



你好，我是王争。首先祝你新年快乐！

专栏的正文部分已经结束，相信这半年的时间，你学到了很多，究竟学习成果怎样呢？

我整理了数据结构和算法中必知必会的30个代码实现，从今天开始，分7天发布出来，供你复习巩固所用。你可以每天花一点时间，来完成测验。测验完成后，你可以根据结果，回到相应章节，有针对性地进行复习。

除此之外，@Smallfly 同学还整理了一份配套的LeetCode练习题，你也可以一起练习一下。在此，我谨代表我本人对@Smallfly 表示感谢！

另外，我还为假期坚持学习的同学准备了丰厚的[春节加油礼包](#)。

1. 2月5日-2月14日，只要在专栏文章下的留言区写下你的答案，参与答题，并且留言被精选，即可获得极客时间10元无门槛优惠券。
2. 7篇中的所有题目，只要回答正确3道及以上，即可获得极客时间99元专栏通用阅码。
3. 如果7天连续参与答题，并且每天的留言均被精选，还可额外获得极客时间价值365元的每日一课年度会员。

## 关于数组和链表的几个必知必会的代码实现

### 数组

- 实现一个支持动态扩容的数组
- 实现一个大小固定的有序数组，支持动态增删改操作
- 实现两个有序数组合并为一个有序数组

## 链表

- 实现单链表、循环链表、双向链表，支持增删操作
- 实现单链表反转
- 实现两个有序的链表合并为一个有序链表
- 实现求链表的中间结点

## 对应的LeetCode练习题 (@Smallfly 整理)

### 数组

- Three Sum (求三数之和)

英文版: <https://leetcode.com/problems/3sum/>

中文版: <https://leetcode-cn.com/problems/3sum/>

- Majority Element (求众数)

英文版: <https://leetcode.com/problems/majority-element/>

中文版: <https://leetcode-cn.com/problems/majority-element/>

- Missing Positive (求缺失的第一个正数)

英文版: <https://leetcode.com/problems/first-missing-positive/>

中文版: <https://leetcode-cn.com/problems/first-missing-positive/>

### 链表

- Linked List Cycle I (环形链表)

英文版: <https://leetcode.com/problems/linked-list-cycle/>

中文版: <https://leetcode-cn.com/problems/linked-list-cycle/>

- Merge k Sorted Lists (合并k个排序链表)

英文版: <https://leetcode.com/problems/merge-k-sorted-lists/>

中文版: <https://leetcode-cn.com/problems/merge-k-sorted-lists/>

---

做完题目之后，你可以点击“请朋友读”，把测试题分享给你的朋友，说不定就帮他解决了一个难题。

祝你取得好成绩！明天见！

# 数据结构与算法之美

为工程师量身打造的数据结构与算法私教课

王争

前 Google 工程师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

## 精选留言



李皮皮皮皮

感谢分享，虽然工作很忙，每天下班就不想动了。但是还是要不断克服自己。数据结构和算法的重要性可能在面试的时候才能深刻感悟。如果平时多下点功夫，结果可能会大不一样。前面很多期因为各种原因没有跟上，庆幸的是后面慢慢追上了。现在养成每天做一道算法题的习惯。每天装着一道算法题在脑子里。这感觉其实也不错，不是任务，感觉像是习惯

2019-02-04 21:09



Jerry银银

早上起来拿出电脑，准备做题。

老妈说：今天就别工作了，玩一天吧，啥也别干，啥也别想。

我说：不行呀，老师布置了题目，必须得做呀。

老妈说：大过年的老师还在工作，真不容易，替我向你老师说声：新年好！！

2019-02-05 11:11



Smallfly

哈哈，被提名了，谢谢老师。

有兴趣的同学可以把你的答案分享到 Github: <https://github.com/iostalks/Algorithms>

有问题也可以在 issue 中一起讨论。

新的一年跟大家一起进步，一起流弊。

2019-02-05 09:58



fancyyou

新年好！

leetcode的题都做过了。

2019-02-05 10:29



峰

第三题，看这题，我就会想到用快排的思想在一堆数中求第n大。于是乎我就套，先把负数全部移掉， $O(n)$ 不影响。然后每轮迭代随机取个数n，比它小的放左边，比他大的放右边。比如说第一轮迭代，左边的数据个数小于n-1那么必然在左边。但这里有

