node1);
        System.out.println();
        printList(head);
    }

    /**
     * 0 ms
     * Your runtime beats 13.35% of java submissions
     */
    public static ListNode swapPairs(ListNode head) {
        if (head == null || head.next == null) return head;
        ListNode node = head, next = node.next, pre;
        head = head.next;

        while(node != null && next != null) {
            pre = node;
            node.next = next.next;
            next.next = node;

            node = node.next;
            if (node == null || node.next == null) break;
            next = node.next;
            pre.next = next;
        }
        return head;
    }
```

```
    private static void printList(ListNode head) {
        ListNode node = head;
        while(node != null) {
            System.out.print(node.val + "-->");
            node = node.next;
        }
        System.out.println();
    }
}
```