高频算法之王——
双指针算法之相向双指针

主讲人 令狐冲
课程版本 v7.0

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像

一经发现，将被追究法律责任和赔偿经济损失

双指针的类型

相向双指针

两根指针一头一尾，向中间靠拢直到相遇

时间复杂度 O(n)

Two Sum 类

先修内容中我们已经讲解了双指针的经典题 Two Sum
接下来我们来看这类问题可能的变化

两数之和 - 数据结构设计

http://www.lintcode.com/problem/two-sum-data-structure-design/

http://www.jiuzhang.com/solutions/two-sum-data-structure-design/

使用 HashMap 的做法

AddNumber - O(1)

FindTwoSum - O(n)

其中 FindTwoSum 函数会输入一个目标和 value

需要 for 循环 HashMap 里的每个 num，检查一下 value - num 是不是也在 HashMap 里，这个过程需要总共 O(n * 1) 的时间

代码


```java
public class TwoSum {
    private Map<Integer, Integer> counter;

    public TwoSum() {
        counter = new HashMap<Integer, Integer>();
    }

    // Add the number to an internal data structure.
    public void add(int number) {
        counter.put(number, counter.getOrDefault(number, 0) + 1);
    }

    // Find if there exists any pair of numbers which sum is equal to the value.
    public boolean find(int value) {
        for (Integer num1 : counter.keySet()) {
            int num2 = value - num1;
            int desiredCount = num1 == num2 ? 2 : 1;
            if (counter.getOrDefault(num2, 0) >= desiredCount) {
                return true;
            }
        }
        return false;
    }
}
```

```python
class TwoSum(object):

    def __init__(self):
        # initialize your data structure here
        self.count = {}

    # Add the number to an internal data structure.
    # @param number {int}
    # @return nothing
    def add(self, number):
        if number in self.count:
            self.count[number] += 1
        else:
            self.count[number] = 1

    # Find if there exists any pair of numbers which sum is e
    # @param value {int}
    # @return true if can be found or false
    def find(self, value):
        for num in self.count:
            if value - num in self.count and \
                (value - num != num or self.count[num] > 1):
                return True
        return False
```

使用 Two Pointers 的做法

AddNumber - O(n)

FindTwoSum - O(n)

其中 AddNumber 可以使用 Insertion Sort 的方法

先将数加到数组的末尾，然后一直往前交换到它的所在位置

代码


```java
public class TwoSum {
    public List<Integer> nums;
    public TwoSum() {
        nums = new ArrayList<Integer>();
    }

    public void add(int number) {
        nums.add(number);
        int index = nums.size() - 1;
        while (index > 0 && nums.get(index - 1) > nums.get(index)) {
            int temp = nums.get(index);
            nums.set(index, nums.get(index - 1));
            nums.set(index - 1, temp);
            index--;
        }
    }

    public boolean find(int value) {
        int left = 0, right = nums.size() - 1;
        while (left < right) {
            int twoSum = nums.get(left) + nums.get(right);
            if (twoSum < value) {
                left++;
            } else if (twoSum > value) {
                right--;
            } else {
                return true;
            }
        }
        return false;
    }
}
```

```python
class TwoSum:

    def __init__(self):
        self.nums = []

    def add(self, number):
        self.nums.append(number)
        index = len(self.nums) - 1
        while index > 0 and self.nums[index - 1] > self.nums[index]:
            temp = self.nums[index - 1]
            self.nums[index - 1] = self.nums[index]
            self.nums[index] = temp
            index -= 1

    def find(self, value):
        left, right = 0, len(self.nums) - 1
        while left < right:
            two_sum = self.nums[left] + self.nums[right]
            if two_sum < value:
                left += 1
            elif two_sum > value:
                right -= 1
            else:
                return True
        return False
```

一边增加数一边保持数组有序?

