

数据结构暑期

算法练习报告

专业班级： 19C199

学生姓名： 张宁

学生学号： 20191001382

目 录

[一、链表算法练习 6](#_Toc47732260)

[实验一、35.翻转链表 6](#_Toc47732261)

[1、主要任务 6](#_Toc47732262)

[2、算法原理 6](#_Toc47732263)

[3、实验数据测试 7](#_Toc47732264)

[实验二、96.链表划分 7](#_Toc47732265)

[1、主要任务 7](#_Toc47732266)

[2、算法原理 8](#_Toc47732267)

[3、实验数据测试 8](#_Toc47732268)

[实验三、112.删除排序链表中的重复元素 9](#_Toc47732269)

[1、主要任务 9](#_Toc47732270)

[2、算法原理 9](#_Toc47732271)

[3、实验数据测试 10](#_Toc47732272)

[实验四、165.合并两个排序链表 10](#_Toc47732273)

[1、主要任务 10](#_Toc47732274)

[2、算法原理 11](#_Toc47732275)

[3、实验数据测试 11](#_Toc47732276)

[实验五、166.链表倒数第n个结点 12](#_Toc47732277)

[1、主要任务 12](#_Toc47732278)

[2、算法原理 12](#_Toc47732279)

[3、实验数据测试 13](#_Toc47732280)

[实验六、167.链表求和 13](#_Toc47732281)

[1、主要任务 13](#_Toc47732282)

[2、算法原理 14](#_Toc47732283)

[3、实验数据测试 14](#_Toc47732284)

[实验七、173.链表插入排序 15](#_Toc47732285)

[1、主要任务 15](#_Toc47732286)

[2、算法原理 15](#_Toc47732287)

[3、实验数据测试 16](#_Toc47732288)

[实验八、174.删除链表中的倒数第n个节点 16](#_Toc47732289)

[1、主要任务 16](#_Toc47732290)

[2、算法原理 17](#_Toc47732291)

[3、实验数据测试 17](#_Toc47732292)

[实验九、372.在O(1)时间复杂度删除链表节点 18](#_Toc47732293)

[1、主要任务 18](#_Toc47732294)

[2、算法原理 18](#_Toc47732295)

[3、实验数据测试 19](#_Toc47732296)

[实验十、451.两两交换链表中的节点 19](#_Toc47732297)

[1、主要任务 19](#_Toc47732298)

[2、算法原理 20](#_Toc47732299)

[3、实验数据测试 20](#_Toc47732300)

[实验十一、452.删除链表中的元素 20](#_Toc47732301)

[1、主要任务 20](#_Toc47732302)

[2、算法原理 21](#_Toc47732303)

[3、实验数据测试 22](#_Toc47732304)

[实验十二、492.队列维护 22](#_Toc47732305)

[1、主要任务 22](#_Toc47732306)

[2、算法原理 23](#_Toc47732307)

[3、实验数据测试 23](#_Toc47732308)

[实验十三、756.两数相乘 24](#_Toc47732309)

[1、主要任务 24](#_Toc47732310)

[2、算法原理 24](#_Toc47732311)

[3、实验数据测试 25](#_Toc47732312)

[实验十四、822.相反的顺序存储 25](#_Toc47732313)

[1、主要任务 25](#_Toc47732314)

[2、算法原理 26](#_Toc47732315)

[3、实验数据测试 26](#_Toc47732316)

[实验十五、1609.链表的中间节点 26](#_Toc47732317)

[1、主要任务 26](#_Toc47732318)

[2、算法原理 27](#_Toc47732319)

[3、实验数据测试 27](#_Toc47732320)

[二、数组算法练习 28](#_Toc47732321)

[实验十六、14.二分查找 28](#_Toc47732322)

[1、主要任务 28](#_Toc47732323)

[2、算法原理 28](#_Toc47732324)

[3、实验数据测试 29](#_Toc47732325)

[实验十七、39.恢复旋转排序数组 29](#_Toc47732326)

[1、主要任务 29](#_Toc47732327)

[2、算法原理 30](#_Toc47732328)

[3、实验数据测试 30](#_Toc47732329)

[实验十八、56.两数之和 31](#_Toc47732330)

[1、主要任务 31](#_Toc47732331)

[2、算法原理 31](#_Toc47732332)

[3、实验数据测试 32](#_Toc47732333)

[实验十九、60.搜索插入位置 32](#_Toc47732334)

[1、主要任务 32](#_Toc47732335)

[2、算法原理 33](#_Toc47732336)

[3、实验数据测试 33](#_Toc47732337)

[实验二十、64.合并排序数组 33](#_Toc47732338)

[1、主要任务 33](#_Toc47732339)

[2、算法原理 34](#_Toc47732340)

[3、实验数据测试 35](#_Toc47732341)

[实验二十一、80.中位数 35](#_Toc47732342)

[1、主要任务 35](#_Toc47732343)

[2、算法原理 36](#_Toc47732344)

[3、实验数据测试 36](#_Toc47732345)

[实验二十二、100.删除排序数组中的重复数字 37](#_Toc47732346)

[1、主要任务 37](#_Toc47732347)

[2、算法原理 37](#_Toc47732348)

[3、实验数据测试 38](#_Toc47732349)

[实验二十三、101.删除排序数组中的重复数字Ⅱ 38](#_Toc47732350)

[1、主要任务 38](#_Toc47732351)

[2、算法原理 39](#_Toc47732352)

[3、实验数据测试 39](#_Toc47732353)

[实验二十四、114.不同的路径 40](#_Toc47732354)

[1、主要任务 40](#_Toc47732355)

[2、算法原理 40](#_Toc47732356)

[3、实验数据测试 41](#_Toc47732357)

[实验二十五、156.合并区间 41](#_Toc47732358)

[1、主要任务 41](#_Toc47732359)

[2、算法原理 42](#_Toc47732360)

[3、实验数据测试 42](#_Toc47732361)

[实验二十六、172.删除元素 42](#_Toc47732362)

[1、主要任务 42](#_Toc47732363)

[2、算法原理 43](#_Toc47732364)

[3、实验数据测试 43](#_Toc47732365)

[实验二十七、397.最长上升连续子序列 44](#_Toc47732366)

[1、主要任务 44](#_Toc47732367)

[2、算法原理 44](#_Toc47732368)

[3、实验数据测试 45](#_Toc47732369)

[实验二十八、495.实现栈 45](#_Toc47732370)

[1、主要任务 45](#_Toc47732371)

[2、算法原理 46](#_Toc47732372)

[3、实验数据测试 46](#_Toc47732373)

[实验二十九、822.相反的顺序存储 47](#_Toc47732374)

[1、主要任务 47](#_Toc47732375)

[2、算法原理 47](#_Toc47732376)

[3、实验数据测试 48](#_Toc47732377)

[实验三十、1157.最短无序连续子数组 48](#_Toc47732378)

[1、主要任务 48](#_Toc47732379)

[2、算法原理 49](#_Toc47732380)

[3、实验数据测试 49](#_Toc47732381)

[实验三十一、1250.第三大的数 49](#_Toc47732382)

[1、主要任务 49](#_Toc47732383)

[2、算法原理 50](#_Toc47732384)

[3、实验数据测试 51](#_Toc47732385)

[三、栈算法练习 51](#_Toc47732386)

[实验三十二、193.最长有效括号 51](#_Toc47732387)

[1、主要任务 51](#_Toc47732388)

[2、算法原理 52](#_Toc47732389)

[3、实验数据测试 52](#_Toc47732390)

[实验三十三、227.用栈模拟汉诺塔问题 53](#_Toc47732391)

[1、主要任务 53](#_Toc47732392)

[2、算法原理 53](#_Toc47732393)

[3、实验数据测试 54](#_Toc47732394)

[实验三十四、423.有效括号序列 54](#_Toc47732395)

[1、主要任务 54](#_Toc47732396)

[2、算法原理 55](#_Toc47732397)

[3、实验数据测试 55](#_Toc47732398)

[实验三十五、693.移掉k位数字 56](#_Toc47732399)

[1、主要任务 56](#_Toc47732400)

[2、算法原理 56](#_Toc47732401)

[3、实验数据测试 57](#_Toc47732402)

[实验三十六、834.移除多余字符 57](#_Toc47732403)

[1、主要任务 57](#_Toc47732404)

[2、算法原理 58](#_Toc47732405)

[3、实验数据测试 58](#_Toc47732406)

[实验三十七、983.棒球游戏 59](#_Toc47732407)

[1、主要任务 59](#_Toc47732408)

[2、算法原理 60](#_Toc47732409)

[3、实验数据测试 61](#_Toc47732410)

[实验三十八、1206.下一个更大的数Ⅰ 61](#_Toc47732411)

[1、主要任务 61](#_Toc47732412)

[2、算法原理 62](#_Toc47732413)

[3、实验数据测试 63](#_Toc47732414)

[四、队列算法练习 63](#_Toc47732415)

[实验三十九、492.队列维护 63](#_Toc47732416)

[1、主要任务 63](#_Toc47732417)

[2、算法原理 64](#_Toc47732418)

[3、实验数据测试 65](#_Toc47732419)

[实验四十、1206.下一个更大的数Ⅰ 65](#_Toc47732420)

[1、主要任务 65](#_Toc47732421)

[2、算法原理 66](#_Toc47732422)

[3、实验数据测试 66](#_Toc47732423)

[实验四十一、1166.推荐结果打散 67](#_Toc47732424)

[1、主要任务 67](#_Toc47732425)

[2、算法原理 68](#_Toc47732426)

[3、实验数据测试 68](#_Toc47732427)

# 一、链表算法练习

# 实验一、35.翻转链表

## 1、主要任务

翻转一个链表。

样例1：

输入: 1->2->3->null

输出: 3->2->1->null

样例 2:

输入: 1->2->3->4->null

输出: 4->3->2->1->null

## 2、算法原理

首先对链表进行判空，当链表的头结点为空时为空链表。当为空链表或者只有一个结点时直接返回头结点。之后定义三个指针：p指向头结点，q指向首元结点，r指向首元结点的下一个结点。在进行链表翻转时首先将头结点变为尾结点，即将头结点指向NULL，当r不为空时，反转指针，三个指针后移一步。当r为空时还要更改指向。

## 3、实验数据测试



# 实验二、96.链表划分

## 1、主要任务

给定一个单链表和数值x，划分链表使得所有小于x的节点排在大于等于x的节点之前。应该保留两部分内链表节点原有的相对顺序。

样例 1:

输入: list = null, x = 0

输出: null

样例解释: 空链表本身满足要求

样例 2:

输入: list = 1->4->3->2->5->2->null, x = 3

输出: 1->2->2->4->3->5->null

样例解释: 要保持原有的相对顺序。

## 2、算法原理

使用head指针从头到尾来遍历该链表的每一个元素，逐个与x进行比较，建立两个头结点，一个用来链接比x小的元素，一个用来链接比x大的元素最后串接起来并合并两个链表。

## 3、实验数据测试



# 实验三、112.删除排序链表中的重复元素

## 1、主要任务

给定一个排序链表，删除所有重复的元素，每个元素只留下一个。

样例 1:

输入: null

输出: null

样例 2:

输入: 1->1->2->null

输出: 1->2->null

样例 3:

输入: 1->1->2->3->3->null

输出: 1->2->3->null

## 2、算法原理

首先对链表进行判空当为空链表时，直接返回NULL。定义p指针指向头结点。当p指针的next域不为空即未到达链表尾部时如果p的数据域等于p的下一个结点的数据域，则改变p指针的next域，使p指针指向p的下一个结点从而实现去除一个重复的元素，其他情况p指针向后遍历。

## 3、实验数据测试



# 实验四、165.合并两个排序链表

## 1、主要任务

将两个排序链表合并为一个新的排序链表。

样例 1:

输入: list1 = null, list2 = 0->3->3->null

输出: 0->3->3->null

样例2:

输入: list1 = 1->3->8->11->15->null, list2 = 2->null

输出: 1->2->3->8->11->15->null

## 2、算法原理

首先初始化一个结点值为空的空结点，定义两个指针p,q指向节点值为空的空结点。p用于记录所创建链表的头结点的位置，q用于链接元素。

当链表l1,l2均不为空链表时如果l1的首元结点的数据域小于l2的首元结点的数据域，则q指针指向l1的这个元素，并且l1链表的l1指针向后遍历；否则q指针指向l2的这个元素，并且l2链表的l2指针向后遍历。直到两个链表的指针一个指向链表末尾为止。若此时l1指针不为空则q链接上l1的剩余元素否则则链接上l2的剩余元素。最终返回头结点的next域即返回了链表中的所有元素。

## 3、实验数据测试



# 实验五、166.链表倒数第n个结点

## 1、主要任务

找到单链表的倒数第n个结点，保证链表中的结点的最少数量为n

样例 1:

输入: list = 3->2->1->5->null, n = 2

输出: 1

样例 2:

