

题目一：移除字符串中的尾随0

题目描述 | 通过 x | 笔记 x | 题解 | 提交记录

2710. 移除字符串中的尾随零

已解答

简单 | 相关标签 | 相关企业 | 提示 | Aa

给你一个用字符串表示的正整数 `num`，请你以字符串形式返回不含尾随零的整数 `num`。

示例 1:

输入: `num = "51230100"`

输出: `"512301"`

解释: 整数 `"51230100"` 有 2 个尾随零，移除并返回整数 `"512301"`。

示例 2:

输入: `num = "123"`

输出: `"123"`

解释: 整数 `"123"` 不含尾随零，返回整数 `"123"`。

提示:

- `1 <= num.length <= 1000`
- `num` 仅由数字 `0` 到 `9` 组成
- `num` 不含前导零

面试中遇到过这道题? 1/5

C++
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
已存
通
输入
ni
"
输出
"

解题思路一：

利用c++的特性，对于尾巴有0的字符串，我们直接`nums.pop_back`即直接去除尾巴

```
class Solution {
public:
    string removeTrailingZeros(string num) {
        //处理空
        if (num.empty()) return num;
        int len = num.size();
        if (num[len - 1] != '0') return num;
        else
        {
            for (int i = len - 1; i > 0; i--)
            {
                if (num[i] == '0')
                {
```

```

        num.pop_back();
    }
    else
        break;
}
}
return num;
}
};

```

解题思路二：

利用substr函数对字符串进行覆盖,利用find_last_not_of从末尾开始查找值

```

class Solution {
public:
    string removeTrailingZeros(string num) {
        return num.substr(0, num.find_last_not_of('0') + 1);
    }
};

```

解题思路三：


利用erase函数进行擦除

```

class Solution {
public:
    string removeTrailingZeros(string s) {
        s.erase(s.begin() + 1 + s.find_last_not_of('0'), s.end()); // 原地操作
        return s;
    }
};

```

题目二：分隔链表

 题库 < > 运行

题目描述 | 通过 x | 笔记 x | 题解 | 提交记录

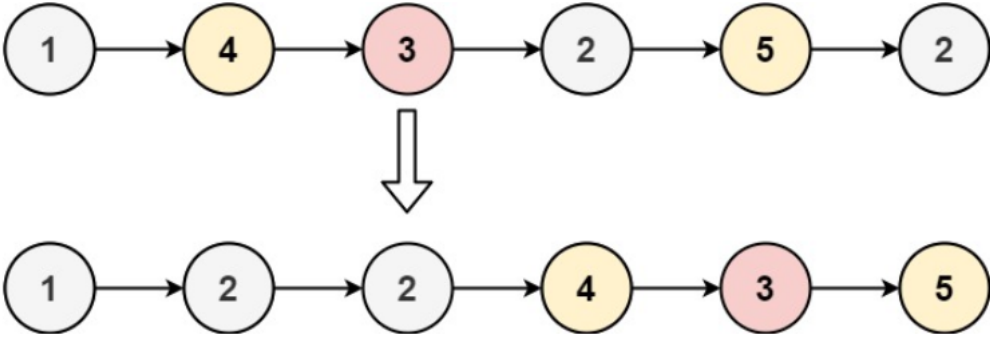
86. 分隔链表

中等 相关标签 相关企业 Aa

给你一个链表的头节点 `head` 和一个特定值 `x`，请你对链表进行分隔，使得所有 **小于** `x` 的节点都出现在 **大于或等于** `x` 的节点之前。

你应当 **保留** 两个分区中每个节点的初始相对位置。


示例 1:



输入: `head = [1,4,3,2,5,2]`, `x = 3`
输出: `[1,2,2,4,3,5]`

示例 2:

输入: `head = [2,1]`, `x = 2`
输出: `[1,2]`

已解答 

已存储

测试通过

输入

head

[1

x =

3

解题思路一：

双指针法，虚拟两个链表less和more，一个存大于一个存小于，最后把小于的尾节点连接到大于的头节点上

```
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        // 使用虚拟节点代表小于x和大于等于x的链表部分
        ListNode lessDummy(0), moreDummy(0);
        ListNode* less = &lessDummy, * more = &moreDummy;

        // 遍历原始链表并将节点分别添加到less和more链表
        ListNode* cur = head;
        while (cur) {
            if (cur->val < x) {
                less->next = cur;
                less = less->next;
            }
            else {
                more->next = cur;
            }
        }
        less->next = more->next;
        return lessDummy->next;
    }
};
```

```

        more = more->next;
    }
    cur = cur->next;
}

// 将less链表的末尾与more链表的开头连接
less->next = moreDummy.next;
// 确保more链表的末尾节点指向nullptr, 终止链表
more->next = nullptr;

// 返回新链表的头节点
return lessDummy.next;
}
};

```

解题思路二：

利用队列，向量等数据结构，将数据分割成两块，和解题思路一同样的处理方式但是占用了额外空间，在处理时需要判断队列是否为空且对末尾的指向需要明确

```

class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        // 队列存储小于 x 和大于等于 x 的节点
        queue<ListNode*> lessThan;
        queue<ListNode*> greaterOrEqual;

        // 遍历链表，将节点加入相应的队列
        ListNode* cur = head;
        while (cur) {
            if (cur->val < x) {
                lessThan.push(cur);
            } else {
                greaterOrEqual.push(cur);
            }
            cur = cur->next;
        }

        // 创建新的链表，先处理 lessThan 队列
        ListNode dummy(0);
        ListNode* newCur = &dummy;

        while (!lessThan.empty()) {
            newCur->next = lessThan.front();
            lessThan.pop();
            newCur = newCur->next;
        }

        // 再处理 greaterOrEqual 队列
        while (!greaterOrEqual.empty()) {
            newCur->next = greaterOrEqual.front();
            greaterOrEqual.pop();
            newCur = newCur->next;
        }
    }
};

```

```
        // 确保最后一个节点的 next 指针为 nullptr
        newCur->next = nullptr;

        // 返回新链表的头节点
        return dummy.next;
    }
};
```