A: Binary Search + Array Insert

B: Binary Search + Linked List Insert

C: Heap

D: TreeMap(红黑树)

Binary Search + Array Insert

Binary Search 是 O(LogN) 没错

但是 Array Insert 会耗费 O(N) 的时间复杂度

如 [1,3,4,5,6,7] 中插入 0

Binary Search + Linked List Insert

Binary Search 是基于数组的算法，不是基于链表的算法

数组 Array - 连续型存储 - 支持 O(1) Index Access

链表 Linked List - 离散型存储 - 支持 O(n) Index Access

链表中插入一个元素虽然是 O(1) 的，但是找到插入位置需要花 O(n) 的时间

Heap (ProrityQueue)

堆是一个树状结构，堆内部的元素组织顺序不是有序的

从堆里从小到大拿出 n 个元素，需要 O(nlogn) 的时间

所以堆可以实现 O(logN) 的插入，但是对于 FindTwoSum 的方法，是需要基于一个 Sorted Array 进行的，堆无法完成。

TreeMap (红黑树)

TreeMap是一个树状结构

O(N) 时间 中序遍历之后可以得到一个有序数组

TreeMap 可以实现 O(logN) AddNumber 和 O(N) 的 FindTwoSum

但是依然不如 HashMap 的方法好

3数之和

https://www.lintcode.com/problem/3sum/

https://www.jiuzhang.com/solutions/3sum/

统计所有的和为 0 的三元组 (Triples)

代码


```python
def threeSum(self, nums):
    nums = sorted(nums)

    results = []
    for i in range(len(nums)):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        self.find_two_sum(nums, i + 1, len(nums) - 1, -nums[i], results)

    return results

def find_two_sum(self, nums, left, right, target, results):
    last_pair = None
    while left < right:
        if nums[left] + nums[right] == target:
            if (nums[left], nums[right]) != last_pair:
                results.append([-target, nums[left], nums[right]])
            last_pair = (nums[left], nums[right])
            right -= 1
            left += 1
        elif nums[left] + nums[right] > target:
            right -= 1
        else:
            left += 1
```

```java
public List<List<Integer>> threeSum(int[] numbers) {
    Arrays.sort(numbers);

    List<List<Integer>> results = new ArrayList();
    for (int i = 0; i < numbers.length; i++) {
        if (i != 0 && numbers[i] == numbers[i - 1]) {
            continue;
        }
        findTwoSum(numbers, i, results);
    }

    return results;
}

private void findTwoSum(int[] nums, int index, List<List<Integer>> results) {
    int left = index + 1, right = nums.length - 1;
    int target = -nums[index];

    while (left < right) {
        int twoSum = nums[left] + nums[right];
        if (twoSum < target) {
            left++;
        } else if (twoSum > target) {
            right--;
        } else {
            List<Integer> triple = new ArrayList();
            triple.add(nums[index]);
            triple.add(nums[left]);
            triple.add(nums[right]);
            results.add(triple);
            left++;
            right--;
            while (left < right && nums[left] == nums[left - 1]) {
                left++;
            }
        }
    }
}
```

三角形个数

https://www.lintcode.com/problem/triangle-count/

https://www.jiuzhang.com/solutions/triangle-count/

代码


```python
class Solution:
    """
    @param S: A list of integers
    @return: An integer
    """
    def triangleCount(self, S):
        S.sort()

        ans = 0
        for i in range(len(S)):
            left, right = 0, i - 1
            while left < right:
                if S[left] + S[right] > S[i]:
                    ans += right - left
                    right -= 1
                else:
                    left += 1
        return ans
```

```java
public int triangleCount(int S[]) {
    int left = 0, right = S.length - 1;
    int ans = 0;
    Arrays.sort(S);
    for (int i = 0; i < S.length; i++) {
        left = 0;
        right = i - 1;
        while (left < right) {
            if (S[left] + S[right] > S[i]) {
                ans = ans + (right - left);
                right --;
            } else {
                left ++;
            }
        }
    }
    return ans;
}
```

