# 题目

删除链表中等于给定值 val 的所有节点。

**示例:**

输入: 1->2->6->3->4->5->6, val = 6

输出: 1->2->3->4->5

# 分析

## 方法一：递归法

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

if (!head)

return head;

head->next = removeElements(head->next, val);

return head->val == val ? head->next : head;

}

};

## 方法二：原链表

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

ListNode \*p=head;//当前判断元素

ListNode \*k=p;//指向前一个元素

while(p)

{

if(p->val==val)

{

if(p==head)//头元素要删的情况

{

head=p->next;

p=head;

if(head==NULL)

return head;

else continue;

}

else if(p->next==NULL)//末尾元素要删的情况

{

k->next=NULL;

return head;

}

else//中间元素

{

k->next=p->next;

p=k->next;

continue;

}

}

k=p;

p=p->next;

}

return head;

}

};

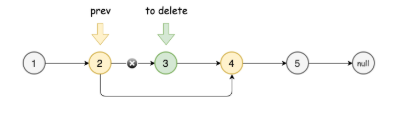
## 方法三：新链表/哨兵节点

**思路：**

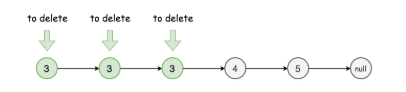
如果删除的节点是中间的节点，则问题似乎非常简单：

选择要删除节点的前一个结点prev。

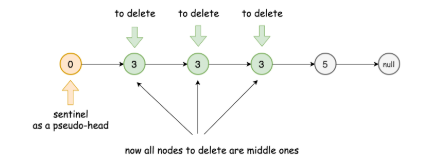
将prev的next设置为要删除结点的next。



当要删除的一个或多个节点位于链表的头部时，事情会变得复杂。



可以通过哨兵节点去解决它，哨兵节点广泛应用于树和链表中，如伪头、伪尾、标记等，它们是纯功能的，通常不保存任何数据，其主要目的是使链表标准化，如使链表永不为空、永不无头、简化插入和删除。



在这里哨兵节点将被用于伪头。

**算法：**

1、初始化哨兵节点为 ListNode(0) 且设置 sentinel.next = head。

2、初始化两个指针 curr 和 prev 指向当前节点和前继节点。

3、当 curr != nullptr：

3.1 比较当前节点和要删除的节点：

若当前节点就是要删除的节点：则 prev.next = curr.next。

否则设 prve = curr。

3.2 遍历下一个元素：curr = curr.next。

4、返回 sentinel.next。

**代码：**

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

ListNode\* sentinel = new ListNode(0);

sentinel->next = head;

ListNode \*prev = sentinel, \*curr = head, \*toDelete = nullptr;

while (curr != nullptr) {

if (curr->val == val) {

prev->next = curr->next;

toDelete = curr;

} else prev = curr;

curr = curr->next;

if (toDelete != nullptr) {

delete toDelete;

toDelete = nullptr;

}

}

ListNode \*ret = sentinel->next;

delete sentinel;

return ret;

}

};

代码：

class Solution {

public:

ListNode\* removeElements(ListNode\* head, int val) {

ListNode \*sentinel=new ListNode(INT\_MAX);

sentinel->next=head; //构造头结点

ListNode \*p=sentinel,\*temp;

while(p->next){

if(p->next->val==val){

temp=p->next;

p->next=p->next->next;

delete temp;

}

else

p=p->next;

}

return sentinel->next;

}

};

复杂度分析：

时间复杂度：O(N)，只遍历了一次。

空间复杂度：O(1)。