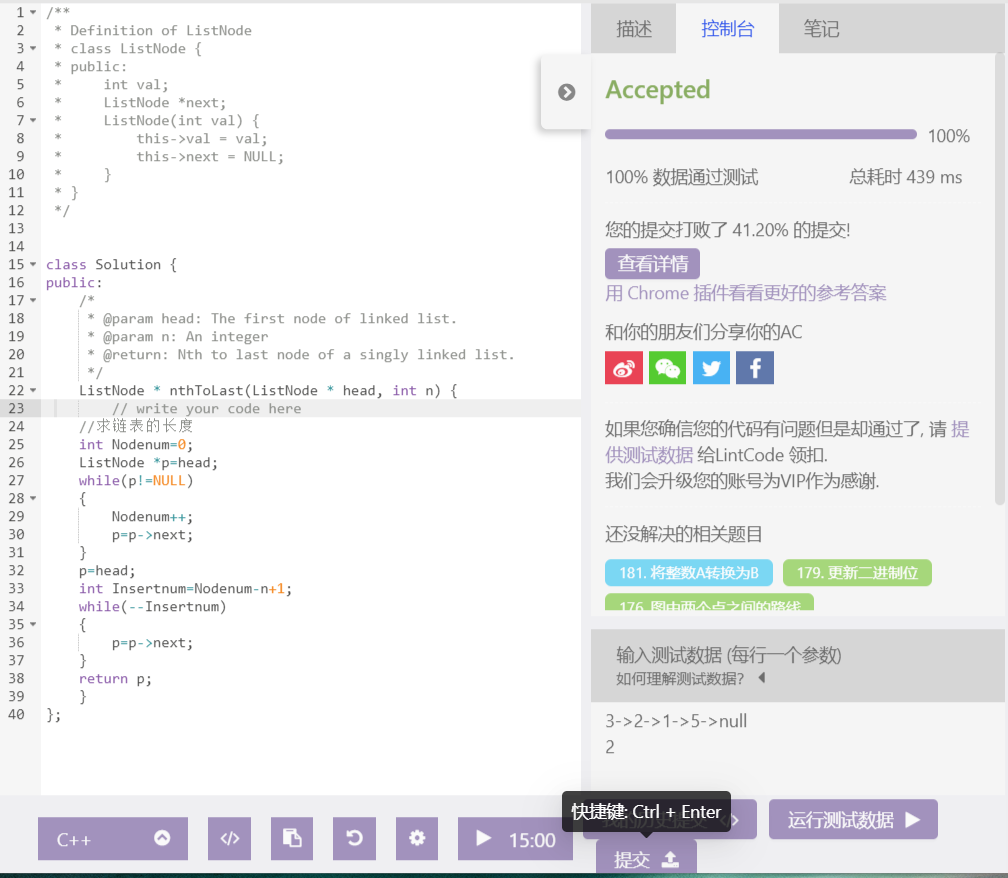
输入: list = 1->2->3->null, n = 3

输出: 1

## 2、算法原理

首先定义Nodenum用于记录链表的长度，定义p指针指向链表的头结点。当p不为空即链表不为空链表时，Nodenum加一且向后遍历计算出链表长度。此时p指针指向链表末尾。使p指针再次指向头结点。由于是倒数第n个结点，插入位置为Nodenum-n+1，进行遍历找到链表中此位置进行插入。最后返回此节点。

## 3、实验数据测试



# 实验六、167.链表求和

## 1、主要任务

有两个用链表代表的整数，其中每个节点包含一个数字。数字存储按照在原来整数中相反的顺序，使得第一个数字位于链表的开头。写出一个函数将两个整数相加，用链表形式返回和。

样例 1:

输入: 7->1->6->null, 5->9->2->null

输出: 2->1->9->null

样例解释: 617 + 295 = 912, 912 转换成链表: 2->1->9->null

样例 2:

输入: 3->1->5->null, 5->9->2->null

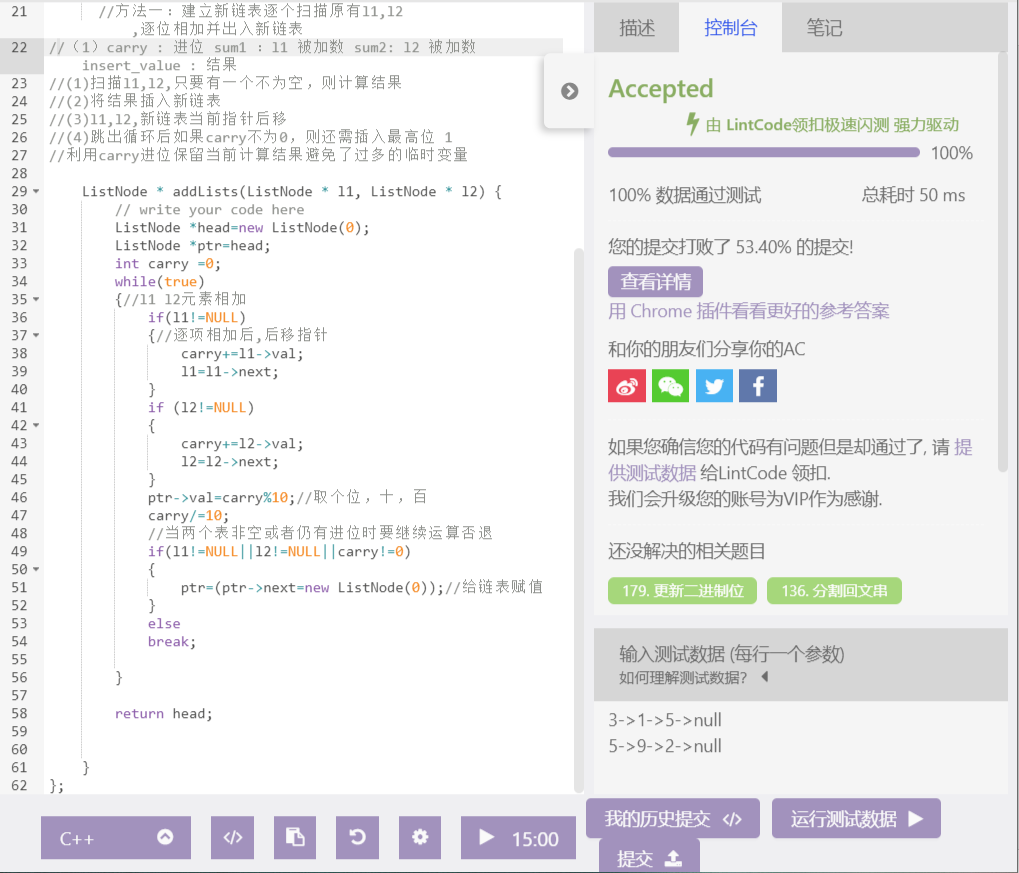
输出: 8->0->8->null

样例解释: 513 + 295 = 808, 808 转换成链表: 8->0->8->null

## 2、算法原理

首先初始化一个结点值为空的头结点，定义指针ptr指向头结点。定义carry用于记录加和结果。首先carry记录l1和l2的最后一个元素，对carry取余10即得到了个位，随后对carry整除10后carry为十位进位.此时两个指针非空或carry不为0时继续相加l1,l2链表的第二个元素的值，保存在carry中，对carry取余10得到十位保存在下一个结点中。并且重复此操作直到最后两指针均指向链表尾且无进位此时跳出循环，返回链表。

## 3、实验数据测试



# 实验七、173.链表插入排序

## 1、主要任务

用插入排序对链表排序。

样例 1:

输入: 0->null

输出: 0->null

样例 2:

输入: 1->3->2->0->null

输出 :0->1->2->3->null

## 2、算法原理

首先初始化一个值为空的空结点并令指针p指向它，当链表不为空时，定义指针node和指针p指向相同。当node指针的next域不为空同时node的下一个结点的值大于原链表头结点的值时，node指针后移。定义q指针指向原链表的首元结点，将node的next域赋给head的next域后再将head赋给node的next域，q赋给head。最后返回链表。

## 3、实验数据测试



# 实验八、174.删除链表中的倒数第n个节点

## 1、主要任务

给定一个链表，删除链表中倒数第n个结点，返回链表的头结点。

（链表中的结点数大于等于n）

样例 1:

输入: list = 1->2->3->4->5->null， n = 2

输出: 1->2->3->5->null

样例 2:

输入: list = 5->4->3->2->1->null, n = 2

输出: 5->4->3->1->null

## 2、算法原理

使用两个指针进行扫描，当第一个指针扫描到第n个结点后，第二个指针从表头与第一个指针同时向后移动。

当第一个指针指向空结点时，第二个指针就指向倒数第n个结点。

## 3、实验数据测试



# 实验九、372.在O(1)时间复杂度删除链表节点

## 1、主要任务

给定一个单链表中的一个等待被删除的结点（非表头或表尾）。在O(1)的时间复杂度删除该链表节点。

样例 1：

输入：1->2->3->4->null

3

输出：1->2->4->null

样例 2：

输入：1->3->5->null

3

输出：1->5->null

## 2、算法原理

首先对链表进行判空。当链表的头结点node为空时，直接返回。当node的next域为空时，释放node指针，node赋值为空后返回。

其他情况定义p指针指向node的首元结点。将p指针结点的数值域赋给node指针结点的数值域。p的next域赋给node的next域后释放p指针。

## 3、实验数据测试



# 实验十、451.两两交换链表中的节点

## 1、主要任务

给一个链表，两两交换其中的结点，然后返回交换后的链表。

样例 1：

输入：1->2->3->4->null

输出：2->1->4->3->null

样例 2：

输入：5->null

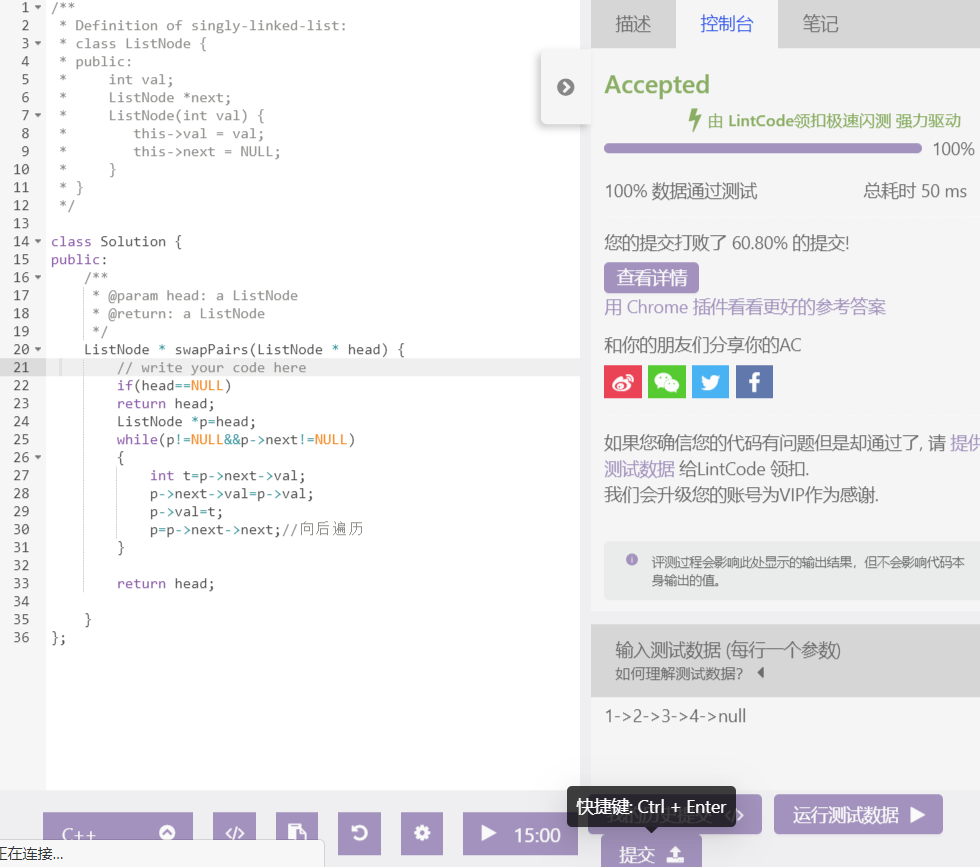
输出：5->null

## 2、算法原理

首先对链表进行判空。若为空链表则直接返回此链表。

否则定义p指针和head指针指向相同。当p指针不为空，同时p的next域不为空时，定义t保存p的下一个节点的数值域，p的数值域赋给p的下一个节点的数值域，将t赋给p的数值域（即将p的下一个节点的数值域赋给p的数值域），p指针向后遍历。重复直到p指针指向末尾返回链表。

## 3、实验数据测试



# 实验十一、452.删除链表中的元素

## 1、主要任务

删除链表中等于给定值val的所有结点。

样例 1：

输入：head = 1->2->3->3->4->5->3->null, val = 3

输出：1->2->4->5->null

样例 2：

输入：head = 1->1->null, val = 1

输出：null

## 2、算法原理

首先对链表进行判空，空链表则直接返回。当链表不为空同时头结点的数值域等于给定值时，定义p指针和head指向相同，head指针后移，释放p指针，若head指针指向末尾或head指向的数值域不为给定值。定义pre指针指向与head相同。当pre指针不为空且pre的next域不为空时，p指针指向pre的下一个节点，若p的数值域等于给定值，将p的next域赋值给pre的next域同时释放p，否则，p赋给pre。最终返回链表。

## 3、实验数据测试



# 实验十二、492.队列维护

## 1、主要任务

实现一个队列的操作：

enqueue(item).将新元素放入队列中。

dequeue(). 将第一个元素移出队列，返回它。

例1:

输入：

enqueue(1)

enqueue(2)

enqueue(3)

dequeue() // return 1

enqueue(4)

dequeue() // return 2

例2:

输入：

enqueue(10)

