链滴

常见算法总结 - 链表篇

作者：valarchie

原文链接：https://ld246.com/article/1588388018595

来源网站：链滴

本文总结了常见高频的关于链表的算法考察。

1.如何找到链表的中间元素?

我们可以采用快慢指针的思想，使用步长为1的慢指针和步长为2的快指针，当快指针抵达链表末尾时此时慢指针指向的即为中点位置。

```
public static LinkNode findMiddleByPointer(LinkNode node) {

    LinkNode slow = node;
    LinkNode fast = node;
    // 检测快指针是否可以安全移动
    while (fast.next != null && fast.next.next != null) {

        slow = slow.next;
        fast = fast.next.next;

    }

    return slow;

}
```

我们还可以采用递归的方式，当递归到最末尾的时候，我们已经能知道链表的长度，此时当递归回去时候，判断当前递归层级等于链表长度一半的时候，即为链表的重点。

```
public static void findMiddleByRecursion(LinkNode node, int recursionIndex) {

        if (node.next != null) {
            findMiddleByRecursion(node.next, recursionIndex + 1);
        } else {
            middleIndex = recursionIndex % 2 == 0 ? recursionIndex / 2 : recursionIndex / 2 + 1;
        }

        if (middleIndex == recursionIndex) {
            System.out.println(node.value);
        }

        return;

    }
```

2.检测链表是否有环。

检测链表是否有环是非常常见的算法考察。常见的就是使用快慢指针，如果快慢指针有重合的时候，明链表是有环的。

```
 public static boolean isExistCircle(LinkNode node) {

        LinkNode slow = node;
        LinkNode fast = node;
```

```
        while (fast.next != null && fast.next.next != null) {

            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast) {
                return true;
            }

        }

        return false;
}
```

3.如何列表反转（递归）

我们可以在递归回溯的时候，反向更改节点的指针。

```
 public static void reverseLinkListByRecursion(LinkNode cur, LinkNode next) {

        if (next == null) {
            return;
        }

        reverseLinkListByRecursion(next, next.next);
        next.next = cur;
        cur.next = null;

        return;

}
```

4.如何反转链表（非递归）

反转链表的非递归方式，我们可以采用三个指针，分别指向当前节点，下个节点，下下个节点，调整下个节点的next指向后，继续利用下下个节点进行往后操作。

```
public static void reverseLinkListByPointer(LinkNode node) {

        LinkNode cur = node;
        LinkNode next = node.next;
        LinkNode nextNext = null;

        cur.next = null;

        while (next != null) {
            nextNext = next.next;
            next.next = cur;
            cur = next;
            next = nextNext;
        }
```

```
}
```

5.删除经过排序的链表中重复的节点。

通过在遍历中，判断当前的数字是否与之前的重复数字相同，如果相同的话，直接跳过，直到找到与前重复数字不同时，将原先的指针跳过重复的节点，指向当前非重复数字节点。

```
public static void removeDuplicateNode(LinkNode node) {

        if (node == null) {
            return;
        }

        LinkNode cur = node;
        LinkNode next = node.next;
        int dupicateVal = node.value;

        while (next != null) {
            if (next.value == dupicateVal) {
                next = next.next;
                continue;
            }
            dupicateVal = next.value;

            cur.next = next;
            cur = next;
            next = next.next;
        }
    }
```

6.如何计算两个链表的代表数字之和。

将链表代表的数字进行相加即可，注意首位代表的是个位。3->1->5 代表513，5->9->2 代表295 最终计算结果为 8->0->8， 808。

```
 public static LinkNode sumTwoLinkList(LinkNode num1, LinkNode num2) {

        // 如果其中一个链表为空的，直接当做0处理，返回另外一个链表
        if (num1 == null) {
            return num2;
        }
        if (num2 == null) {
            return num1;
        }
        LinkNode sum = new LinkNode();
        // 保存头结点，如果计算完成后直接返回头结点
        LinkNode head = sum;
        // 是否进位
        boolean isOver = false;
        // 当两个节点，其中一个存在时，即可进行累加
        while (num1 != null || num2 != null) {
            // 默认初始化为0
            int num1Val = 0;
```

```
            int num2Val = 0;

            if (num1 != null) {
                num1Val = num1.value;
            }
            if (num2 != null) {
                num2Val = num2.value;
            }
            // 如果进位的话 多加1
            int singleSum = num1Val + num2Val + (isOver == true ? 1 : 0);

            if (singleSum >= 10) {
                isOver = true;
                singleSum = singleSum % 10;
            } else {
                isOver = false;
            }

            sum.value = singleSum;
            // 移动指针
            num1 = num1 != null ? num1.next : null;
            num2 = num2 != null ? num2.next : null;

            // 没有数字相加，直接结束
            if (num1 == null && num2 == null) {
                break;
            } else {
            // 还有数字相加的话 提前生成下个数字
                sum.next = new LinkNode();
                sum = sum.next;
            }
        }
        return head;
}
```