求具体方案 vs 求方案数

求具体方案只能一个个数出来，时间复杂度 O(n^3)

求不可以重复的方案数也只能一个个数

求可以重复的方案数可以批量累加，时间复杂度 O(n^2)

4数之和

https://www.lintcode.com/problem/4sum/

https://www.jiuzhang.com/solutions/4sum

在数组中求 a + b + c + d = target 的所有四元组

来自4个数组的4数之和

https://www.lintcode.com/problem/4sum-ii/

https://www.jiuzhang.com/solutions/4sum-ii

在4个数组中，分别取4个数，使得和为 target

求满足条的四元组个数

代码


```python
def fourSumCount(self, A, B, C, D):
    counter = {}
    for a in A:
        for b in B:
            counter[a + b] = counter.get(a + b, 0) + 1
    answer = 0
    for c in C:
        for d in D:
            answer += counter.get(-c - d, 0)
    return answer
```

```java
public int fourSumCount(int[] A, int[] B, int[] C, int[] D) {
    Map<Integer, Integer> counter = new HashMap<>();
    for (int i = 0; i < A.length; i++) {
        for (int j = 0; j < B.length; j++) {
            int sum = A[i] + B[j];
            counter.put(sum, counter.getOrDefault(sum, 0) + 1);
        }
    }

    int answer = 0;
    for (int i = 0; i < C.length; i++) {
        for (int j = 0; j < D.length; j++) {
            int sum = C[i] + D[j];
            answer += counter.getOrDefault(-sum, 0);
        }
    }
    return answer;
}
```

k数之和

https://www.lintcode.com/problem/k-sum/description 求方案总数

https://www.lintcode.com/problem/k-sum-ii/description 求具体方案

敬请期待在动态规划和深度优先搜索中对这两个问题的讲解

相关题

统计所有和 <= target 的配对数

http://www.lintcode.com/problem/two-sum-less-than-or-equal-to-target/

http://www.jiuzhang.com/solutions/two-sum-less-than-or-equal-to-target/

统计所有和 >= target 的配对数

http://www.lintcode.com/en/problem/two-sum-greater-than-target/

http://www.jiuzhang.com/solutions/two-sum-greater-than-target/

两数之和本质不同的方案数

统计所有和 >= target 的配对数

休息一会儿

Take a break

分割数组

https://www.lintcode.com/problem/partition-array/

https://www.jiuzhang.com/solutions/partition-array/

数组严格的分为 < k 的部分和 >= k 的部分

Partition Array vs Quick Sort / Quick Select


```
while (left <= right) {
    while (left <= right && nums[left] < k) {
        left++;
    }
    while (left <= right && nums[right] >= k) {
        right--;
    }

    if (left <= right) {
        int temp = nums[left];
        nums[left] = nums[right];
        nums[right] = temp;

        left++;
        right--;
    }
}
```

Partition Array

```
while (left <= right) {
    while (left <= right && A[left] < pivot) {
        left++;
    }
    while (left <= right && A[right] > pivot) {
        right--;
    }

    if (left <= right) {
        int temp = A[left];
        A[left] = A[right];
        A[right] = temp;

        left++;
        right--;
    }
}
```

Quick Sort / Quick Select

目标不同

Partition Array 需要严格的左半部分 < k，右半部分 >= k

Quick Sort / Quick Select 只需要左半部分整体 <= 右半部分即可

如果 Quick Sort / Quick Select 把 == pivot 的严格划分到左边或者右边，会导致极端情况发生从而时间复杂度很容易退化到 O(n^2)

如排序 [1, 1, 1, 1, 1]

Partition类双指针算法模板


```
while left <= right:
    while left <= right and nums[left] 应该在左侧:
        left += 1
    while left <= right and nums[right] 应该在右侧:
        right -= 1

    if left <= right:
        # 找到了一个不该在左侧的和不该在右侧的，交换他们
        nums[left], nums[right] = nums[right], nums[left]
        left += 1
        right -= 1
```

