春节7天练|Day1：数组和链表

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！

精选留言：

- 李皮皮皮皮皮 2019-02-04 21:09:26
  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 [13赞]

- Jerry银银 2019-02-05 11:11:12
  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！！！ [7赞]

- Smallfly 2019-02-05 09:58:25
  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。 [7赞]

- William 2019-02-06 16:23:10
  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i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。 [2赞]

- fancyyou 2019-02-05 10:29:24
  新年好！
  leetcode的题都做过了。 [2赞]

- 峰 2019-02-05 10:23:51
  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。 [2赞]

- ALAN 2019-02-07 11:00:40
  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;
}
}
[1赞]

- ALAN 2019-02-07 10:57:52
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;
}
}
```

[1赞]

- SyndromePolynomial 2019-02-06 10:35:47
  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。[1赞]

- abner 2019-02-05 21:56:58

  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;
privvate 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;
} [1赞]
```

- _CountingStars 2019-02-05 19:51:37

  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))
}
```

[1赞]

- 吴... 2019-02-05 12:29:55
  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。[1赞]

- Hot Heat 2019-02-10 19:30:49
  Leetcode 前三道题
  https://github.com/hotheat/LeetCode/tree/master/015.%203Sum
  https://github.com/hotheat/LeetCode/tree/master/169.%20Majority%20Element
  https://github.com/hotheat/LeetCode/tree/master/041.%20First%20Missing%20Positive
  关于链表和数组的一些操作
  https://github.com/hotheat/JiKeExcercise/tree/master/python-code/05_array

https://github.com/hotheat/JiKeExcercise/blob/master/python-code/06_linked_list/Single_Linked_List.py

- 墨禾 2019-02-09 19:17:03
  做出来的时候有点兴奋，做不出来的时候颓废得有点想放弃，但是看到大家的精彩评论，又有一种征服难题的动力。“不擅长的刚好是自己的短板”心里磨练无数次，才让自己在刷剧和刷算法之间做了抉择

- 你看起来很好吃 2019-02-08 17:21:05
  第一个缺失的正整数Python实现，时间 复杂度O(nlogn)：
  class Solution:
  def firstMissingPositive(self, nums):
  result = min = 1

  nums.sort()
  for num in nums:
  if num == result:
  result += 1
  return result

- 纯洁的憎恶 2019-02-08 16:08:32
  1.Three Sum：暴力匹配三元组，三层循环结束后打印保存三元组的数组即可，时间复杂度O（n^3），空间复杂度O（n）。简化一下，为减少比较次数先排序。外层循环i遍历数组，内层循环从数组两头元素（s、t）开始考察，找出使num【s】+num【t】=-num【i】的s和t，若大了t- -，若小了s++（内层要避开i）s大于等于t则匹配失败。这样两层循环就可以了，时间复杂度O（n^2）。

  2.Majority Element：重点在于统计每个元素出现次数，可以先排序，然后顺序计算出每个数的出现次数，与阈值比较，大于则输出，时间复杂度O（nlog n）。也可以采用散列表，把每个元素存入散列表，并记录出现次数，最后把出现次数超过阈值的元素输出即可，时间复杂度O（n），空间复杂度O（n）。

  3.Missing Positive：本来想用散列表，发现要求时间复杂度O（n），空间复杂度为常量，有点捉急。只能从原数组上做文章。假设数组A长度为n，若i为1到n的正整数，若i存在于A中，我们就把它的位置调整到A【i-1】处，这样通过A【i】是否为i+1即可知道i+1是否在数组中。那么A中不满足上述条件的最小下标+1即为缺失的最小正整数值。

  4. Linked List Cycle I（环形链表）：用图的拓扑排序算法可以，但是要统计顶点出入度，空间复杂度无法达到O（1）。那可以用快慢指针，*fast以*slow的两倍速前进，如果fast和slow重合则说明有环。

5. Merge k Sorted Lists（合并 k 个排序链表）：两两硬生生合并，时间复杂度应该是O（kN），再高级的方法想不出来。ps：如果可以抛弃原来的链表，那么新建一个合并后链表的时间复杂度可以是O（N）吧？N是k个链表的总长。

- 你看起来很好吃 2019-02-08 16:01:52
  Majority Element用python实现，使用散列表实现：

  ```
  from collections import defaultdict

  class Solution:
  def majorityElement(self, nums: 'List[int]') -> 'int':
  count = len(nums)//2
  result = defaultdict(int)

  for num in nums:
  result[num] += 1

  for key in result.keys():
  if result[key] > count:
  return int(key)
  ```

- 你看起来很好吃 2019-02-07 23:33:14
  三数之和python实现：

  ```
  class Solution:
  def threeSum(self, nums: 'List[int]') -> 'List[List[int]]':
  result = []

  nums.sort()
  N = len(nums)

  for i in range(N):
  if i > 0 and nums[i] == nums[i-1]:
  continue
  ```

```
target = nums[i] * -1
s, e = i+1, N-1

while s < e:
if nums[s] + nums[e] == target:
result.append([nums[i], nums[s], nums[e]])
s += 1
while s < e and nums[s] == nums[s-1]:
s += 1
elif nums[s] + nums[e] < target:
s += 1
else:
e -= 1

return result
```

时间复杂度主要由外层for和内层while决定，是O(n*n) + O(list排序)，实际上python中List排序使用了timsort, 时间复度度是O(nlogn), 所以整个代码的时间复杂度是O(n*n)

- 神盾局闹别扭 2019-02-07 22:08:05
  实现求链表的中间结点：

```
Node *GetMidNode(Node *head) {
if (head == nullptr)
return nullptr;

Node *slow = head;
Node *fast = head;

while (fast->next != nullptr && fast->next->next != nullptr)
{
fast = fast->next->next;
```

```
slow = slow->next;
}

return slow;
}
```

- 神盾局闹别扭 2019-02-07 21:53:12

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;
}
```