7.链表-奇数位升序偶数位降序-让链表变成升序

先将链表拆分成奇数的链表，和偶数的链表，然后翻转偶数的链表，在合并两个链表。

```
public class SortAscDescLinkList {

    public static LinkNode oddLinkList = null;

    public static LinkNode evenLinkList = null;


    public static void main(String[] args) {

        // 初始化测试链表
        LinkNode x1 = new LinkNode(1);
        LinkNode x2 = new LinkNode(10);
        LinkNode x3 = new LinkNode(2);
        LinkNode x4 = new LinkNode(9);
        LinkNode x5 = new LinkNode(3);
```

```
        LinkNode x6 = new LinkNode(8);
        LinkNode x7 = new LinkNode(4);
        LinkNode x8 = new LinkNode(7);
        LinkNode x9 = new LinkNode(5);
        LinkNode x10 = new LinkNode(6);

        x1.setNext(x2).setNext(x3).setNext(x4).setNext(x5).setNext(x6).setNext(x7).setNext(x8).se
Next(x9).setNext(x10);

        // 先按奇数偶数位拆分链表
        splitList(x1);
        // 再反转链表
        evenLinkList = reverseLinkList(evenLinkList);
        // 再合并链表
        mergeList(oddLinkList, evenLinkList);

        oddLinkList.printList();


    }


    /**
     * 拆分两个链表
     * @param node
     */
    public static void splitList(LinkNode node) {

        oddLinkList = node;
        evenLinkList = node.next;

        LinkNode oddCur = node;
        LinkNode evenCur = node.next;

        while (oddCur != null || evenCur != null) {

            if (oddCur.next != null && oddCur.next.next != null) {
                oddCur.next = oddCur.next.next;
                oddCur = oddCur.next;
            }else {
                oddCur.next = null;
                oddCur = null;
            }

            if (evenCur.next != null && evenCur.next.next != null) {
                evenCur.next = evenCur.next.next;
                evenCur = evenCur.next;
            }else {
                evenCur.next = null;
                evenCur = null;
            }

        }
```

```
    }

    /**
     * 反转链表
     * @param node
     * @return
     */
    public static LinkNode reverseLinkList(LinkNode node) {

        LinkNode cur = node;
        LinkNode next = node.next;
        LinkNode nextNext = null;

        cur.next = null;

        while (next != null) {

            nextNext = next.next;
            next.next = cur;
            cur = next;
            next = nextNext;

        }

        return cur;

    }


    /**
     * 合并两个链表
     * @param oddLinkList
     * @param evenLinkList
     */
    public static void mergeList(LinkNode oddLinkList, LinkNode evenLinkList) {

        LinkNode oddTail = oddLinkList;

        while (oddTail.next != null) {
            oddTail = oddTail.next;
        }

        oddTail.next = evenLinkList;

    }


}
```

8.一个单向链表，删除倒数第N个数据。

准备两个指针，初始指向头结点，1号指针先走n步，然后2号指针开始走，当1号指针走到尾节点时，

号指针即为倒数第N个数据。

```
public static LinkNode findLastKNumber(LinkNode node, int k) {

        LinkNode fast = node;
        LinkNode slow = node;

        for (int i = 0; i < k; i++) {
            // 如果fast为空的话，说明k超出范围
            if (fast == null) {
                throw new RuntimeException("超出链表范围！");
            }
            fast = fast.next;
        }

        while (fast != null) {
            fast = fast.next;
            slow = slow.next;
        }
        return slow;
}
```

笔者个人总结，如有错误之处望不吝指出。