目录

第 1 章

List, Hashtable, Stack, Sort

1.1 Add Two Numbers

Description

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Input: `(2 -> 4 -> 3) + (5 -> 6 -> 4)`

Output: `7 -> 0 -> 8`

Solution

```
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0), cur = dummy;

    for (int carry = 0; l1 != null || l2 != null || carry > 0; ) {
        int n1 = l1 != null ? l1.val : 0;
        l1 = l1 != null ? l1.next : null;
        int n2 = l2 != null ? l2.val : 0;
        l2 = l2 != null ? l2.next : null;

        int sum = n1 + n2 + carry;
        ListNode node = new ListNode(sum % 10);
        carry = sum / 10;
        cur.next = node;
        cur = node;
    }

    return dummy.next;
}
```

1.2 Add Two Numbers II

Description

You are given two non-empty linked lists representing two non-negative integers. The most significant digit comes first and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Follow up:

What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

Example:

Input: `(7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)`

Output: `7 -> 8 -> 0 -> 7`

Solution

```
public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
    Stack<Integer> s1 = new Stack<Integer>();
    Stack<Integer> s2 = new Stack<Integer>();

    while (l1 != null) {
        s1.push(l1.val);
        l1 = l1.next;
    }
    ;
    while (l2 != null) {
        s2.push(l2.val);
        l2 = l2.next;
    }

    int sum = 0;
    ListNode list = new ListNode(0);
    while (!s1.empty() || !s2.empty()) {
        if (!s1.empty()) sum += s1.pop();
        if (!s2.empty()) sum += s2.pop();
        list.val = sum % 10;
        ListNode head = new ListNode(sum / 10);
        head.next = list;
        list = head;
        sum /= 10;
    }

    return list.val == 0 ? list.next : list;
}
```

1.3 Reverse Linked List

Description

Reverse a singly linked list.

Hint:

A linked list can be reversed either iteratively or recursively. Could you implement both?

Solution I

```
public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null) {
        return head;
    }
    ListNode next = head.next;
    ListNode newHead = reverseList(next);
    next.next = head;
    head.next = null;
    return newHead;
}
```

Solution II

```
// 耗时 0ms
public ListNode reverseList2(ListNode head) {
    ListNode dummy = new ListNode(0);

    for (ListNode p = head; p != null; ) {
        ListNode next = p.next;
        p.next = dummy.next;
        dummy.next = p;
        p = next;
    }

    return dummy.next;
}
```

1.4 Reverse Linked List II

Description

Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:

Given `1->2->3->4->5->NULL`, m = 2 and n = 4,

return `1->4->3->2->5->NULL`.

Note:

Given m, n satisfy the following condition:

$1 \le m \le n \le$ length of list.

Solution

```
public ListNode reverseBetween(ListNode head, int m, int n) {
    if (head == null) return null;
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode pre = dummy;
    for (int i = 0; i < m - 1; i++) pre = pre.next;

    ListNode start = pre.next;
    ListNode then = start.next;

    for (int i = 0; i < n - m; i++) {
        start.next = then.next;
        then.next = pre.next;
        pre.next = then;
        then = start.next;
    }

    return dummy.next;
}
```

1.5 Sort List

Description

Sort a linked list in O(n log n) time using constant space complexity.

Solution

```
public ListNode sortList(ListNode head) {
    if (head == null || head.next == null)
        return head;

    ListNode prev = null, slow = head, fast = head;

    while (fast != null && fast.next != null) {
        prev = slow;
        slow = slow.next;
        fast = fast.next.next;
    }

    prev.next = null;
    ListNode l1 = sortList(head);
    ListNode l2 = sortList(slow);
    return merge(l1, l2);
}

ListNode merge(ListNode l1, ListNode l2) {
    ListNode l = new ListNode(0), p = l;

    while (l1 != null && l2 != null) {
        if (l1.val < l2.val) {
            p.next = l1;
            l1 = l1.next;
        } else {
            p.next = l2;
            l2 = l2.next;
        }
        p = p.next;
    }

    if (l1 != null)
        p.next = l1;

    if (l2 != null)
        p.next = l2;

    return l.next;
}
```

1.6 Linked List Cycle

Description

Given a linked list, determine if it has a cycle in it.

Follow up:

Can you solve it without using extra space?

Solution

```
public boolean hasCycle(ListNode head) {
    if (head == null) {
        return false;
    }

    ListNode fast = head.next, slow = head;

    for ( ; fast != null && fast.next != null; fast = fast.next.next, slow = slow.next) {
        if (fast == slow) {
            return true;
        }
    }

    return false;
}
```

1.7 Linked List Cycle II

Description

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Note: Do not modify the linked list.