个问题是数据是可以重复的，怎么办，想呀想，我就选定n后，开始扫描，如果是1我就放第一个位置，如果是2我就放第二个位置，如果再有1，发现重复了，不用移动了，这样我就能计算小于n大于n的正整数有多少种了，然后就能迭代下去了。当然里面还有些细节，比如如果n很大已超过了数组长度，那说明那个数一定在左边。

2019-02-05 10:23

kai

### 3. 实现求链表的中间结点

```
public class FindMidNode {

    // 1.  $T(n) = O(2*n)$  遍历2次
    public static Node findMidNode(Node head) {
        if (head == null) {
            return null;
        }

        int len = 0;
        Node p = head;

        while(p != null) {
            len++;
            p = p.next;
        }

        p = head;
        for (int i = 0; i < len/2; i++) {
            p = p.next;
        }

        return p;
    }

    // 2.  $T(n) = O(n)$  遍历1次
    // 快慢指针法
    public static Node findMidNodeFast(Node head) {
        if (head == null) {
            return null;
        }

        Node fast = head;
        Node slow = head;

        while (fast != null && fast.next != null) {
            fast = fast.next.next;
            slow = slow.next;
        }

        return slow;
    }

    public static Node createNode(int value) {
        return new Node(value, null);
    }
}
```

```
}
```

```
public static class Node {  
    public int data;  
    public Node next;
```

```
    public Node(int data, Node next) {  
        this.data = data;  
        this.next = next;  
    }  
}  
}
```

#### 4. Linked List Cycle I (环形链表)

```
/**  
 * 141. Linked List Cycle  
 * https://leetcode.com/problems/linked-list-cycle/  
 */  
public class LinkedListCycle {  
    public boolean hasCycle(ListNode head) {  
        if (head == null || head.next == null) {  
            return false;  
        }
```

```
        ListNode fast = head;  
        ListNode slow = head;
```

```
        while (fast != null && fast.next != null) {  
            fast = fast.next.next;  
            slow = slow.next;  
            if (fast == slow) return true;  
        }
```

```
        return false;  
    }
```

```
    public static class ListNode {  
        int val;  
        ListNode next;  
        ListNode(int x) { val = x; }  
    }  
}
```

2019-02-11 10:12



William

特地新开了一个git仓库, <https://github.com/Si3ver/LeetCode>。刷完5道题, 思路大致写一下。1.数组三数之和, 时间复杂度是 $O(n^2)$ , 先排序, 外层遍历数组, 内层左右双指针, 寻找两数之和 =  $-nums[i]$ 。2. 求数组中出现次数大于一半的数字。复杂度 $O(n)$ , 是利用摩尔投票法。3.求缺失的最小正整数, 复杂度 $O(n)$ , 思路是哈希表统计。4.环形链表用快慢指针。5.合并k个有序链表, 用的是两两归并, 据说用堆会更快, 这个有待补充。

2019-02-06 16:23



abner

Java语言实现一个大小固定的有序数组, 支持动态增删改操作

代码如下:

```
public class Array {
    private String[] data;
    private int count;
    private int size;
    public Array(int capacity) {
        data = new String[capacity];
        count = 0;
        size = capacity;
    }
    public boolean insert(int index, String value) {
        if (count >= size) {
            return false;
        }
        if (index < 0 || index > count) {
            return false;
        }
        for (int i = count - 1; i >= index; i--) {
            data[i+1] = data[i];
        }
        data[index] = value;
        count++;
    }
    public String delete(int index, String value) {
        if (count == 0) {
            return false;
        }
        if (index < 0 || index > count) {
            return false;
        }
        value = data[index];
        for (int i = index; i <= count - 1; i++) {
            data[i - 1] = data[i];
        }
        count--;
        return value;
    }
}
```

2019-02-05 21:56



\_CountingStars

合并有序数组 go 语言实现

package main

import "fmt"

```
func mergeOrderedArray(a, b []int) (c []int) {
    i, j, k := 0, 0, 0
    mergedOrderedArrayLength := len(a) + len(b)
    c = make([]int, mergedOrderedArrayLength)
    for {
        if i >= len(a) || j >= len(b) {
            break
        }
    }
}
```

```

}

if a[i] <= b[j] {
c[k] = a[i]
i++
} else {
c[k] = b[j]
j++
}
k++
}

for ; i < len(a); i++ {
c[k] = a[i]
k++
}

for ; j < len(b); j++ {
c[k] = a[j]
k++
}

return
}

func main() {
a := []int{1, 3, 5, 7, 9, 10, 11, 13, 15}
b := []int{2, 4, 6, 8}
fmt.Println("ordered array a: ", a)
fmt.Println("ordered array b: ", b)
fmt.Println("merged ordered array: ", mergeOrderedArray(a, b))
}