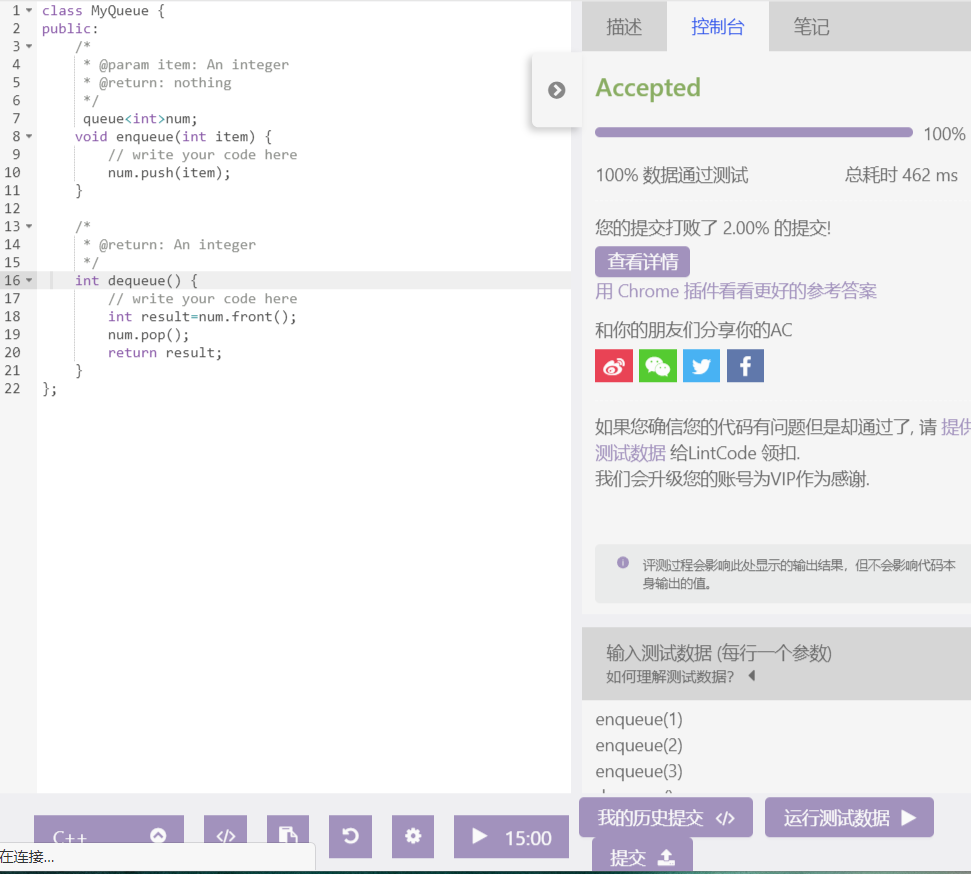
dequeue()// return 10

## 2、算法原理

对于enqueue()操作使用队列的push()方法来实现。

对于dequeue()操作定义result保存num队列的头指针指向的元素。使用队列的pop()方法来实现dequeue()。返回result即返回了出列的元素。

## 3、实验数据测试



# 实验十三、756.两数相乘

## 1、主要任务

给出两个链表形式表示的数字，写一个函数得到这两个链表相乘乘积。

样例 1:

输入：9->4->6->null,8->4->null

输出：79464

解释：946\*84=79464

样例 2:

输入：3->2->1->null,1->2->null

输出：3852

解释：321\*12=3852

## 2、算法原理

定义sum1和sum2分别记录两个链表的结果，当l1不为空时，sum1为l1的数据域，l1指针向后遍历，sum1乘十用于记录第一位，sum2同理。最终用sum1乘以sum2得到最终结果。

## 3、实验数据测试



# 实验十四、822.相反的顺序存储

## 1、主要任务

给出一个链表，并将链表的值以倒序存储到数组中。不能改变原始链表的结构，ListNode 有两个成员变量：ListNode.val 和 ListNode.next。

样例1

输入: 1 -> 2 -> 3 -> null

输出: [3,2,1]

样例2

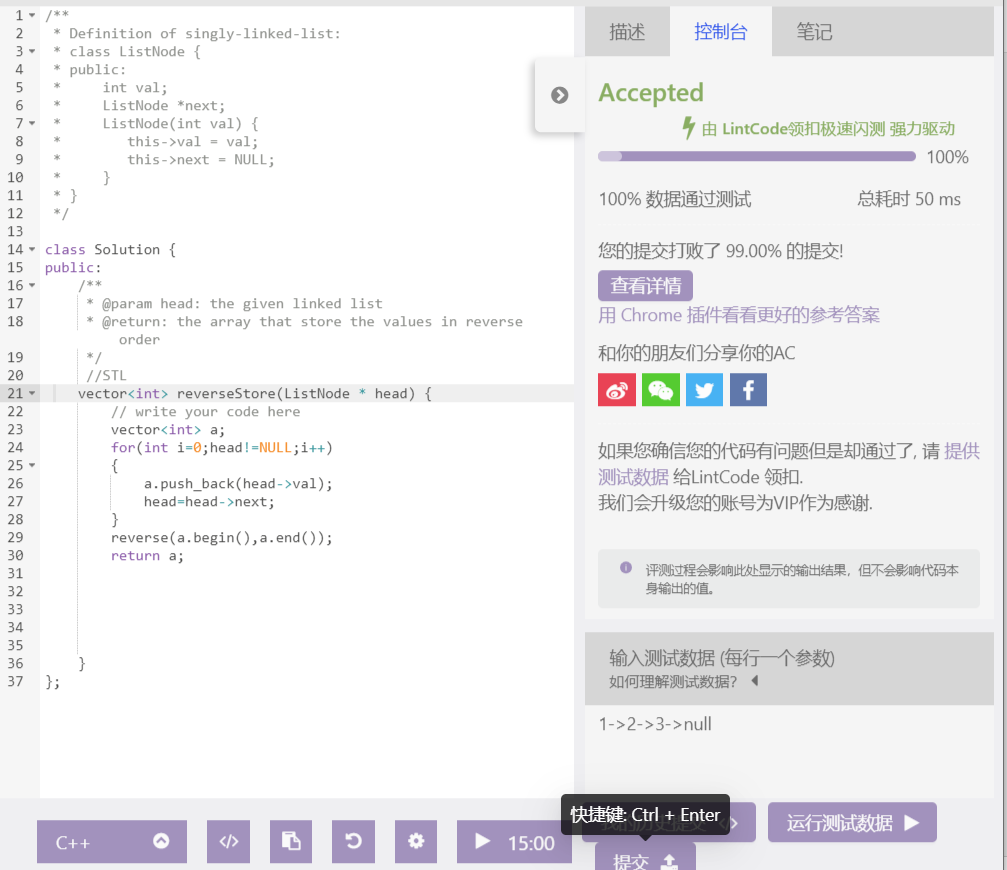
输入: 4 -> 2 -> 1 -> null

输出: [1,2,4]

## 2、算法原理

首先定义数组a，遍历链表，调用push\_back()方法将数据域的值均逐个读入数组。最后调用reverse()方法进行倒序排列。

## 3、实验数据测试



# 实验十五、1609.链表的中间节点

## 1、主要任务

给定一个带有头结点head的非空单链表，返回链表的中间节点。若有两个中间节点，则返回第二个中间结点。给定链表的结点数在1-100之间。

样例 1:

输入：1->2->3->4->5->null

输出：3->4->5->null

样例 2:

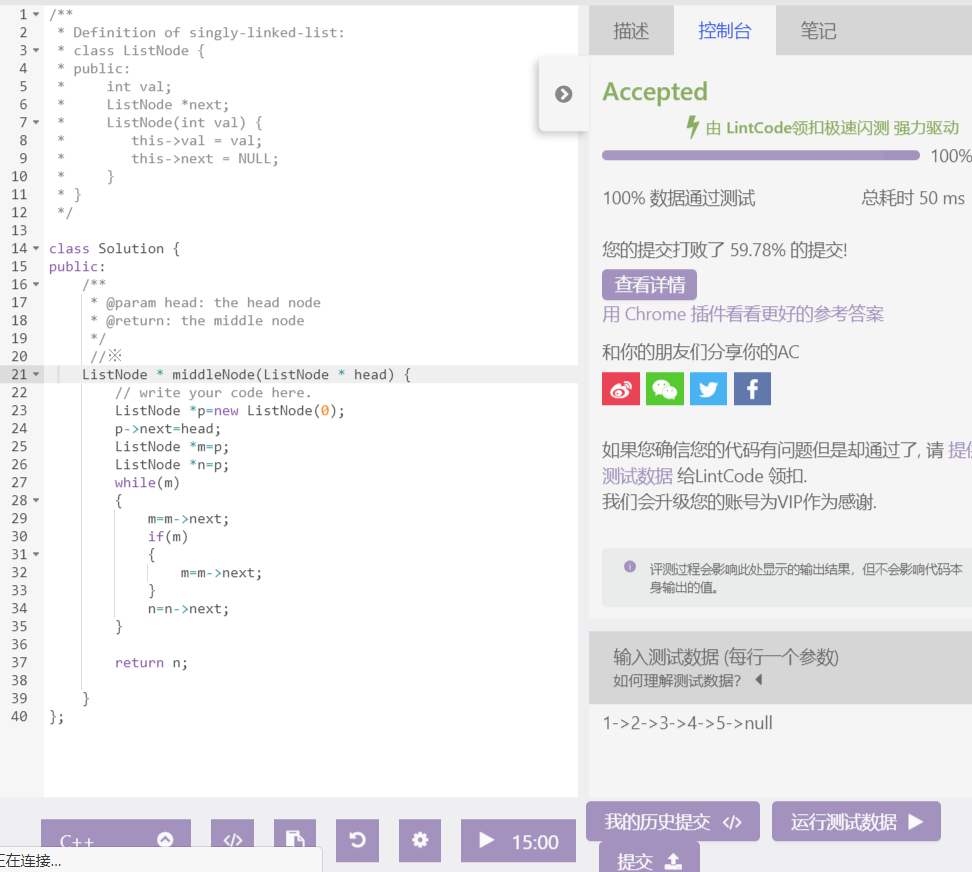
输入：1->2->3->4->5->6->null

输出：4->5->6->null

## 2、算法原理

首先初始化一个结点值为空的空结点并用p指向它，p的next域赋值为head，定义m,n指向与p相同，当m不为空时，向后遍历，再进行m是否为空的判断，最终返回n。

## 3、实验数据测试



# 二、数组算法练习

# 实验十六、14.二分查找

## 1、主要任务

给定一个排序的整数数组（升序）和一个要查找的整数target，用O(logn)的时间查找到target第一次出现的下标（从0开始），如果target不存在于数组中，返回-1。

样例 1:

输入:[1,4,4,5,7,7,8,9,9,10]，1

输出: 0

样例解释:

第一次出现在第0个位置。

样例 2:

输入: [1, 2, 3, 3, 4, 5, 10]，3

输出: 2

样例解释:

第一次出现在第2个位置

样例 3:

输入: [1, 2, 3, 3, 4, 5, 10]，6

输出: -1

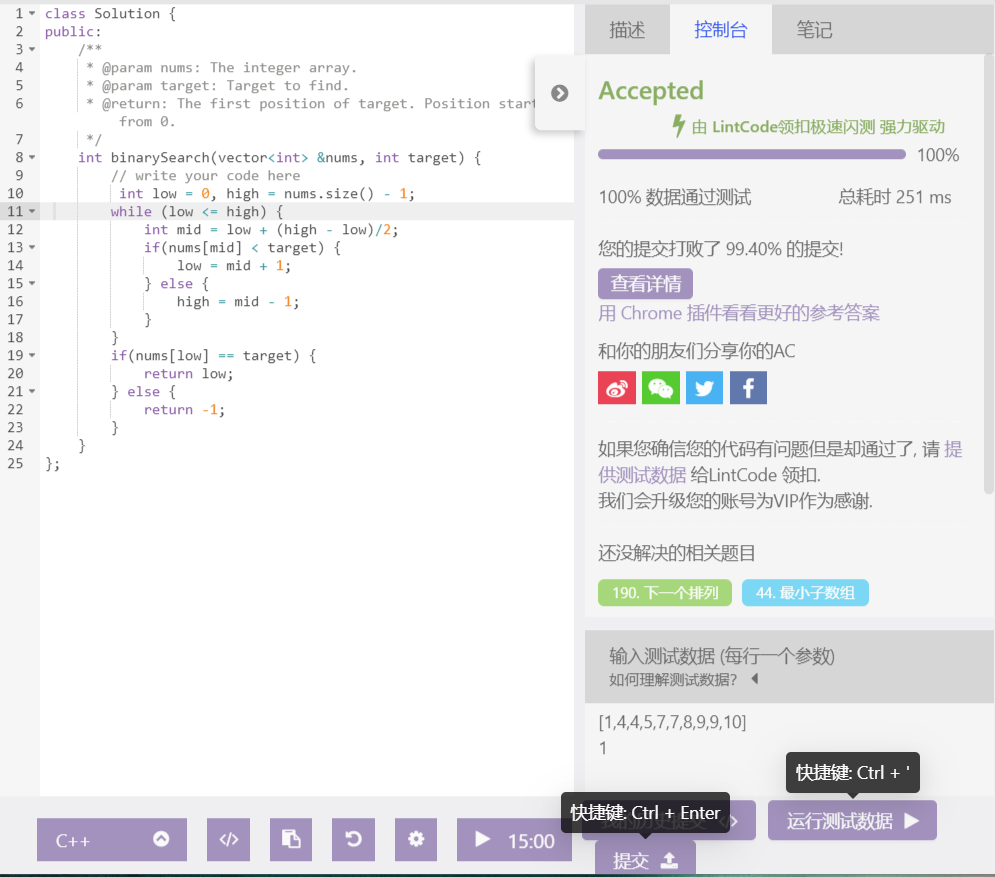
样例解释:

