# 题目

给定一个单链表，把所有的奇数节点和偶数节点分别排在一起。请注意，这里的奇数节点和偶数节点指的是节点编号的奇偶性，而不是节点的值的奇偶性。

请尝试使用原地算法完成。你的算法的空间复杂度应为O(1)，时间复杂度应为O(nodes)，nodes为节点总数。

**示例 1:**

输入: 1->2->3->4->5->NULL

输出: 1->3->5->2->4->NULL

**示例 2:**

输入: 2->1->3->5->6->4->7->NULL

输出: 2->3->6->7->1->5->4->NULL

**说明:**

应当保持奇数节点和偶数节点的相对顺序。

链表的第一个节点视为奇数节点，第二个节点视为偶数节点，以此类推。

# 分析

## 方法一：双指针

**思路：**

双指针法求解，一个指向奇数链表，另一个指向偶数链表，每次迭代后注意奇偶连接，然后迭代进行。

注意考虑pre和cur进行一次迭代后如何衔接，ListNode\* tmp=pre->next表明每次衔接点的位置（如果用cur进行衔接的话会丢失一些元素），最后用pre->next->next=tmp可以保证奇偶链表衔接正常。

**代码：**

class Solution {

public:

ListNode\* oddEvenList(ListNode\* head) {

if(!head || !head->next) return head;

ListNode \*pre=head,\*cur=head->next;

while(cur && cur->next){

ListNode\* tmp=pre->next;

pre->next=cur->next;

cur->next=cur->next->next;

pre->next->next=tmp;

pre=pre->next;

cur=cur->next;

}

return head;

}

};

## 方法二：三指针

## 方法三：两链表合并

**思路：**

这种方法比较具有普适性，对于链表重排非常有效！

**代码：**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

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

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

\* };

\*/

class Solution {

public:

ListNode\* oddEvenList(ListNode\* head) {

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

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

// 这里也可以不申请内存，直接声明一下即可

ListNode \*tmpOddNode = oddNode;

ListNode \*tmpEvenNode = evenNode;

int i = 1;

while (head) {

if (i % 2 != 0) {

oddNode->next = head;

oddNode = oddNode->next;

} else {

evenNode->next = head;

evenNode = evenNode->next;

}

i++;

head = head->next;

}

evenNode->next = nullptr;

oddNode->next = tmpEvenNode->next;

return tmpOddNode->next;

}

};

或：

class Solution {

public:

ListNode\* oddEvenList(ListNode\* head) {

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

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

ListNode \*tmpOddNode = oddNode;

ListNode \*tmpOEvenNode = evenNode;

int i=0;

while(head)

{

i++;

if(i%2!=0)

{

oddNode->next = head;

oddNode = oddNode->next;

}

else

{

evenNode->next = head;

evenNode = evenNode->next;

}

head->next;

}

evenNode->next = NULL;

oddNode->next = tmpEvenNode->next;

return tmpOddNode->next;

}

};

说明：这种方法类似LeetCode86，但是不符合题目的要求。