```

2019-02-05 19:51



abner

java实现一个动态扩容的数组（扩容2倍）

代码如下：

```
package array;
```

```
public class DynamicArray {
```

```
    private String[] data;
```

```
    private int count;
```

```
    private int size;
```

```
    public DynamicArray(int capacity) {
```

```
        data = new String[capacity];
```

```
        count = 0;
```

```
        size = capacity;
```

```
    }
```

```

public String[] expand(String[] data) {
    if (count >= size) {
        String[] newArray = new String[this.size * 2];
        this.size = this.size * 2;
        for (int i = 0; i < count; i++) {
            newArray[i] = this.data[i];
        }
        return newArray;
    } else {
        return this.data;
    }
}

public boolean append(String item) {
    if (count >= size) {
        this.data = expand(this.data);
    }
    this.data[count] = item;
    count++;
    return true;
}

public void printAll() {
    for (int i = 0; i < count; i++) {
        System.out.print(data[i] + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    DynamicArray dynamicArray = new DynamicArray(5);
    for (int i = 0; i < dynamicArray.size; i++) {
        dynamicArray.data[i] = "This value is " + i;
        dynamicArray.count++;
    }
    dynamicArray.append("This value is 5");
    System.out.println("Now the size of data is " + dynamicArray.size);
    dynamicArray.printAll();
}
}

```

2019-02-13 01:06



Zoctopus  
Zhang

Three Sum (求三数之和) Go语言:

```

func threeSum(nums []int) [][]int {
    results := [][]int{}
    n := len(nums)
    if n == 0 || n < 3 {
        return results
    }
}

```



```

}
sort.Ints(nums) //首先, 对数组进行排序
for i := 0; i < n-2; i++ {
if i > 0 && nums[i] == nums[i-1] { //如果相邻两个数相等
continue
}
target := -nums[i]
left := i + 1
right := n - 1
for left < right {
sum := nums[left] + nums[right]
if sum == target {
results = append(results, []int{nums[left], nums[right], nums[i]})
left++
right--
for left < right && nums[left] == nums[left-1] {
left++
}
for left < right && nums[right] == nums[right+1] {
right--
}
} else if sum > target {
right--
} else if sum < target {
left++
}
}
}
return results
}

```

2019-02-12 19:32



Sharry

链表篇

### 1. 翻转单链表

/\*翻转单链表\*/

```

void reversalList(Node<int>* head) {
Node<int>* p = head;
Node<int>* prev = NULL;
Node<int>* temp = NULL;
while (p) {
// 1. 保存要遍历的下一个结点
temp = p->next;
// 2. 将 node->next 指向前驱结点
p->next = prev;
// 3. 更新前驱结点
prev = p;
// 4. 更新下一个要遍历的结点
p = temp;
}
}

```

```
}
```

## 2. 将两个有序的单链表合并

```
/* 合并两个有序链表, 将 list2 合并到 list1 中 */
Node<int>* mergeOrderList(Node<int>* list1, Node<int>* list2) {
// 记录 list2 的头结点
Node<int>* head = list2;
// 创建哨兵, 用于处理将 list2 中的元素插入到 list1 头结点前面的情况
Node<int>* sentry = new Node<int>(-1);
sentry->next = list1;
// 记录 list1 要遍历的元素
Node<int>* node = sentry;
Node<int>* temp = NULL;
while (node->next && head) {
if (node->next->data > head->data) {
temp = head->next;
head->next = node->next;
node->next = head;
head = temp;
}
else {
node = node->next;
}
}
// 若 list2 的头结点不为 NULL, 则说明 list1 中的元素提前遍历结束了
// 剩下的 list2 中的元素均比 list1 中的大
// 直接将 list1 的尾结点连接到 list2 的首结点即可
if (head) {
node->next = head;
}
// 释放哨兵结点内存
list1 = sentry->next;
sentry->next = NULL;
delete(sentry);
return list1;
}
```

## 3. 求单链表的中间结点

```
/* 查询单链表的中间结点 */
template<typename E>
Node<E>* findMidNode(Node<E>* head, Node<E>** mid_node) {
if (!head) {
return NULL;
}
Node<E>* fast = head;
Node<E>* slow = head;
while (fast && fast->next && fast->next->next) {
// 快指针走两步
fast = fast->next->next;
// 慢指针走一步
slow = slow->next;
}
```

```
}
*mid_node = slow;
}
```