没有出现过6， 返回-1

## 2、算法原理

首先定义low和high来记录下标的首地址和数组的最后一个元素的地址。由此计算mid，当数组的下标mid的元素小于给定值时，low为mid+1,否则high为mid-1。如果数组下标为low的元素等于给定值时返回low，否则即找不到，返回-1。

## 3、实验数据测试



# 实验十七、39.恢复旋转排序数组

## 1、主要任务

给定一个旋转排序数组，在原地恢复其排序。（升序）

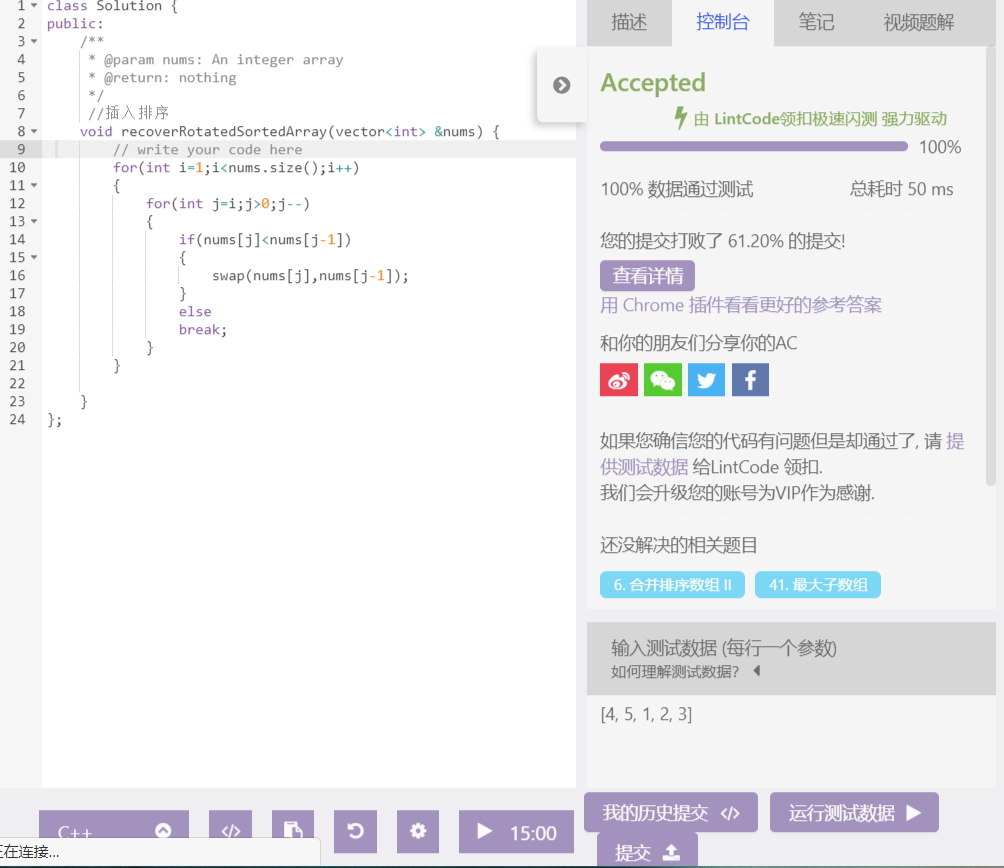
旋转数组：比如，原始数组为[1,2,3,4], 则其旋转数组可以是[1,2,3,4], [2,3,4,1], [3,4,1,2], [4,1,2,3]。

样例1:  
[4, 5, 1, 2, 3] -> [1, 2, 3, 4, 5]  
样例2:  
[6,8,9,1,2] -> [1,2,6,8,9]

## 2、算法原理

利用插入排序进行对其恢复排序。调用swap()函数来恢复排序。

## 3、实验数据测试



# 实验十八、56.两数之和

## 1、主要任务

给一个整数数组，找到两个数使得他们的和等于一个给定的数target。需要实现的函数twosum需要返回这两个数的下标，并且第一个下标小于第二个下标。这里下标的范围是0到n-1。

样例1:

给出 numbers = [2, 7, 11, 15], target = 9, 返回 [0, 1].

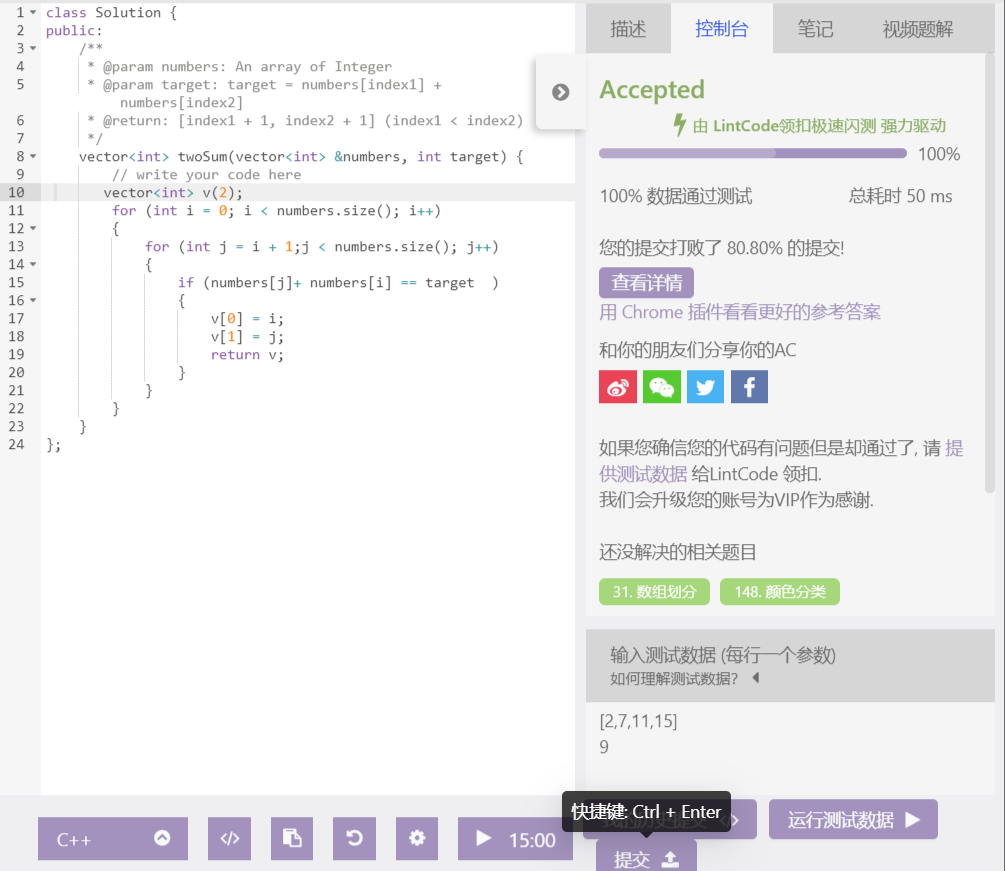
样例2:

给出 numbers = [15, 2, 7, 11], target = 9, 返回 [1, 2].

## 2、算法原理

首先定义一个含有两个元素的数组。定义两个指针分别遍历，当numbers[j]+ numbers[i] == target时，数组的第一个元素赋值为i，数组的第二个元素赋值为j。最后返回此数组

## 3、实验数据测试



# 实验十九、60.搜索插入位置

## 1、主要任务

给定一个排序数组和一个目标值，如果在数组中找到目标值则返回索引。如果没有，返回到它将会被按顺序插入的位置。在数组中没有重复元素。

样例

[1,3,5,6]，5 → 2

[1,3,5,6]，2 → 1

[1,3,5,6]， 7 → 4

[1,3,5,6]，0 → 0

## 2、算法原理

首先对数组判空，若为空则返回0，若数组的第一个元素大于或者等于给定值则返回0。如果数组的最后一个元素小于给定值，返回的下表为数组长度。遍历数组，如果下标为i的值小于给定值，下标i+1的值大于或者等于给定值，则返回i+1。

## 3、实验数据测试



# 实验二十、64.合并排序数组

## 1、主要任务

合并两个排序的整数数组A和B变成一个新的数组。

A具有足够的空间（A数组的大小大于或等于m+n）去添加B中的元素。

样例 1:

输入：[1, 2, 3] 3 [4,5] 2

输出：[1,2,3,4,5]

解释:

经过合并新的数组为[1,2,3,4,5]

样例 2:

输入：[1,2,5] 3 [3,4] 2

输出：[1,2,3,4,5]

解释：

经过合并新的数组为[1,2,3,4,5]

## 2、算法原理

将B数组放入到A数组里面后，调用sort()函数对数组进行排序。即为先合并后排序。

## 3、实验数据测试



# 实验二十一、80.中位数

## 1、主要任务

给定一个未排序的整数数组，找到其中位数。中位数是排序后数组的中间值，如果数组的个数是偶数个，则返回排序后数组的第N/2个数。数组大小不超过10000。

样例 1:

输入：[4, 5, 1, 2, 3]

输出：3

解释：

经过排序，得到数组[1,2,3,4,5]，中间数字为3

样例 2:

输入：[7, 9, 4, 5]

输出：5

解释：

经过排序，得到数组[4,5,7,9]，第二个(4/2)数字为5

## 2、算法原理

首先定义用n来记录数组的长度，调用sort()函数来进行排序。最后返回数组的下标为(n-1)/2的元素。

## 3、实验数据测试



# 实验二十二、100.删除排序数组中的重复数字

## 1、主要任务

给定一个排序数组，在原数组中“删除”重复出现的数字，是的每个元素只出现一次，并且返回“新”数组的长度。

不要使用额外的数组空间，必须在不使用额外空间的条件下原地完成。

样例 1:

输入: []

输出: 0

样例 2:

输入: [1,1,2]

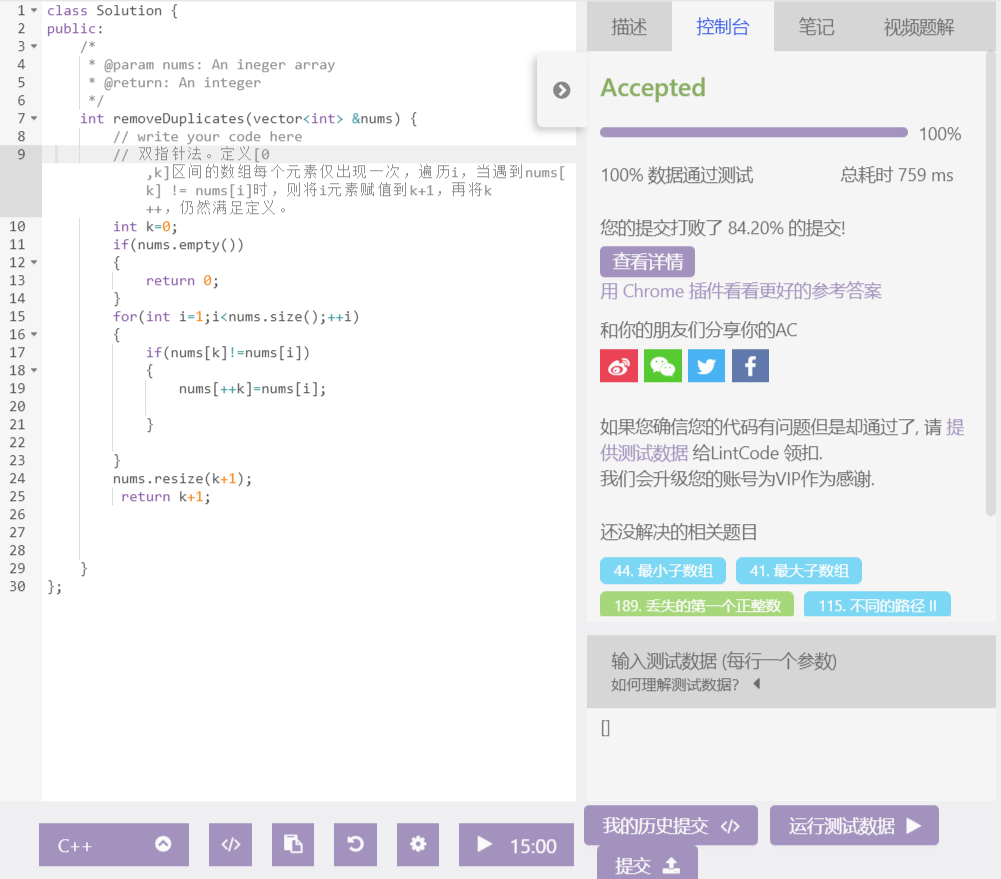
输出: 2

解释: 数字只出现一次的数组为: [1,2]

## 2、算法原理

采用双指针法，定义[0,k]区间的数组每个元素仅出现一次，遍历i，当遇到nums[k]!=nums[i]时，则将i元素赋值到k+1，再将k++，仍然满足定义。

## 3、实验数据测试



# 实验二十三、101.删除排序数组中的重复数字Ⅱ

## 1、主要任务

给定一个排序数组，删除其中的重复元素，使得每个数字最多出现两次，返回新的数组的长度。如果一个数字出现超过两次，则这个数字最后保留两个。同时需要在原数组中操作。

样例 1:

输入: []

输出: 0

样例 2:

输入: [1,1,1,2,2,3]