Follow up:

Can you solve it without using extra space?

Solution

```
public ListNode detectCycle(ListNode head) {
    if (head == null || head.next == null) return null;

    ListNode firstp = head;
    ListNode secondp = head;
    boolean isCycle = false;

    while (firstp != null && secondp != null) {
        firstp = firstp.next;
        if (secondp.next == null) return null;
        secondp = secondp.next.next;
        if (firstp == secondp) {
            isCycle = true;
            break;
        }
    }

    if (!isCycle) return null;
    firstp = head;
    while (firstp != secondp) {
        firstp = firstp.next;
        secondp = secondp.next;
    }

    return firstp;
}
```

1.8 Odd Even Linked List

Description

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example:

```
Given 1->2->3->4->5->NULL,
return 1->3->5->2->4->NULL.
```

Note:

The relative order inside both the even and odd groups should remain as it was in the input.

The first node is considered odd, the second node even and so on ...

Solution

```
public ListNode oddEvenList(ListNode head) {
    ListNode odd = new ListNode(0), pOdd = odd;
    ListNode even = new ListNode(0), pEven = even;

    int index = 1;
    for (ListNode p = head; p != null; p = p.next) {
        if ((index++ & 1) > 0) {
            pOdd.next = p;
            pOdd = pOdd.next;
        } else {
            pEven.next = p;
            pEven = pEven.next;
        }
    }

    pOdd.next = null;
    pEven.next = null;

    pOdd.next = even.next;
    return odd.next;
}
```

1.9 Merge Two Sorted Lists

Description

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 two lists.

Solution

```
// 耗时 15ms
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    ListNode dummy = new ListNode(0);
    ListNode p = l1, q = l2, cur = dummy;
    for ( ; p != null && q != null; ) {
        if (p.val < q.val) {
            cur.next = p;
            p = p.next;
        } else {
            cur.next = q;
            q = q.next;
        }
        cur = cur.next;
    }
    cur.next = p != null ? p : q;
    return dummy.next;
}
```

1.10 Intersection of Two Linked Lists

Description

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
B:     b1 → b2 → b3
```

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 returns.

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.

Solution

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
    int lenA = 0, lenB = 0;
    for (ListNode p = headA; p != null; p = p.next, lenA++);
    for (ListNode p = headB; p != null; p = p.next, lenB++);
    ListNode p = lenA > lenB ? headA : headB;
    ListNode q = lenA > lenB ? headB : headA;
    for (int i = 0; i < Math.abs(lenA - lenB); i++, p = p.next);
    for ( ; p != null && q != null; p = p.next, q = q.next) {
        if (p == q) {
            return p;
        }
    }
    return null;
}
```

1.11 Copy List with Random Pointer

Description

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null. Return a deep copy of the list.

Solution

```
public RandomListNode copyRandomList(RandomListNode head) {
    for (RandomListNode node = head; node != null; ) {
        RandomListNode next = node.next;

        RandomListNode copy = new RandomListNode(node.label);
        copy.next = next;
        node.next = copy;
        node = next;
    }

    for (RandomListNode node = head; node != null; ) {
        node.next.random = node.random != null ? node.random.next : null;
        node = node.next.next;
    }

    RandomListNode dummy = new RandomListNode(0), cur = dummy;
    for (RandomListNode node = head; node != null; ) {
        cur.next = node.next;
        cur = cur.next;

        node.next = node.next.next;
        node = node.next;
    }

    return dummy.next;
}
```

1.12 Merge k Sorted Lists

Description

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Solution

```
/**
 * 这里要注意 lists 中可能有 node 为 null
 */
public ListNode mergeKLists(ListNode[] lists) {
    ListNode dummy = new ListNode(0), cur = dummy;

    PriorityQueue<ListNode> queue = new PriorityQueue<>(new Comparator<ListNode>() {
        @Override
        public int compare(ListNode node1, ListNode node2) {
            if (node1.val == node2.val) {
                return 0;
            } else if (node1.val < node2.val) {
                return -1;
            } else {
                return 1;
            }
        }
    });

    for (ListNode node : lists) {
        if (node != null) {
            queue.offer(node);
        }
    }

    while (!queue.isEmpty()) {
        ListNode node = queue.poll();
        cur.next = node;
        cur = cur.next;
        if (node.next != null) {
            queue.offer(node.next);
        }
    }

    return dummy.next;
}
```

1.13 Palindrome Linked List

Description

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?