2019-02-12 11:26

kai

## 1. 实现单链表反转:

```
/**
 * 206. Reverse Linked List
 * https://leetcode.com/problems/reverse-linked-list/
 */
public class ReverseList {
    public ListNode reverseList(ListNode head) {
        if (head == null || head.next == null) return head;

        ListNode pre = null;
        ListNode next = null;

        while (head != null) {
            next = head.next;
            head.next = pre;
            pre = head;
            head = next;
        }

        return pre;
    }

    public static class ListNode {
        int val;
        ListNode next;
        ListNode(int x) {
            this.val = x;
        }
    }
}
```

## 2. 实现两个有序的链表合并为一个有序链表

```
/**
 * 21. Merge Two Sorted Lists
 * https://leetcode.com/problems/merge-two-sorted-lists/
 */
public class Merge2SortedLists {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

        // 利用哨兵（前哨节点）简化实现难度
        ListNode outpost = new ListNode(-1);
        ListNode temp = outpost;

        while (l1 != null && l2 != null) {
```

```

    if (l1.val <= l2.val) {
        temp.next = l1;
        l1 = l1.next;
    } else {
        temp.next = l2;
        l2 = l2.next;
    }

    temp = temp.next;
}

if (l1 == null) {
    temp.next = l2;
}

if (l2 == null) {
    temp.next = l1;
}

return outpost.next;
}

public ListNode mergeTwoListsRecur(ListNode l1, ListNode l2) {
    if (l1 == null) return l2;
    if (l2 == null) return l1;
    if (l1.val < l2.val) {
        l1.next = mergeTwoListsRecur(l1.next, l2);
        return l1;
    } else {
        l2.next = mergeTwoListsRecur(l1, l2.next);
        return l2;
    }
}

public static class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x;}
}
}

```

2019-02-11 10:10



神盾局闹别扭

实现两个有序的链表合并为一个有序链表：

```

Node *MergeNode(Node *head1, Node *head2)
{
    if (head1 == NULL)
        return head2;

```

```

if (head2 == NULL)
return head1;
stu *pMergedHead;
if (head1->age < head2->age)
{
pMergedHead = head1;
pMergedHead->next = MergeNode(head1->next, head2);
}
else
{
pMergedHead = head2;
pMergedHead->next = MergeNode(head1, head2->next);
}
return pMergedHead;
}

```

2019-02-07 21:53



ALAN

linkedList answer:

```

import java.util.ArrayList;
import java.util.List;

```

```

/**
 *
 * @author root alan
 *
 */
public class List1 {

Node tail;
Node head;

public void addOneWay(int value) {
if (head == null) {
head = new Node(value);
tail = head;
} else {
Node node = new Node(value);
tail.next = node;
tail = node;
}
}

public void deleteOneWay(int value) {
Node node = head;
Node prev = head;
while (node.value != value) {
prev = node;
node = node.next;
}
if (node == head)

```

```

head = head.next;
else if (node != tail)
prev.next = node.next;
else {
tail = prev;
prev.next = null;
}
}

```

```

public Node reverse(Node node) {
Node prev = null;
Node now = node;
while (now != null) {
Node next = now.next;
now.next = prev;
prev = now;
now = next;
}

return prev;
}

```

```

public Node middle() {
Node nd = head;
Node nd2 = head;
while (nd2 != null) {
nd = nd.next;
nd2 = nd2.next.next;
}
return nd;
}
}

```

```

class Node {
Node prev;
Node next;
int value;

public Node(int ele) {
value = ele;
}
}

```

2019-02-07 11:00



ALAN

array answer:

```
import java.util.Arrays;
```

```

public class Array1 {
    public int n;
    public int cur;
    public static int ary[]; //dynamic expand
    public static int fix[]; //fixed array
    public Array1(int size) {
        n=size;
        ary=new int [n];
    }

    //dynamic expand
    public void insert(int ele) {
        if(cur==ary.length) {

            ary=Arrays.copyOf(ary, ary.length*2);
            System.out.println("length:"+ary.length);

        }
        ary[cur]=ele;
        cur++;

    }
    //fixed array --add
    public void add(int ele) {
        if(cur==fix.length) {
            return;
        }
        fix[cur]=ele;
        cur++;
    }
    //fixed array --delete
    public void delete() {
        if(cur== -1)
            return ;
        fix[cur]=0;
        cur--;
    }
    //fixed array --update
    public void update(int index,int ele) {
        if(index>=0 && index<fix.length)
            fix[index]=ele;
    }
    //merge
    public int[] merge(int[] a,int[] b ) {
        int[]c =new int[a.length+b.length];
        int j=0,k=0;
        for(int i=0;i<a.length+b.length;i++) {
            if(j==a.length) {
                c[i]=b[k];
                k++;
            }
        }
    }
}