输出: 5

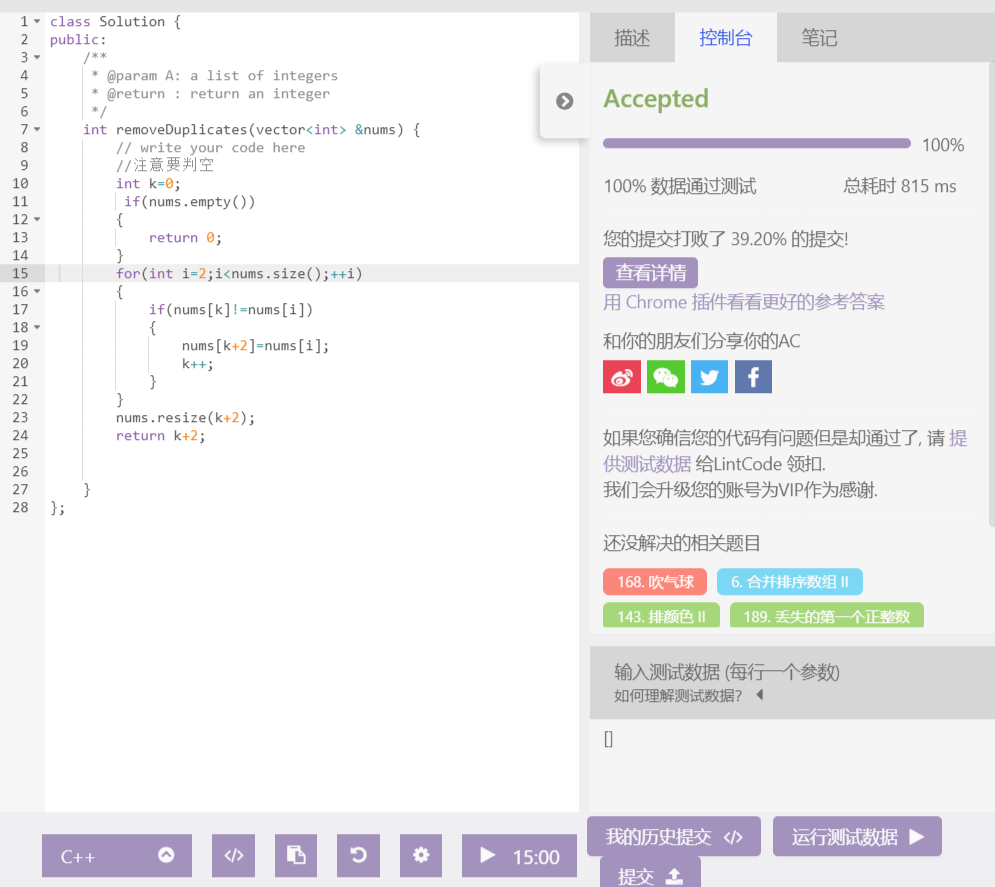
样例解释:

长度为 5， 数组为：[1,1,2,2,3]

## 2、算法原理

首先对数组进行判空，若数组为空数组则直接返回。否则进行判断，注意如果一个数字出现超过2次则这个数字最后保留两个。

## 3、实验数据测试



# 实验二十四、114.不同的路径

## 1、主要任务

有一个机器人的位于一个 m × n 个网格左上角。机器人每一时刻只能向下或者向右移动一步。机器人试图达到网格的右下角。问有多少不同的路径。

N和m都不超过100，且答案保证在32位整数可表示范围内。

样例 1:

输入: n = 1, m = 3

输出: 1

样例解释: Only one path to target position.

样例 2:

输入: n = 3, m = 3

输出: 6

样例解释:

D : Down

R : Right

1) DDRR

2) DRDR

3) DRRD

4) RRDD

5) RDRD

6) RDDR

## 2、算法原理

由于只能在向右或向下移动，则最上方一行与最左边一列的值为1，并且当前位置的位数则是上边格子的步数与左边各自的步数之和。

## 3、实验数据测试



# 实验二十五、156.合并区间

## 1、主要任务

给出若干闭合区间，合并所有重叠部分。

样例1:

输入: [(1,3)]

输出: [(1,3)]

样例 2:

输入: [(1,3),(2,6),(8,10),(15,18)]

输出: [(1,6),(8,10),(15,18)]

## 2、算法原理

首先自定义比较函数，用size来记录数组的长度。若数组的长度小于等于0时，返回空数组。定义一个数组result，调用sort()函数，排序条件为自定义的比较函数，逐个排序，最终返回数组。

## 3、实验数据测试



# 实验二十六、172.删除元素

## 1、主要任务

给定一个数组和一个值，在原地删除与值相同的数字，返回新数组的长度。元素的顺序可以改变，并且对新的数组不会有影响。

样例 1:

输入: [], value = 0

输出: 0

样例 2:

输入: [0,4,4,0,0,2,4,4], value = 4

输出: 4

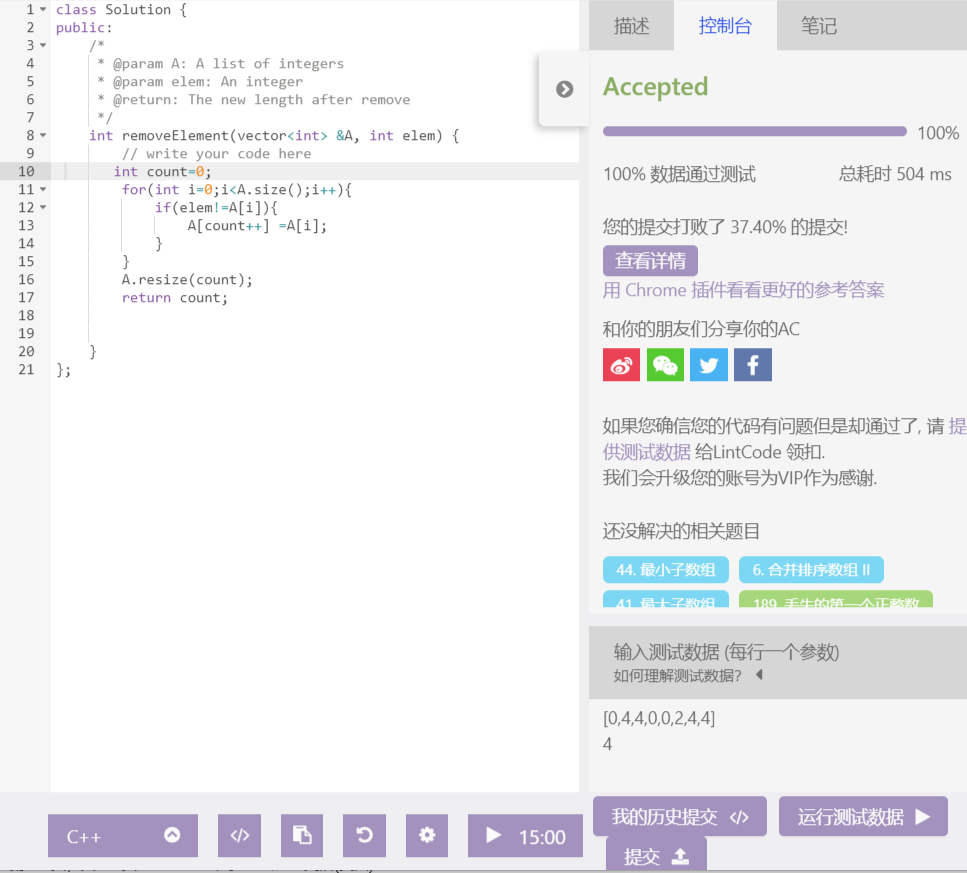
样例解释:

删除后的数组为 [0,0,0,2]

## 2、算法原理

首先定义count作为计数器。遍历A数组，如果数组的元素不等于给定值，则存入数组，最后重新分配数组的空间，返回计数器。

## 3、实验数据测试



# 实验二十七、397.最长上升连续子序列

## 1、主要任务

给定一个整数数组，（下标为从0到n-1,n表示整个数组的规模），找出该数组中的最长上升连续子序列。（最长上升连续子序列可以定义为从右到左或从左到右的序列）

样例 1：

输入：[5, 4, 2, 1, 3]

输出：4

解释：

给定 [5, 4, 2, 1, 3]，其最长上升连续子序列（LICS）为 [5, 4, 2, 1]，返回 4。

样例 2：

输入：[5, 1, 2, 3, 4]

输出：4

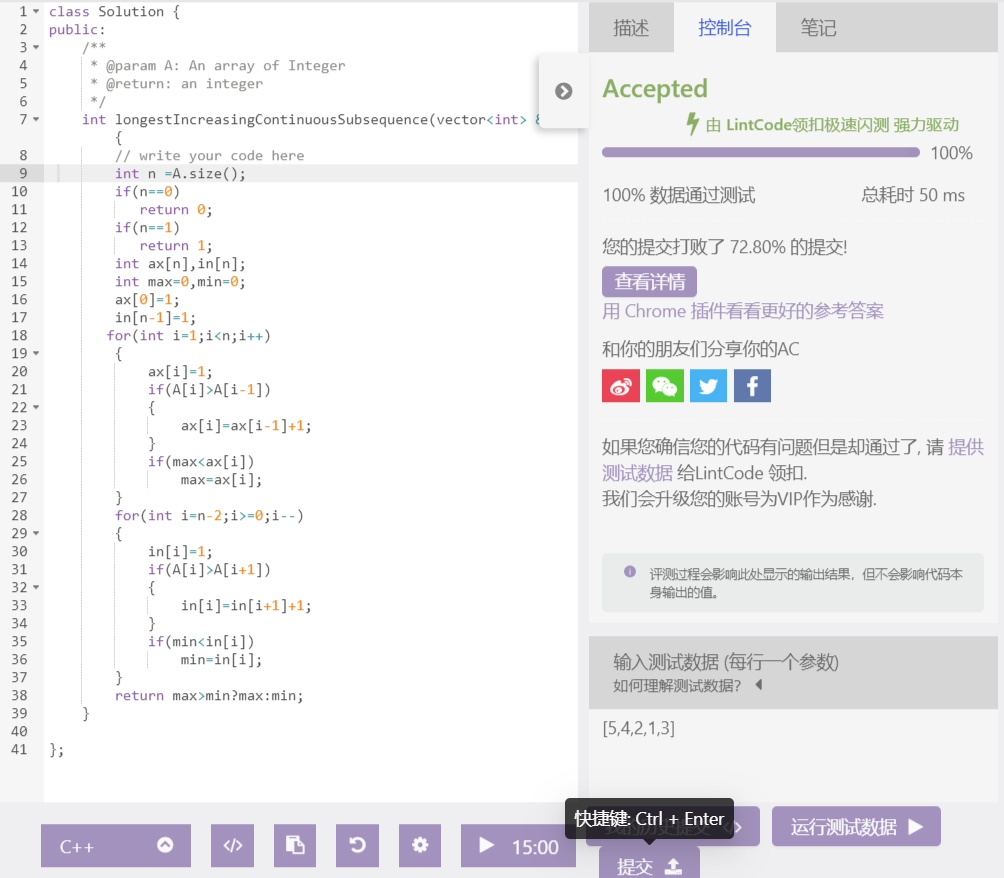
解释：

给定 [5, 1, 2, 3, 4]，其最长上升连续子序列（LICS）为 [1, 2, 3, 4]，返回 4。

## 2、算法原理

当数组长度为0时直接返回0，当数组长度为1时直接返回1。否则通过遍历和比较来得到最长上升连续子序列。

## 3、实验数据测试



# 实验二十八、495.实现栈

## 1、主要任务

实现一个栈，可以实现除了栈之外的数据结构。

例1:

输入：

push(1)

pop()

push(2)

top() // return 2

pop()

isEmpty() // return true

push(3)

isEmpty() // return false

例2:

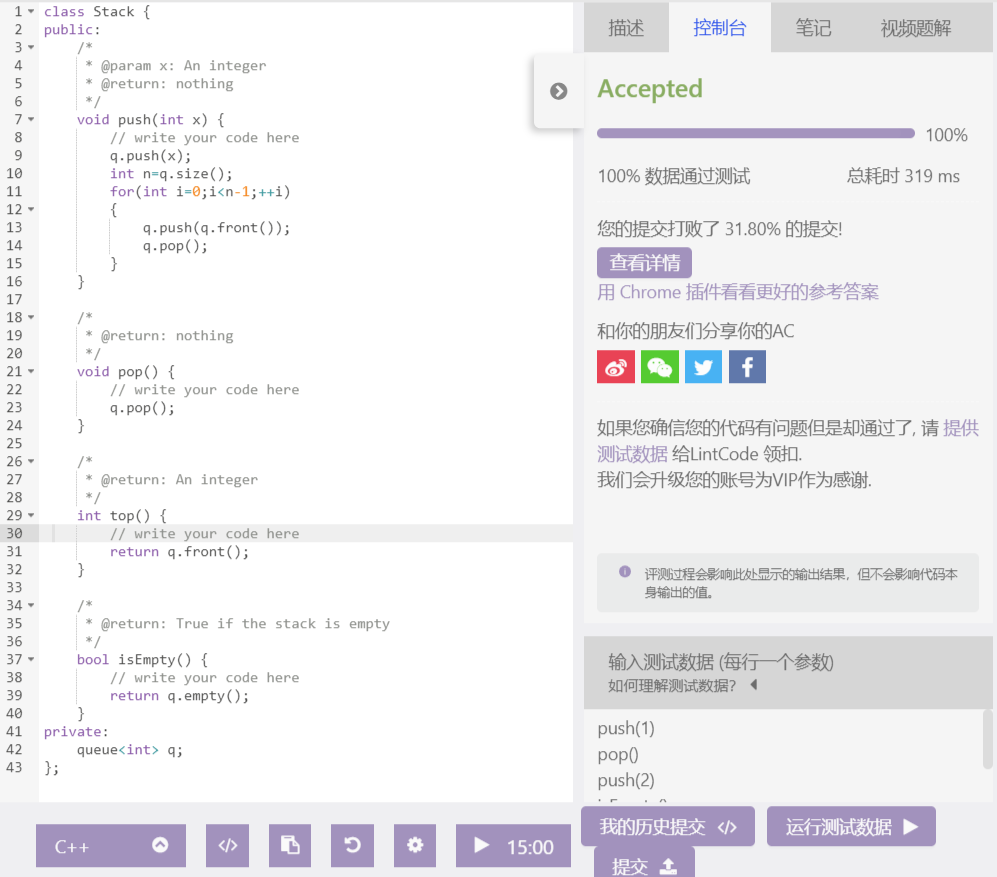
输入：

isEmpty()

