# 0092. 反转链表 II

ITCharge 大约 5 分钟

- 标签：链表
- 难度：中等

## 题目链接

- 0092. 反转链表 II - 力扣

## 题目大意

**描述**：给定单链表的头指针 `head` 和两个整数 `left` 和 `right` ，其中 `left <= right` 。

**要求**：反转从位置 `left` 到位置 `right` 的链表节点，返回反转后的链表 。

**说明**：

- 链表中节点数目为 `n` 。
- $1 \leq n \leq 500$。
- $-500 \leq Node.val \leq 500$。
- $1 \leq left \leq right \leq n$。

**示例**：

- 示例 1：

```py
输入：head = [1,2,3,4,5], left = 2, right = 4
输出：[1,4,3,2,5]
```

## 解题思路

在「0206. 反转链表 」中我们可以通过迭代、递归两种方法将整个链表反转。这道题而这道题要求对链表的部分区间进行反转。我们同样可以通过迭代、递归两种方法将链表的部分区间进行反转。

## 思路 1：迭代

我们可以先遍历到需要反转的链表区间的前一个节点，然后对需要反转的链表区间进行迭代反转。最后再返回头节点即可。

但是需要注意一点，如果需要反转的区间包含了链表的第一个节点，那么我们可以事先创建一个哑节点作为链表初始位置开始遍历，这样就能避免找不到需要反转的链表区间的前一个节点。

这道题的具体解题步骤如下：

1. 先使用哑节点 `dummy_head` 构造一个指向 `head` 的指针，使得可以从 `head` 开始遍历。使用 `index` 记录当前元素的序号。
2. 我们使用一个指针 `reverse_start` ，初始赋值为 `dummy_head` 。然后向右逐步移动到需要反转的区间的前一个节点。
3. 然后再使用两个指针 `cur` 和 `pre` 进行迭代。 `pre` 指向 `cur` 前一个节点位置，即 `pre` 指向需要反转节点的前一个节点， `cur` 指向需要反转的节点。初始时， `pre` 指向 `reverse_start` ， `cur` 指向 `pre.next` 。
4. 当当前节点 `cur` 不为空，且 `index` 在反转区间内时，将 `pre` 和 `cur` 的前后指针进行交换，指针更替顺序为：
    1. 使用 `next` 指针保存当前节点 `cur` 的后一个节点，即 `next = cur.next` ；
    2. 断开当前节点 `cur` 的后一节点链接，将 `cur` 的 `next` 指针指向前一节点 `pre` ，即 `cur.next = pre` ；
    3. `pre` 向前移动一步，移动到 `cur` 位置，即 `pre = cur` ；
    4. `cur` 向前移动一步，移动到之前 `next` 指针保存的位置，即 `cur = next` 。
    5. 然后令 `index` 加 `1` 。
5. 继续执行第 `4` 步中的 `1` 、 `2` 、 `3` 、 `4` 、 `5` 步。
6. 最后等到 `cur` 遍历到链表末尾（即 `cur == None` ）或者遍历到需要反转区间的末尾时（即 `index > right` ）时，将反转区间的头尾节点分别与之前保存的需要反转的区间的前一个节点 `reverse_start` 相连，即 `reverse_start.next.next = cur` ， `reverse_start.next = pre` 。
7. 最后返回新的头节点 `dummy_head.next` 。

## 思路 1：代码

```py
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseBetween(self, head: ListNode, left: int, right: int) -> ListNode:
        index = 1
        dummy_head = ListNode(0)
        dummy_head.next = head
        pre = dummy_head

        reverse_start = dummy_head
        while reverse_start.next and index < left:
            reverse_start = reverse_start.next
            index += 1

        pre = reverse_start
        cur = pre.next
        while cur and index <= right:
            next = cur.next
            cur.next = pre
            pre = cur
            cur = next
            index += 1

        reverse_start.next.next = cur
        reverse_start.next = pre

        return dummy_head.next
```

## 思路 1：复杂度分析

- **时间复杂度**：$O(n)$。其中 $n$ 是链表节点个数。
- **空间复杂度**：$O(1)$。

## 思路 2：递归算法

### 1. 翻转链表前 n 个节点

1. 当 `left == 1` 时，无论 `right` 等于多少，实际上都是将当前链表到 `right` 部分进行翻转，也就是将前 `right` 个节点进行翻转。
2. 我们可以先定义一个递归函数 `reverseN(self, head, n)` ，含义为：将链表前第 $n$ 个节点位置进行翻转。
    1. 然后从 `head.next` 的位置开始调用递归函数，即将 `head.next` 为头节点的链表的的前 $n-1$ 个位置进行反转，并返回该链表的新头节点 `new_head` 。
    2. 然后改变 `head` （原先头节点）和 `new_head` （新头节点）之间的指向关系，即将 `head` 指向的节点作为 `head` 下一个节点的下一个节点。
    3. 先保存 `head.next` 的 `next` 指针，也就是新链表前 $n$ 个节点的尾指针，即 `last = head.next.next` 。
    4. 将 `head.next` 的 `next` 指针先指向当前节点 `head` ，即 `head.next.next = head` 。
    5. 然后让当前节点 `head` 的 `next` 指针指向 `last` ，则完成了前 $n-1$ 个位置的翻转。
3. 递归终止条件：当 `n == 1` 时，相当于翻转第一个节点，直接返回 `head` 即可。

### 4. 翻转链表 `[left, right]` 上的节点。

接下来我们来翻转区间上的节点。

1. 定义递归函数 `reverseBetween(self, head, left, right)` 为
2.

## 思路 2：代码

```py
class Solution:
    def reverseBetween(self, head: Optional[ListNode], left: int, right: int) ->
Optional[ListNode]:
        if left == 1:
            return self.reverseN(head, right)

        head.next = self.reverseBetween(head.next, left - 1, right - 1)
```

```
        return head

    def reverseN(self, head, n):
        if n == 1:
            return head
        last = self.reverseN(head.next, n - 1)
        next = head.next.next
        head.next.next = head
        head.next = next
        return last
```

## 思路 2：复杂度分析

- **时间复杂度**：$O(n)$。
- **空间复杂度**：$O(n)$。最多需要 $n$ 层栈空间。

# 参考资料

- 【题解】动画图解：翻转链表的指定区间 - 反转链表 II - 力扣
- 【题解】【宫水三叶】一个能应用所有「链表」题里的「哨兵」技巧 - 反转链表 II - 力扣