# 题目

反转从位置 m 到 n 的链表。请使用一趟扫描完成反转。

**说明:**

1 ≤m≤n≤ 链表长度。

**示例:**

输入: 1->2->3->4->5->NULL, m = 2, n = 4

输出: 1->4->3->2->5->NULL

# 分析

## 方法一：原地翻转

思路：

这个与原地链表反转类似，只是增加了一个起点的特殊处理。

代码：

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode\* reverseBetween(ListNode\* head, int m, int n) {

ListNode\* dummy=new ListNode(-1);

ListNode\* pre=dummy;

dummy->next=head;

//移动到第m个节点

for(int i=0;i<m-1;i++)

pre=pre->next;

ListNode\* cur=pre->next;

for(int i=m;i<n;i++){

//原地翻转链表

ListNode\* tmpNode=cur->next;

cur->next= tmpNode ->next;

tmpNode ->next=pre->next;

pre->next= tmpNode;

}

return dummy->next;

}

};

或：

class Solution {

public:

ListNode\* reverseBetween(ListNode\* head, int m, int n) {

if (head == nullptr || head->next == nullptr || m == n) {

return head;

}

ListNode\* dummy = new ListNode(0); // 构造一个哑结点

dummy->next = head;

ListNode\* pre = dummy;

for (int i = 0; i < m - 1; ++i) {

pre = pre->next;

}

ListNode\* cur = pre->next;

ListNode\* next = nullptr; // 可以认为是tmpNode

for (int i = 0; i < n - m; ++i) {

next = cur->next;

cur->next = next->next;

next->next = pre->next;

pre->next = next;

}

return dummy->next;

}

};

或：

class Solution {

private:

void reverseLinkedList(ListNode \*head) {

// 也可以使用递归反转一个链表

ListNode \*pre = nullptr;

ListNode \*cur = head;

while (cur != nullptr) {

ListNode \*next = cur->next;

cur->next = pre;

pre = cur;

cur = next;

}

}

public:

ListNode \*reverseBetween(ListNode \*head, int left, int right) {

// 因为头节点有可能发生变化，使用虚拟头节点可以避免复杂的分类讨论

ListNode \*dummyNode = new ListNode(-1);

dummyNode->next = head;

ListNode \*pre = dummyNode;

// 第 1 步：从虚拟头节点走 left - 1 步，来到 left 节点的前一个节点

// 建议写在 for 循环里，语义清晰

for (int i = 0; i < left - 1; i++) {

pre = pre->next;

}

// 第 2 步：从 pre 再走 right - left + 1 步，来到 right 节点

ListNode \*rightNode = pre;

for (int i = 0; i < right - left + 1; i++) {

rightNode = rightNode->next;

}

// 第 3 步：切断出一个子链表（截取链表）

ListNode \*leftNode = pre->next;

ListNode \*curr = rightNode->next;

// 注意：切断链接

pre->next = nullptr;

rightNode->next = nullptr;

// 第 4 步：同第 206 题，反转链表的子区间

reverseLinkedList(leftNode);

// 第 5 步：接回到原来的链表中

pre->next = rightNode;

leftNode->next = curr;

return dummyNode->next;

}

};

**复杂度分析：**

时间复杂度：O(N)，其中N是链表总节点数。最坏情况下，需要遍历整个链表。

空间复杂度：O(1)。只使用到常数个变量。

## 方法二：双指针

**思路：**

以第m-1节点为"头节点"，将[m,n]范围的节点不断插入"头节点"后，最后，第m个节点指向第n+1个节点。

**代码：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution {

public:

ListNode\* reverseBetween(ListNode\* head, int m, int n) {

ListNode \*dummy = new ListNode(-1);

dummy->next = head;

// 第m-1个节点

ListNode \*pre=dummy;

for(int i=1; i<m; i++){

pre = pre->next;

}

// 第m个节点

ListNode \*t, \*cur=pre->next, \*mNode=pre->next;

// 头插法

for(int i=m; i<=n; i++){

t = cur->next;

cur->next = pre->next;

pre->next = cur;

cur = t;

}

// 第m个节点指向第n+1个节点

mNode->next = cur;

return dummy->next;

}

};