Solution

```
// 耗时 2ms
public boolean isPalindrome(ListNode head) {
    ListNode slow = head, fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    fast = reverse(slow);
    /**
     * 注意退出条件是 p1 != slow
     */
    for (ListNode p1 = head, p2 = fast; p1 != slow; p1 = p1.next, p2 = p2.next) {
        if (p1.val != p2.val) {
            return false;
        }
    }
    return true;
}

private ListNode reverse(ListNode node) {
    ListNode dummy = new ListNode(0), cur = dummy;
    while (node != null) {
        ListNode next = node.next;
        node.next = cur.next;
        cur.next = node;
        node = next;
    }
    return dummy.next;
}
```

1.14 Insertion Sort List

Description

Sort a linked list using insertion sort.

Solution

```
public ListNode insertionSortList(ListNode head) {
    if (head == null) {
        return head;
    }
    ListNode helper = new ListNode(0); //new starter of the sorted list
    ListNode cur = head; //the node will be inserted
    ListNode pre = helper; //insert node between pre and pre.next
    ListNode next = null; //the next node will be inserted
    //not the end of input list
    while (cur != null) {
        next = cur.next;
        //find the right place to insert
        while (pre.next != null && pre.next.val < cur.val) {
            pre = pre.next;
        }
        //insert between pre and pre.next
        cur.next = pre.next;
        pre.next = cur;
        pre = helper;
        cur = next;
    }
    return helper.next;
}
```

1.15 Remove Nth Node From End of List

Description

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->3->5`.

Note:

Given n will always be valid.

Try to do this in one pass.

Solution

```
public ListNode removeNthFromEnd(ListNode head, int n) {
    if (head == null) {
        return null;
    }

    ListNode p = head;
    for (int i = 1; i < n; i++) {
        p = p.next;
    }

    ListNode dummy = new ListNode(-1);
    ListNode cur = dummy;
    cur.next = head;

    for ( ; p.next != null; p = p.next) {
        cur = cur.next;
    }

    cur.next = cur.next.next;

    return dummy.next;
}
```

1.16 Reorder List

Description

Given a singly linked list L: L0?L1?⋯?Ln-1?Ln, reorder it to: L0?Ln?L1?Ln-1?L2?Ln-2?⋯

You must do this in-place without altering the nodes' values.

For example,

Given 1,2,3,4, reorder it to 1,4,2,3.

Solution

```
public void reorderList(ListNode head) {
    if (head == null || head.next == null) return;

    ListNode p1 = head;
    ListNode p2 = head;
    while (p2.next != null && p2.next.next != null) {
        p1 = p1.next;
        p2 = p2.next.next;
    }

    ListNode preMiddle = p1;
    ListNode preCurrent = p1.next;
    while (preCurrent.next != null) {
        ListNode current = preCurrent.next;
        preCurrent.next = current.next;
        current.next = preMiddle.next;
        preMiddle.next = current;
    }

    p1 = head;
    p2 = preMiddle.next;
    while (p1 != preMiddle) {
        preMiddle.next = p2.next;
        p2.next = p1.next;
        p1.next = p2;
        p1 = p2.next;
        p2 = preMiddle.next;
    }
}
```

1.17 Swap Nodes in Pairs

Description

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 changed.

Solution

```
public ListNode swapPairs(ListNode head) {
    ListNode dummy = new ListNode(0);

    ListNode node = head, tail = dummy;

    for ( ; node != null && node.next != null; ) {
        ListNode next = node.next;
        node.next = tail.next;
        tail.next = node;

        ListNode nnext = next.next;
        next.next = node;
        tail.next = next;
        tail = node;

        node = nnext;
    }

    tail.next = node;

    return dummy.next;
}
```

1.18 Remove Linked List Elements

Description

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`

Solution

```
public ListNode removeElements(ListNode head, int val) {
    ListNode dummy = new ListNode(0), node = dummy;
    for ( ; head != null; head = head.next) {
        if (head.val != val) {
            node.next = head;
            node = node.next;
        }
    }
    node.next = null;
    return dummy.next;
}
```

1.19 Remove Duplicates from Sorted List

Description

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,

Given `1->1->2`, return `1->2`.

Given `1->1->2->3->3`, return `1->2->3`.