```

```

continue;
}else if(k==b.length){
c[i]=a[j];
j++;
continue;

}
if(a[j]<b[k]) {
c[i]=a[j];
j++;
}else {
c[i]=b[k];
k++;
}

}

return c;
}
}

```

2019-02-07 10:57



SyndromePolynomial

大小固定的有序数组，支持增删改：既然有序，则查询操作都可以用二分查询。增加操作，找到第一个大于新数据的值的位置，从最后一个有效数据往后移一个位置，目的是为了给新数据腾位置，然后插入。删除操作：找到第一个等于要删除的数据的值，然后将其后面的数据依次向前挪一个位置。改操作，查询再修改。要注意临界条件和找不到数据，以及数组满等情况。

2019-02-06 10:35



吴...

祝大家新年快乐，王老师真的太负责了，不光是在新年更新，更重要的是老师能够在教完之后还为我们安排课程巩固。

2019-02-05 12:29



李汶泽

//合并两个有序链表

```
def mergeTwoLists(self, l1, l2):
```

```
if not l1:
```

```
return l2
```

```
if not l2:
```

```
return l1
```

```
p = ListNode(0)
```

```
head = p
```

```
while l1 and l2 :
```

```
if l1.val <= l2.val:
```

```
p.next = l1
```

```
l1 = l1.next
```

```
else:
```

```
p.next = l2
```

```
l2 = l2.next
```

```
p = p.next
```

```
if l1 :
```

```
p.next = l1
```

```
else:
```



```
p.next = l2  
return head.next  
2019-02-14 23:34
```



李汶泽

//双向链表的实现及其增删操作

```
#include<stdlib.h>  
#define LEN sizeof(struct node)  
struct node{  
    int num;  
    struct node *pre;  
    struct node *next;  
};  
struct node *creatList(int n){  
    struct node *head,*p1,*p2;  
    int i;  
    if(n<1){  
        printf("链表创建失败\n");  
    }  
    else{  
        head=(struct node*)malloc(sizeof(LEN));  
        head->next=NULL;  
        head->pre=NULL;  
        for(i=0 ; i<n ; i++){  
            p1=(struct node*)malloc(sizeof(LEN));  
            p1->num=i;  
            p1->next=head->next;  
            head->next=p1;  
        }  
        p1=head;  
        while(p1->next!=NULL){  
            p1->next->pre=p1;  
            p1=p1->next;  
        }  
        printf("双向链表创建成功\n");  
    }  
    return head;  
}  
struct node *getelement(struct node *head,int n){  
    struct node *p;  
    int i=1;  
    if(n<1){  
        printf("链表中不存在该节点\n");  
    }  
    else{  
        p=head->next;  
        while(p!=NULL&& i<n){  
            p=p->next;  
            i=i+1;  
        }  
    }  
    if(p!=NULL){
```

```

return p;
}
else {
printf("该节点不存在\n");
return NULL;
}
}
//删除
void del(struct node *head,int n){//删除
struct node *p1,*p2;
if(n<1){
printf("该节点不存在\n");
head=NULL;
}
else if(n==1){
head->next->pre=NULL;
head=head->next ;
free(head);
}
else{
p1=getelement(head,n);
if(p1==NULL){
printf("该节点不存在\n");
}else if(p1->next==NULL){
p1->pre->next=NULL;
free(p1);
}else{
p1->pre->next=p1->next;
p1->next->pre=p1->pre;
free(p1);
}
}
}
void insert(struct node *head,int n){
struct node *p1,*p2,*p3;
p3=(struct node*)malloc(sizeof(Len));
p3->num=10000;
if(n<1)
printf("该节点不存在，无法插入\n");
else if(n==1){
p3->next=head;
head->pre=p3;
p3->pre=NULL;
head=p3;
}else{
p1=getelement(head,n-1);
if(p1==NULL){
printf("该节点不存在，无法插入\n");
}else if(p1->next==NULL){
p1->next=p3;
p3->pre=p1;

```

```
p3->next=NULL;
}else{
p1->next->pre=p3;
p1->next=p3;
p3->pre=p1;
p3->next=p1->next;
}
}
}
```

2019-02-14 19:42