## 2、算法原理

对于push()操作（即出栈），调用队列的push()函数，之后调用pop()函数，遍历队列，实现出栈。Pop（）方法通过调用队列的pop()实现。Top()函数通过调用队列的front()函数来返回头指针对应的元素。isEmpty()函数通过调用队列的empty()来实现。

## 3、实验数据测试



# 实验二十九、822.相反的顺序存储

## 1、主要任务

给出一个链表，并将链表的值以倒叙存储到数组中。不能改变原始链表的结构。ListNode有两个成员变量：ListNode.val和ListNode.next。

样例1

输入: 1 -> 2 -> 3 -> null

输出: [3,2,1]

样例2

输入: 4 -> 2 -> 1 -> null

输出: [1,2,4]

## 2、算法原理

首先定义a数组，遍历链表通过push\_back()函数存储到数组中，继续后移指针。最后调用reverse()函数实现倒序，返回数组。

## 3、实验数据测试



# 实验三十、1157.最短无序连续子数组

## 1、主要任务

给定一个整数数组，需要找到一个连续子数组。如果只按升序对这个子数组进行排序，那么整个数组也将按升序排序。找到最短的子数组并输出它的长度。输出的数组长度范围为[1,10,000]，输入的数组可能会包含重复数组，本题升序的含义为<=。

输入: [2, 6, 4, 8, 10, 9, 15]

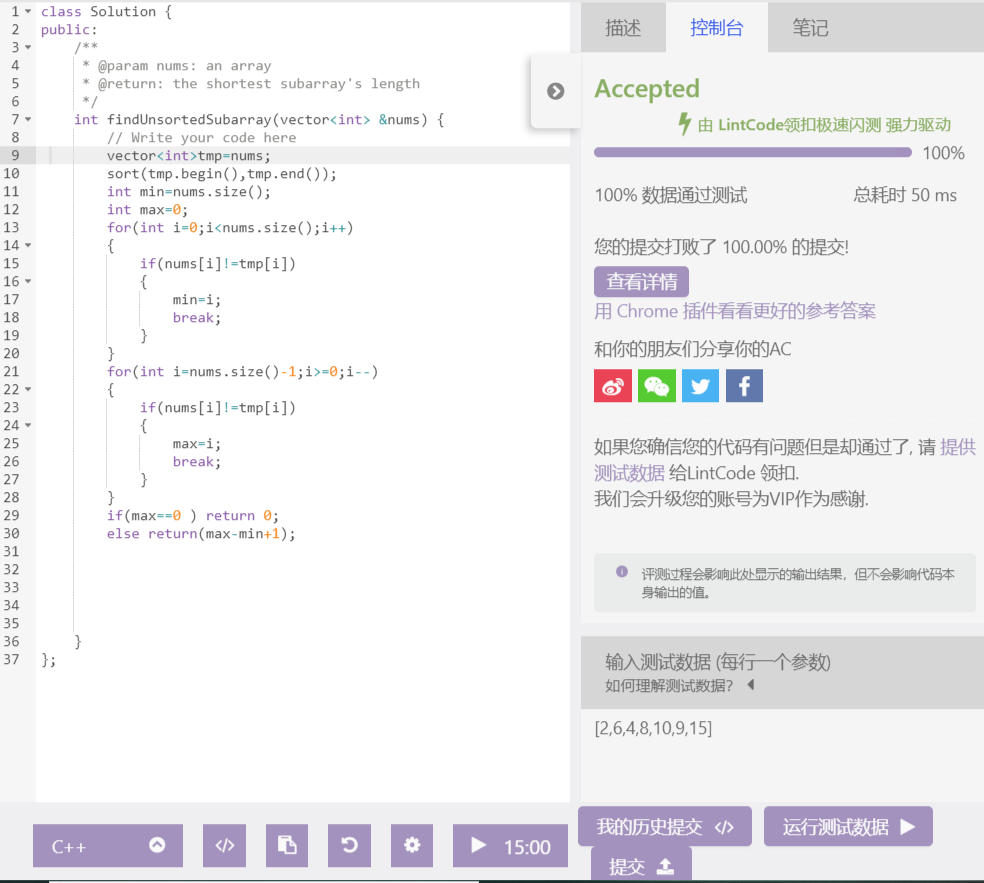
输出: 5

解释: 你需要对[6, 4, 8, 10, 9]按升序排列从而整个数组也变为升序。

## 2、算法原理

首先定义tmp数组，调用sort()函数对数组进行排序。Min初始化为数组长度，定义max。遍历nums数组与排序后的tmp数组比较，min记录无序数组的第一个元素下标，max记录无序数组最后一个元素下标。计算得到子数组的长度。

## 3、实验数据测试



# 实验三十一、1250.第三大的数

## 1、主要任务

给定一个非空数组，返回此数组中第三大的数，如果不存在，则返回数组中最大的数。要求算法的时间复杂度必须是O(n)。

样例 1:

输入: num = [3, 2, 1]

输出: 1

解释: 第三大的数是 1.

样例 2:

输入: num = [1, 2]

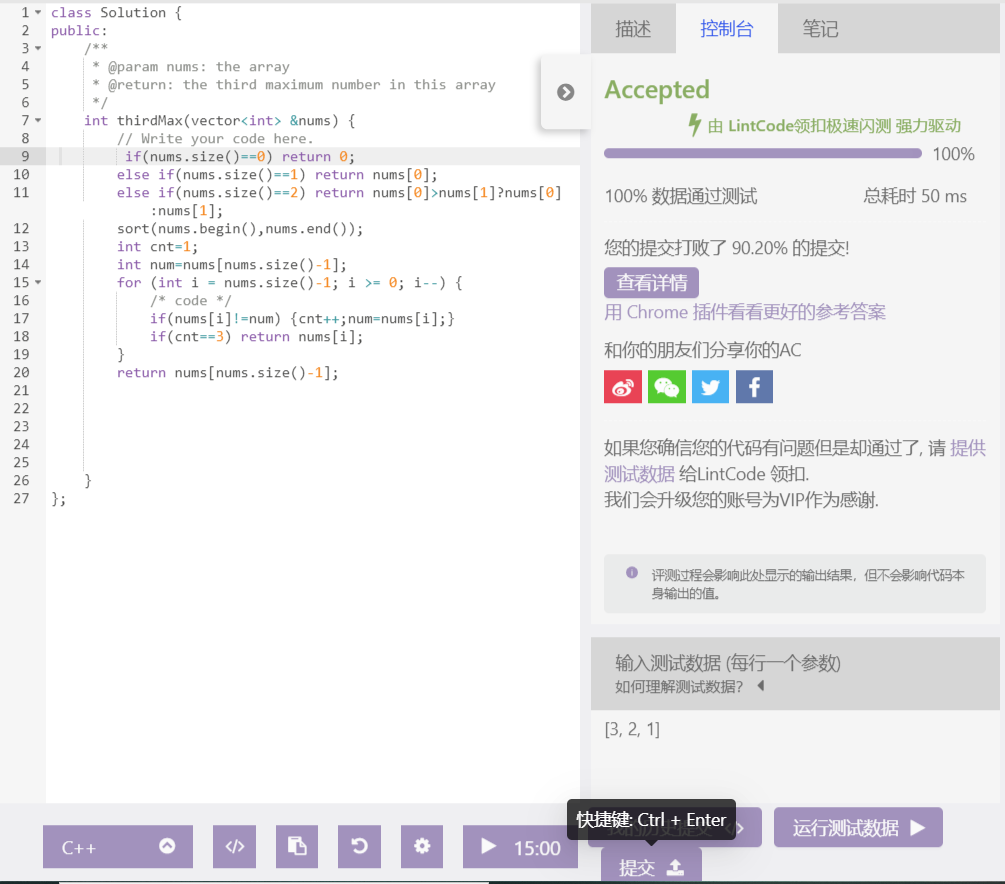
输出: 2

解释: 第三大的数不存在, 所以返回最大的数 2 .

## 2、算法原理

首先对数组进行判空，若为空数组，则直接返回0，若数组长度为1，直接返回数组的第一个元素，如果数组的长度为2，后来返回数组的两个元素中的较大的那个。其他情况，首先调用sort()函数对数组元素排序。定义计数器cnt初始化为1，定义num为数组的最后一个元素，从后向前遍历数组，当数组元素不等于num时，计数器累加，num赋值为此元素，当计数器为3时，返回此元素。其他最后返回nums数组中的最后一个元素。

## 3、实验数据测试



# 三、栈算法练习

# 实验三十二、193.最长有效括号

## 1、主要任务

给出一个只包含'(' 和')'的字符串，找出其中最长的左右括号正确匹配的合法子串。

样例 1:

输入: "(()"

输出: 2

解释: 最长有效括号子串为 "()"

样例 2:

输入: ")()())"

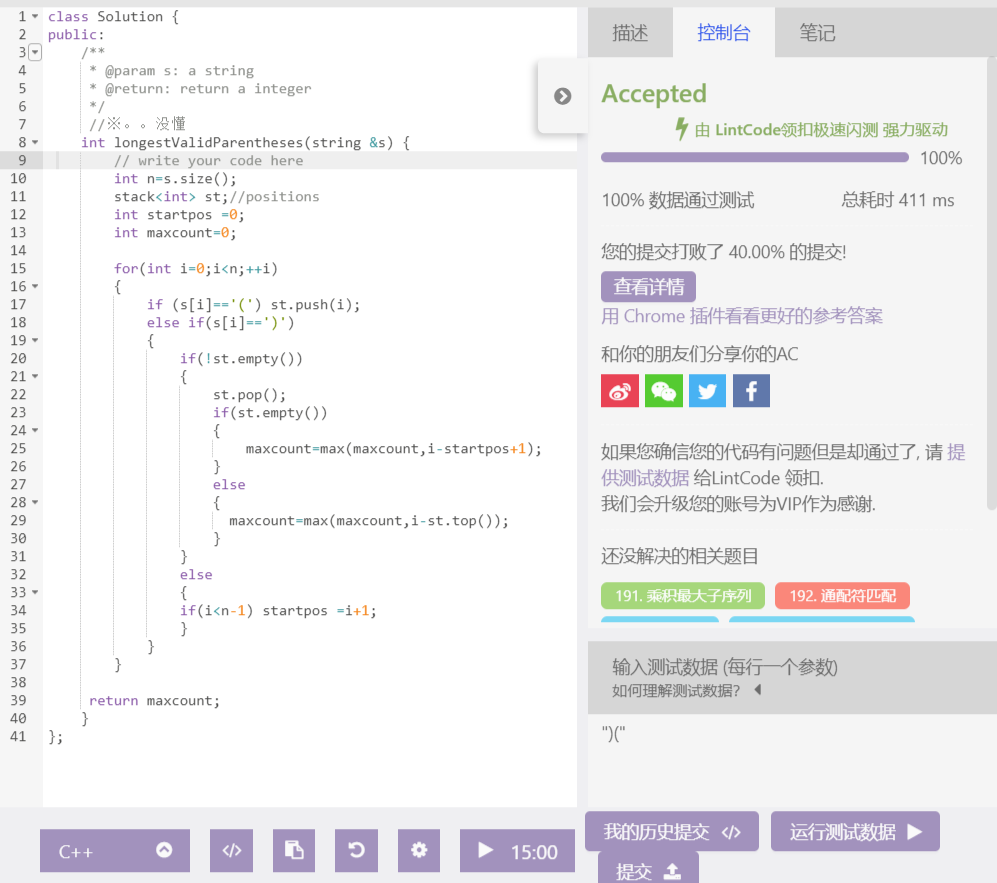
输出: 4

解释: 最长有效括号子串为 "()()"

## 2、算法原理

首先计算字符串的长度，定义栈st，定义开始位置startpos和最大计数器maxcount。遍历字符串，当为“(”时堆栈进入栈中，当为“)”时，若不为空栈，调用pop()函数弹出栈顶元素。此后若为空栈则最长有效括号的长度为maxcount和i-startpos+1中较大的那个，否则为maxcount和i-st.top()中较大的那个。若为空栈，若i<n-1，startpos为i+1。最后返回maxcount。

## 3、实验数据测试



# 实验三十三、227.用栈模拟汉诺塔问题

## 1、主要任务

在经典的汉诺塔问题中，有 3 个塔和 N 个可用来堆砌成塔的不同大小的盘子。要求盘子必须按照从小到大的顺序从上往下堆 （如，任意一个盘子，其必须堆在比它大的盘子上面）。同时，你必须满足以下限制条件：