Solution

```
public ListNode deleteDuplicates(ListNode head) {
    ListNode dummy = new ListNode(0), cur = dummy;
    for ( ; head != null; head = head.next) {
        if (cur == dummy || head.val != cur.val) {
            cur.next = head;
            cur = cur.next;
        }
    }
    cur.next = null;
    return dummy.next;
}
```

1.20 Remove Duplicates from Sorted List II

Description

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,

Given `1->2->3->3->4->4->5`, return `1->2->5`. Given `1->1->1->2->3`, return `2->3`.

Solution

```
public ListNode deleteDuplicates(ListNode head) {
    if (head == null) {
        return null;
    }

    ListNode dummy = new ListNode(0), tail = dummy;
    ListNode prev = head, cur = head.next;

    for ( ; cur != null; cur = cur.next) {
        if (prev.val != cur.val) {
            if (prev.next == cur) {
                tail.next = prev;
                tail = tail.next;
            }
            prev = cur;
        }
    }

    tail.next = prev.next == null ? prev : null;
    return dummy.next;
}
```

1.21 Convert Sorted List to Binary Search Tree

Description

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

Solution

```
public TreeNode sortedListToBST(ListNode head) {
    if (head == null) return null;
    return toBST(head, null);
}

public TreeNode toBST(ListNode head, ListNode tail) {
    ListNode slow = head;
    ListNode fast = head;
    if (head == tail) return null;

    while (fast != tail && fast.next != tail) {
        fast = fast.next.next;
        slow = slow.next;
    }
    TreeNode thead = new TreeNode(slow.val);
    thead.left = toBST(head, slow);
    thead.right = toBST(slow.next, tail);
    return thead;
}
```

1.22 Partition List

Description

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,

Given `1->4->3->2->5->2` and x = 3,

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

Solution

```
ListNode partition(ListNode head, int x) {
    ListNode node1 = new ListNode(0);
    ListNode node2 = new ListNode(0);
    ListNode p1 = node1, p2 = node2;

    while (head != null) {
        if (head.val < x)
            p1 = p1.next = head;
        else
            p2 = p2.next = head;
        head = head.next;
    }
    p2.next = null;
    p1.next = node2.next;
    return node1.next;
}
```

1.23 Reverse Nodes in k-Group

Description

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,

Given this linked list: `1->2->3->4->5`

For k = 2, you should return: `2->1->4->3->5`

For k = 3, you should return: `3->2->1->4->5`

Solution

```
public ListNode reverseKGroup(ListNode head, int k) {
    int size = 0;
    ListNode dummy = new ListNode(0), cur = dummy, p;
    for (p = head; p != null; p = p.next, size++);

    for (p = head; size >= k; size -= k) {
        ListNode tail = p;
        for (int i = 0; i < k; i++) {
            ListNode next = p.next;
            p.next = cur.next;
            cur.next = p;
            p = next;
        }
        cur = tail;
    }
    cur.next = p;
    return dummy.next;
}
```

1.24 Rotate List

Description

Given a list, rotate the list to the right by k places, where k is non-negative.

For example:

Given `1->2->3->4->5->NULL` and k = 2, return `4->5->1->2->3->NULL`.

Solution

```
public ListNode rotateRight(ListNode head, int n) {
    if (head == null || head.next == null) return head;
    ListNode dummy = new ListNode(0);
    dummy.next = head;
    ListNode fast = dummy, slow = dummy;

    int i;
    for (i = 0; fast.next != null; i++)//Get the total length
        fast = fast.next;

    for (int j = i - n % i; j > 0; j--) //Get the i-n%i th node
        slow = slow.next;

    fast.next = dummy.next; //Do the rotation
    dummy.next = slow.next;
    slow.next = null;

    return dummy.next;
}
```

1.25 Plus One Linked List

Description

Given a non-negative integer represented as non-empty a singly linked list of digits, plus one to the integer.

You may assume the integer do not contain any leading zero, except the number 0 itself.

The digits are stored such that the most significant digit is at the head of the list.

Example:

```
Input:
1->2->3

Output:
1->2->4
```

Solution

```
public ListNode plusOne(ListNode head) {
    if (head == null) {
        return head;
    }
    Stack<ListNode> stack = new Stack<ListNode>();
    for (ListNode node = head; node != null; node = node.next) {
        stack.push(node);
    }
    int k = 1;
    while (!stack.isEmpty()) {
        ListNode node = stack.pop();
        int val = node.val + k;
        node.val = val % 10;
        k = val / 10;
        if (k == 0) {
            return head;
        }
    }
    ListNode node = new ListNode(k);
    node.next = head;
    return node;
}
```