为什么用 left <= right 而不是 left < right?

left <= right

如果用 left < right, while 循环结束在 left == right

此时需要多一次 if 一句判断 nums[left] 到底是 < k 还是 >=k

因此使用 left <= right 可以省去这个判断

独孤九剑 —— 破鞭式

时间复杂度与最内层循环主体的执行次数有关

与有多少重循环无关

交替正负数

http://www.lintcode.com/problem/interleaving-positive-and-negative-numbers/

http://www.jiuzhang.com/solutions/interleaving-positive-and-negative-numbers/

将一个数组中的元素正负交替排列

数据确保正负数个数相差不超过1

不使用额外空间(do it in-place)

相关题 Related Questions

- **Partition Array by Odd and Even**
- http://www.lintcode.com/problem/partition-array-by-odd-and-even/
- http://www.jiuzhang.com/solutions/partition-array-by-odd-and-even/

- **Sort Letters by Case**
- http://www.lintcode.com/problem/sort-letters-by-case/
- http://www.jiuzhang.com/solutions/sort-letters-by-case/

排颜色

http://www.lintcode.com/problem/sort-colors/

http://www.jiuzhang.com/solutions/sort-colors/

题目要求不能使用计数排序(Counting Sort)

彩虹排序

https://www.lintcode.com/problem/sort-colors-ii/

https://www.jiuzhang.com/solutions/sort-colors-ii/

猜时间复杂度

O(n*k)

O(n^k)

O(nlogk)

O(klogn)

O(nlogn)

其他有趣的排序

烙饼排序 Pancake Sort（有可能会考哦）

https://en.wikipedia.org/wiki/Pancake_sorting

http://www.geeksforgeeks.org/pancake-sorting/

睡眠排序 Sleep Sort

https://rosettacode.org/wiki/Sorting_algorithms/Sleep_sort

面条排序 Spaghetti Sort

https://en.wikipedia.org/wiki/Spaghetti_sort

猴子排序 Bogo Sort

https://en.wikipedia.org/wiki/Bogosort

移动零

http://www.lintcode.com/problem/move-zeroes/

http://www.jiuzhang.com/solution/move-zeroes

将数组中 0 元素移动到数组的后半部分

确保数组的“修改”次数最少

两种问法

如果不需要维持原来数组中元素的相对顺序，最优算法是什么？

如果需要维持原来数组的相对顺序，最优算法是什么？

错误的解法

- 使用交换的方式无法使得“写”操作最少
- 如 [0,1,0,2,3] 中的 1 一开始会被改为0，然后会被改为2，产生了一次浪费的写操作

```
class Solution:
    """
    @param nums: an integer array
    @return: nothing
    """
    def moveZeroes(self, nums):
        left, right = 0, 0
        while right < len(nums):
            if nums[right] != 0:
                nums[left], nums[right] = nums[right], nums[left]
                left += 1
            right += 1
```

正确的解法


```python
class Solution:
    """
    @param nums: an integer array
    @return: nothing
    """
    def moveZeroes(self, nums):
        left, right = 0, 0
        while right < len(nums):
            if nums[right] != 0:
                if left != right:
                    nums[left] = nums[right]
                left += 1
            right += 1

        while left < len(nums):
            if nums[left] != 0:
                nums[left] = 0
            left += 1
```

```java
public void moveZeroes(int[] nums) {
    // 将两个指针先指向数组头部
    int left = 0, right = 0;
    while (right < nums.length) {
        // 遇到非0数赋值给新数组指针指向的位置
        if (nums[right] != 0) {
            nums[left] = nums[right];
            // 将left向后移动一位
            left++;
        }
        right++;
    }

    // 若新数组指针还未指向尾部，将剩余数组赋值为0
    while (left < nums.length) {
        nums[left] = 0;
        left++;
    }
}
```

同向双指针

请在第 6 周的互动课章节中学习

Thank You

Q & A