(1) 每次只能移动一个盘子。  
(2) 每个盘子从堆的顶部被移动后，只能置放于下一个堆中。  
(3) 每个盘子只能放在比它大的盘子上面。

请写一段程序，实现将第一个堆的盘子移动到最后一个堆中。

样例1:

输入: 3

输出:

towers[0]: []

towers[1]: []

towers[2]: [2,1,0]

样例2:

输入: 10

输出:

towers[0]: []

towers[1]: []

towers[2]: [9,8,7,6,5,4,3,2,1,0]

## 2、算法原理

定义add()函数和moveTopTo()函数来进行操作。当盘子数<=0时直接返回，当盘子数为1时，调用moveUpTo()函数，其他情况调用moveDisks()函数和moveTopTo()函数后在调用moveUpTo()函数。

## 3、实验数据测试



# 实验三十四、423.有效括号序列

## 1、主要任务

给定一个字符串所表示的括号序列，包含以下字符： '(', ')', '{', '}', '[' and ']'， 判定是否是有效的括号序列。括号必须依照 "()" 顺序表示， "()[]{}" 是有效的括号，但 "([)]" 则是无效的括号。

样例 1：

输入："([)]"

输出：False

样例 2：

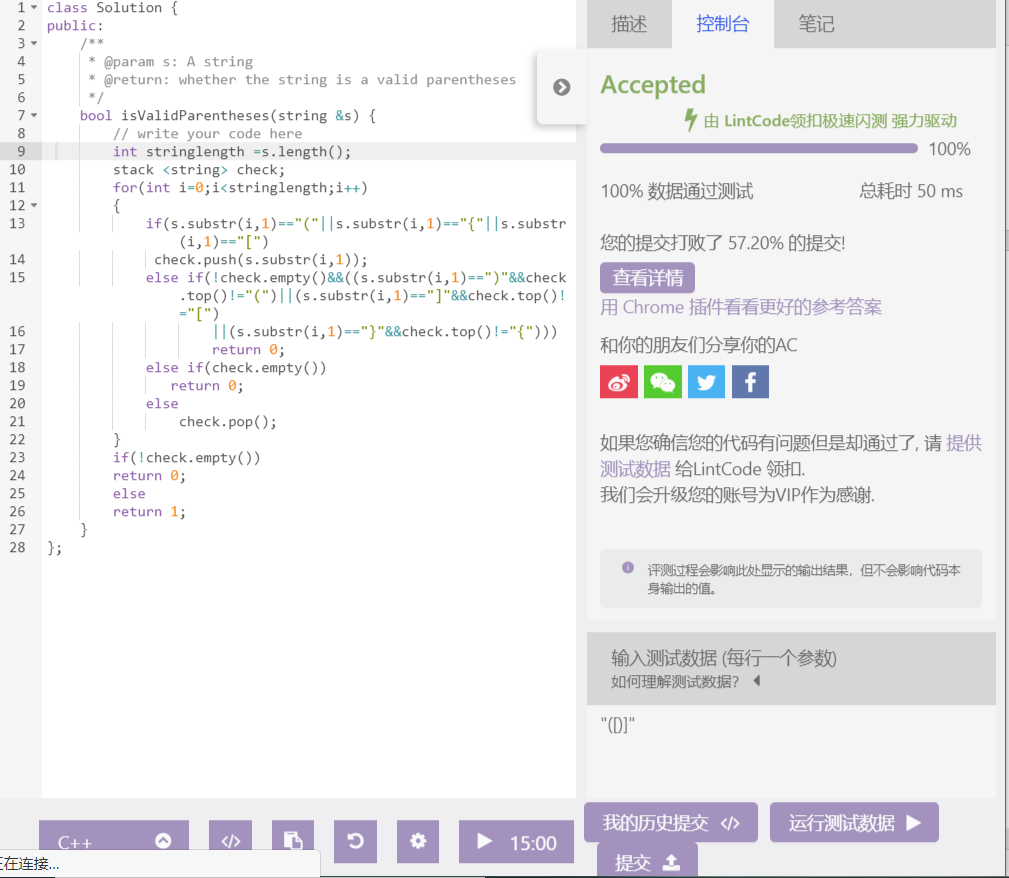
输入："()[]{}"

输出：True

## 2、算法原理

遍历字符串，调用substr()函数获取第i位的字符，若为(或{或[时，入栈。当栈不为空同时第i个字符为]同时栈的栈顶不为(，或第i位为]同时栈顶不为[时，或第i位为}同时栈顶不为{时，返回0.其他若为空栈返回0.或弹出栈顶。此时若不为空返回0，否则返回1。

## 3、实验数据测试



# 实验三十五、693.移掉k位数字

## 1、主要任务

给定一个以字符串表示的非负整数 num，移除这个数中的 k 位数字，使得剩下的数字最小。

num 的长度小于 10002 且 ≥ k。  
num 不会包含任何前导零。

样例 1 :

输入: num = "1432219", k = 3

输出: "1219"

解释: 移除掉三个数字 4, 3, 和 2 形成一个新的最小的数字 1219。

样例 2 :

输入: num = "10200", k = 1

输出: "200"

解释: 移掉首位的 1 剩下的数字为 200. 注意输出不能有任何前导零。

样例 3 :

输入: num = "10", k = 2

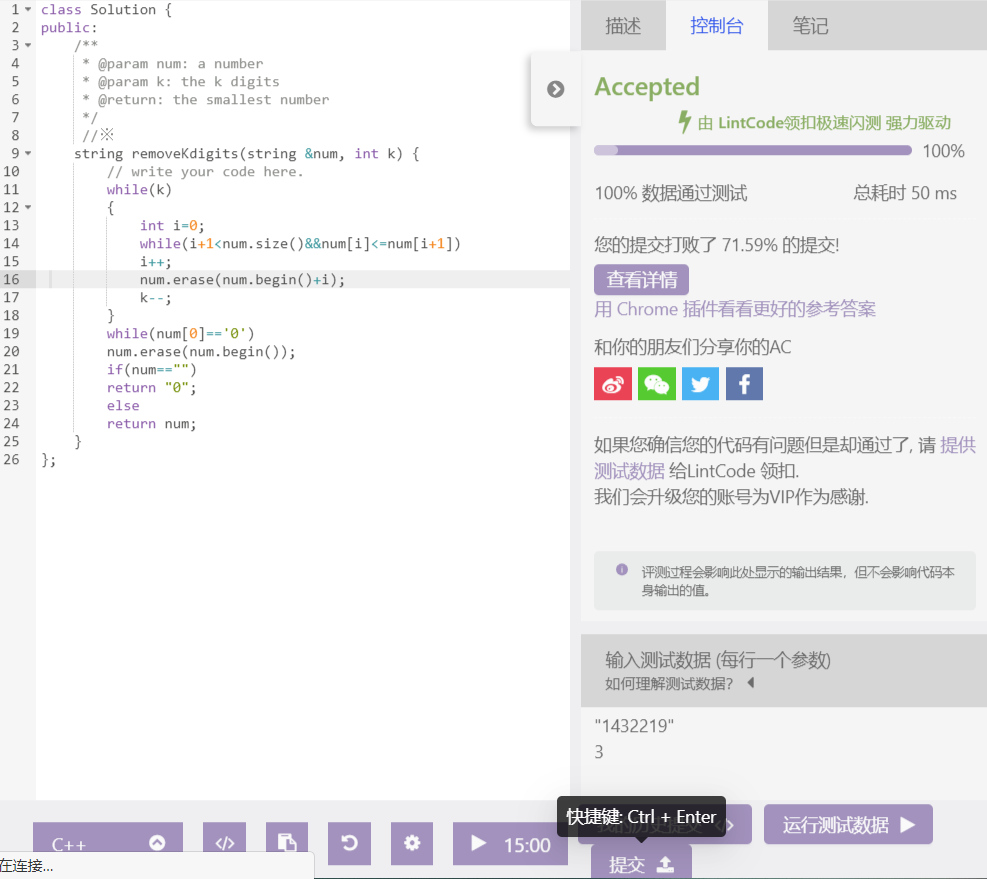
输出: "0"

解释: 从原数字移除所有的数字，剩余为空就是0。

## 2、算法原理

当k不为0时，若i+1小于字符串长度，同时字符的第i位小于或等于第i+1位，i++，调用erase()函数清除第i个元素，k--。直到k为0跳出循环。当num的第一位为0时，清除第一位，如果此时num为空，返回值为0，否则返回num。最终实现移掉k位数字。

## 3、实验数据测试



# 实验三十六、834.移除多余字符

## 1、主要任务

给定一个字符串s由小写字符组成，移除多余的字符使得每个字符只出现一次.必须保证结果是字典序是最小的合法字符串。

样例1

输入: s = "bcabc"

输出: "abc"

样例2

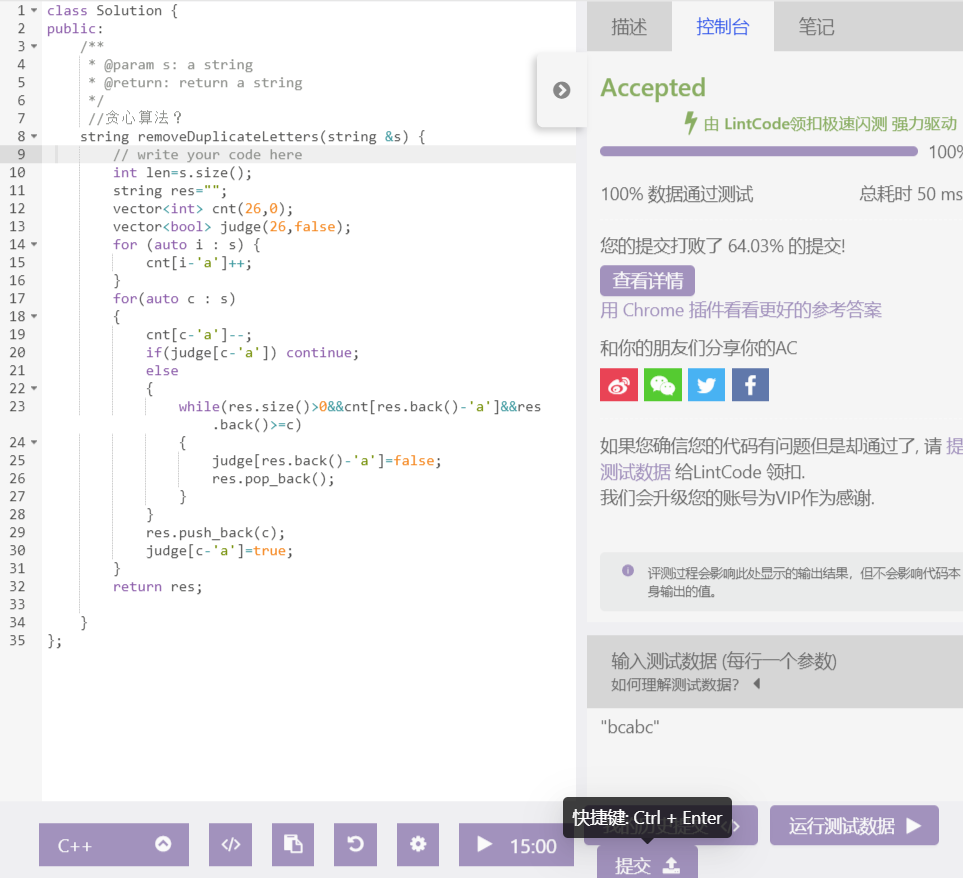
输入: s = "cbacdcbc"

输出: "acdb"

## 2、算法原理

定义了cnt和judge两个数组。利用贪心算法来进行实现。

## 3、实验数据测试



# 实验三十七、983.棒球游戏

## 1、主要任务

现在你是棒球比赛分记录员。

给定一个字符串数组，每一个字符串可以是以下4种中的其中一个：

1. 整数 (一个回合的分数): 直接表示这回合你得到的分数。
2. "+" (一个回合的分数): 表示这回合你获得的分数为前两个 有效 分数之和。
3. "D" (一个回合的分数): 表示这回合你得到的分数为你上一次获得的有效分数的两倍。
4. "C" (一种操作，而非一个回合的分数): 表示你上回合的有效分数是无效的，需要移除。  
   每一轮的操作都是永久性的，可能会影响之前和之后的一轮。

你需要返回在所有回合中获得总分数。

输入列表的大小在 1 到 1000 之间。

列表中的整数大小在 -30000 到 30000 之间。

样例1:

输入: ["5","2","C","D","+"]

输出: 30

解释:

回合 1: 你可以得到 5 分，和为：5。

回合 2: 你可以得到 2 分，和为：7。

操作 1: 回合 2 的数据无效，所以和为 5。

回合 3: 你可以得到 10 分（回合 2 的数据已经被移除了），和为：15。

回合 4: 你可以得到 5 + 10 = 15 分，和为：30。

样例2:

输入: ["5","-2","4","C","D","9","+","+"]

输出: 27

解释:

回合 1: 你可以得到 5 分，和为：5。

回合 2: 你可以得到 -2 分，和为：3。

回合 3: 你可以得到 4 分，和为：7。

操作 1: 回合 3的数据无效，所以和为 3。

回合 4: 你可以得到 -4 分(回合 3的数据已经被移除了)，和为：-1。

回合 5: 你可以得到 9 分，和为：8。

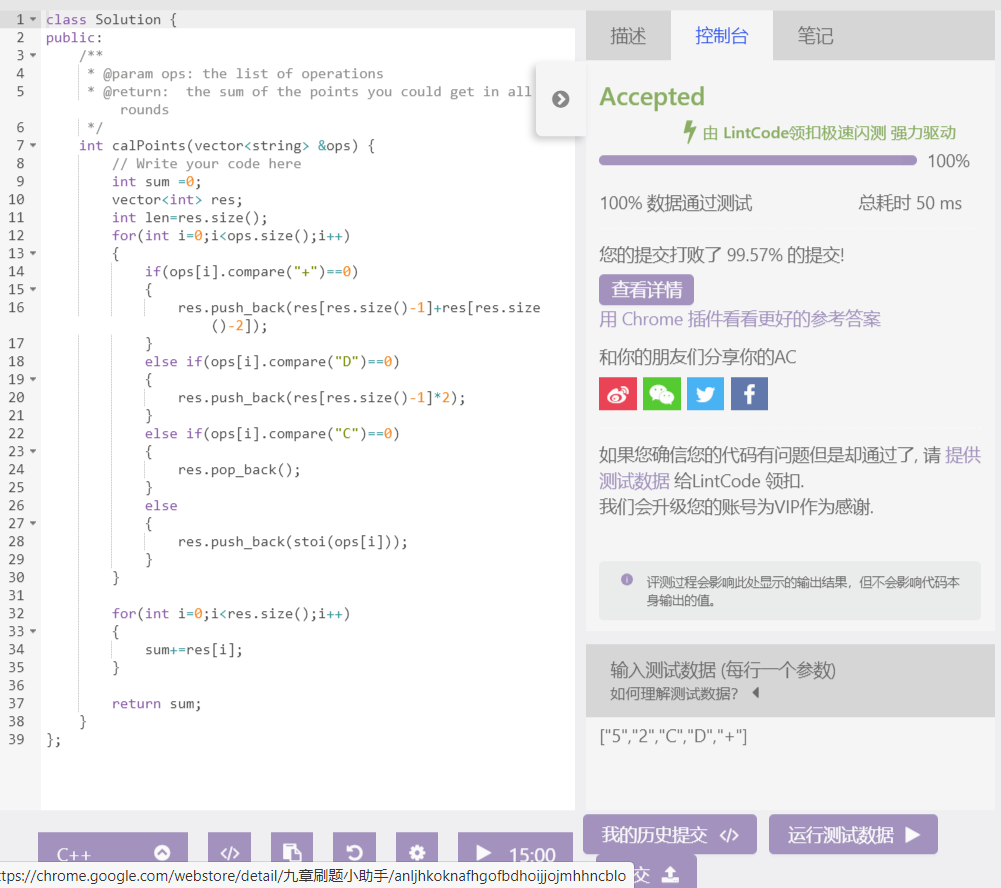
回合 6: 你可以得到 -4 + 9 = 5 分，和为：13。

回合 7: 你可以得到 9 + 5 = 14 分，和为27。

## 2、算法原理

首先定义数组res，遍历字符串数组ops。当元素调用compare()和“+”相同时，将res数组的最后一个元素和倒数第二个元素的和入栈。当元素调用compare()和“D”相同时，最后一个元素乘以2后入栈，当元素调用compare()后和“C”相同时，弹出最后一个元素，否则首先调用stoi()将字符串转化为int型在入栈。最后遍历累加所有元素并且返回总分数。

## 3、实验数据测试



# 实验三十八、1206.下一个更大的数Ⅰ

## 1、主要任务

你有两个数组 nums1和nums2（互不重复），其中nums1是nums2的子集。 在nums2的相应位置找到nums1所有元素的下一个更大数字。

nums1中的数字x的下一个更大数字是nums2中x右边第一个更大的数字。 如果它不存在，则为此数字输出-1。

1.nums1和nums2中的所有数字都是唯一的。  
2.nums1和nums2的长度不超过1000。

例子 1:

输入: nums1 = [4,1,2], nums2 = [1,3,4,2].

输出: [-1,3,-1]

解释:

对于第一个数组中的数字4，在第二个数组中找不到下一个更大的数字，因此输出-1。

     对于第一个数组中的数字1，第二个数组中的下一个更大数字是3。

     对于第一个数组中的数字2，第二个数组中没有下一个更大的数字，因此输出-1。

例子 2:

输入: nums1 = [2,4], nums2 = [1,2,3,4].

输出: [3,-1]

解释:

对于第一个数组中的数字2，第二个数组中的下一个更大数字是3。

     对于第一个数组中的数字4，第二个数组中没有下一个更大的数字，因此输出-1。

## 2、算法原理

首先定义一个和nums1长度相同的数组result，并且分别记录nums1和nums2两个数组的长度。之后根据题目要求遍历数组进行操作。

## 3、实验数据测试



# 四、队列算法练习

# 实验三十九、492.队列维护

## 主要任务

实现一个队列的操作

enqueue(item).将新元素放入队列中。

dequeue(). 将第一个元素移出队列，返回它。

例1:

输入：

enqueue(1)

enqueue(2)

enqueue(3)

dequeue() // return 1

enqueue(4)

dequeue() // return 2

例2:

输入：

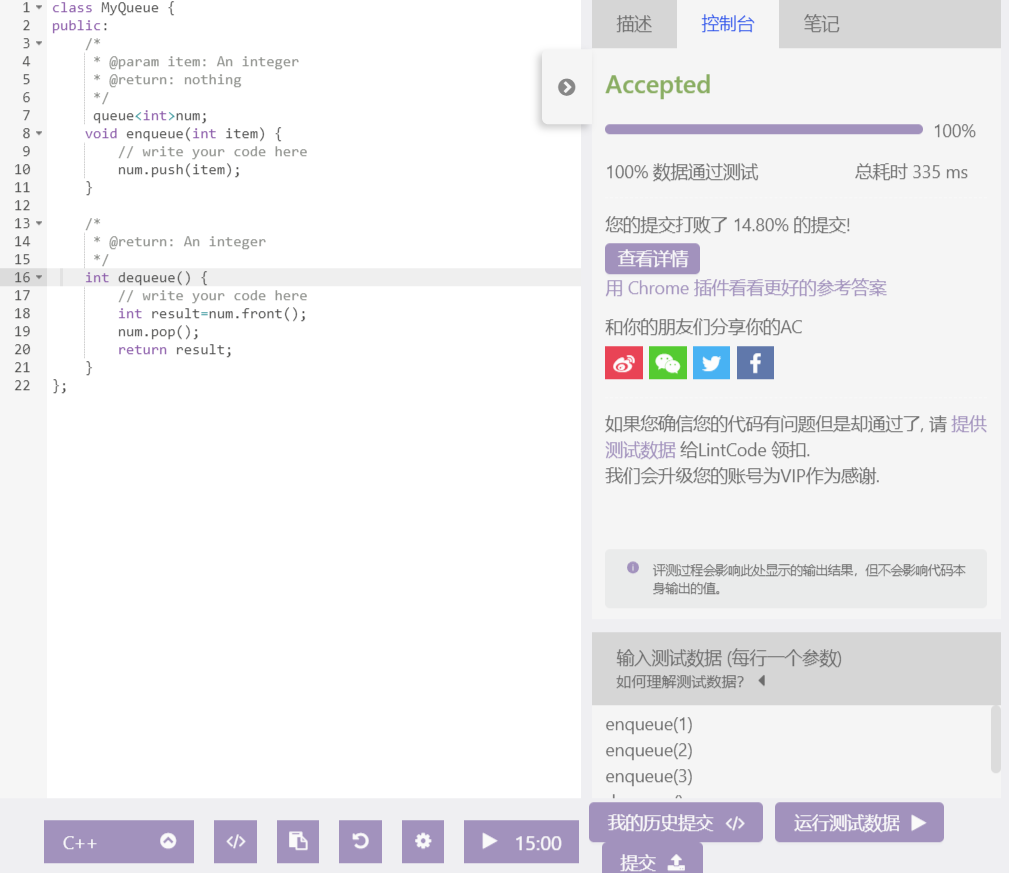
enqueue(10)

dequeue()// return 10

## 算法原理

Enqueue()操作使用队列的push()函数来实现，dequeue()操作首先用result来保存队列的第一个元素，调用pop()出列，返回result即返回了队列的第一个元素。

## 实验数据测试



# 实验四十、1206.下一个更大的数Ⅰ

## 主要任务

给出一串整数流和窗口大小，计算滑动窗口中所有整数的平均值。

样例1 :

MovingAverage m = new MovingAverage(3);

m.next(1) = 1 // 返回 1.00000

m.next(10) = (1 + 10) / 2 // 返回 5.50000

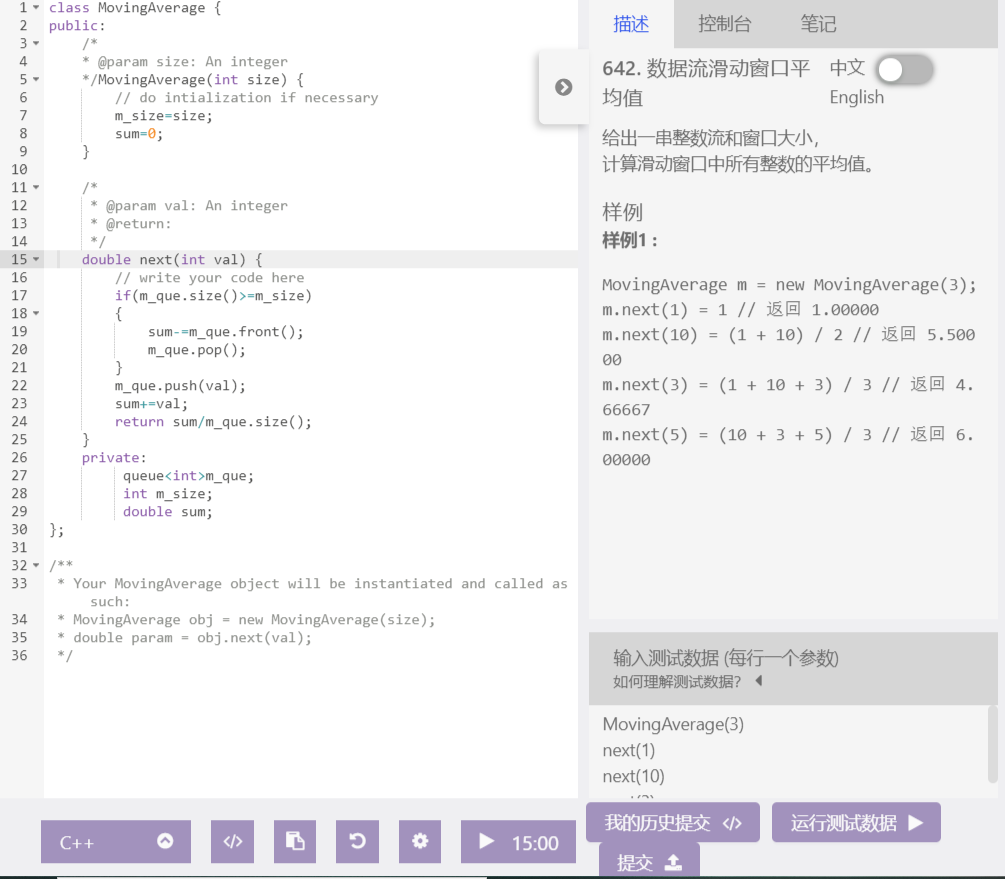
m.next(3) = (1 + 10 + 3) / 3 // 返回 4.66667

m.next(5) = (10 + 3 + 5) / 3 // 返回 6.00000

## 算法原理

使用队列来实现题中所需要的操作，当队列的长度大于或等于m\_size时，sum为sum-队列的第一个元素，之后出列。否则将val入列，sum加上val，返回值为sum/m\_que.size()。

## 实验数据测试



# 实验四十一、1166.推荐结果打散

## 主要任务

某个推荐系统会推荐出一个由视频和图片元素组成的列表。x 表示元素编号，视频用 V\_x 表示，图片用 P\_x 表示。现在需要对这些元素进行打散，打散规则如下：

1. 第一个[图片P]出现的位置不变；
2. 从第一个[图片P]后开始，每 n 个元素里面恰好出现1个[图片P]；
3. 图片之间的相对顺序不变；

不能满足打散规则的元素需要舍去。给出元素列表和 n 的值，请返回打散后的元素列表。

1 ≤元素个数 ≤ 10000  
2 ≤ n ≤ 元素个数  
保证元素中至少有 1 个 [图片P]

样例1：  
输入样例：  
elements=["V\_0", "V\_1", "V\_2", "P\_3", "P\_4", "P\_5", "V\_6", "P\_7", "V\_8", "V\_9"] , n=3  
输出样例：  
["V\_0", "V\_1", "V\_2", "P\_3", "V\_6", "V\_8", "P\_4", "V\_9"]  
样例解释：  
P\_3 需出现在从0开始的第3位，由于每3个元素中恰好一个 P，P\_5 和 P\_7 满足不了打散要求，被删除。

样例2：  
输入样例：  
elements=["V\_0", "P\_1", "V\_2", "V\_3", "V\_4"] , n=2  
输出样例：  
["V\_0", "P\_1", "V\_2"]  
样例解释：  
P\_1 需出现在从0开始的第1位，由于此后每2个元素中恰好一个 P，V\_3 和 V\_4 满足不了打散要求，被删除。

## 算法原理

用队列来实现，先把第一个P出现之前的V都存到结果里面，当出现第一个P之后，用2个队列分别存储P和V，然后打散存储，比如n=3，则1个P2个V，直到V队列全部用完。

## 3、实验数据测试

