宫水三叶的刷题日记

分治

Author ：宫水三叶
Date ：2021/10/07
QQ Group: 703311589
WeChat ：oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「分治」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「分治」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「分治」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **4. 寻找两个正序数组的中位数** ，难度为 **困难**。

Tag：「二分」、「分治」

给定两个大小分别为 m 和 n 的正序（从小到大）数组 nums1 和 nums2。

请你找出并返回这两个正序数组的「中位数」。

示例 1：

```
输入：nums1 = [1,3], nums2 = [2]
输出：2.00000
解释：合并数组 = [1,2,3] ，中位数 2
```

示例 2：

```
输入：nums1 = [1,2], nums2 = [3,4]
输出：2.50000
解释：合并数组 = [1,2,3,4] ，中位数 (2 + 3) / 2 = 2.5
```

示例 3：

```
输入：nums1 = [0,0], nums2 = [0,0]
输出：0.00000
```

示例 4：

```
输入：nums1 = [], nums2 = [1]
输出：1.00000
```

示例 5：

```
输入：nums1 = [2], nums2 = []
输出：2.00000
```

提示：

- nums1.length == m
- nums2.length == n
- 0 <= m <= 1000
- 0 <= n <= 1000
- 1 <= m + n <= 2000
- -10^6 <= nums1[i], nums2[i] <= 10^6

进阶：你能设计一个时间复杂度为 O(log (m+n)) 的算法解决此问题吗？

朴素解法

如果忽略进阶的 $O(\log (m+n))$ 要求，这道题就非常简单。

一个比较直观的做法：将两个数组合并，排序，然后分别取得 `total / 2` 和

`(total - 1) / 2` 两个位置的数，取两者平均值。

这样做的目的是为了避免分情况讨论：合并后的数组长度是奇数还是偶数。

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int n = nums1.length, m = nums2.length;
        int[] arr = new int[n + m];
        int idx = 0;
        for (int i : nums1) arr[idx++] = i;
        for (int i : nums2) arr[idx++] = i;
        Arrays.sort(arr);
        int l = arr[(n + m) / 2], r = arr[(n + m - 1) / 2];
        return (l + r) / 2.0;
    }
```

- 时间复杂度：合并两个数组的复杂度是 $O(m+n)$，对合并数组进行排序的复杂度是 $O((m+n)\log(m+n))$。整体复杂度是 $O((m+n)\log(m+n))$
- 空间复杂度：$O(1)$

注意：`Arrays.sort()` 不只有双轴快排实现，这里的复杂度分析是假定其使用双轴快排。

分治解法

首先可以将原问题等效为：从两个有序数组中找第 k 小的数。

分两种情况讨论：

1. 总个数为偶数：找到 `第 (total / 2) 个小的数` 和 `第 (total / 2 + 1) 个小的数`，结果为两者的平均值。
2. 总个数为奇数：结果为 `第 (total / 2 + 1) 个小的数` 。

具体思路为：

- 默认第一个数组比第二个数组的有效长度短，如果不满足，则调换两个数组（这也是一个常用技巧，目的是减少边界处理工作：原本需要对两个数组做越界检查，现在只需要对短的数组做越界检查）
- 第一个数组的有效长度从 `i` 开始，第二个数组的有效长度从 `j` 开始，其中 `[i,si - 1]` 是第一个数组的前 `k / 2` 个元素，`[j,sj - 1]` 是第二个数组的前

`k - k / 2` 个元素（为了确保 k 为奇数的时候正确）

- 当 `nums1[si - 1] > nums2[sj - 1]`：则表示第 `k` 小一定不在 `[j,sj - 1]` 中，即在 `[i,n]` 或 `[sj,m]` 中
- 当 `nums1[si - 1] <= nums2[sj - 1]`：则表示第 `k` 小一定不在 `[i,si - 1]` 中，即在 `[si,n]` 或 `[j,m]` 中

```
class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int tot = nums1.length + nums2.length;
        if (tot % 2 == 0) {
            int left = find(nums1, 0, nums2, 0, tot / 2);
            int right = find(nums1, 0, nums2, 0, tot / 2 + 1);
            return (left + right) / 2.0;
        } else {
            return find(nums1, 0, nums2, 0, tot / 2 + 1);
        }
    }
    int find(int[] n1, int i, int[] n2, int j, int k) {
        if (n1.length - i > n2.length - j) return find(n2, j, n1, i, k);
        if (i >= n1.length) return n2[j + k - 1];
        if (k == 1) {
            return Math.min(n1[i], n2[j]);
        } else {
            int si = Math.min(i + (k / 2), n1.length), sj = j + k - (k / 2);
            if (n1[si - 1] > n2[sj - 1]) {
                return find(n1, i, n2, sj, k - (sj - j));
            } else {
                return find(n1, si, n2, j, k - (si - i));
            }
        }
    }
}
```

- 时间复杂度：每次递归 k 的规模都减少一半，复杂度为 $O(\log (m+n))$
- 空间复杂度：$O(1)$

总结

今天这道题，我给你介绍了两种技巧：

1. 在机试或者竞赛中，目的是尽可能快的 AC，所以 Java 可以直接不写 `private` 的

修饰符（不写代表使用默认的包权限），这没有问题，不用纠结

2. 在机试或者竞赛中，遇到一些是从文字上限制我们的题目，例如本题限制我们使用 $O(\log (m+n))$ 算法。可以分析是否能够不按照限制要求来做，具体分析思路为：
    1. 先有一个很容易实现的算法思路。例如本题很容易就想到直接使用双指针找第 k 个小的数，复杂度为 $O(n)$。
    2. 看题目的数据规模 ① 是否支撑我们使用限制以外的算法。例如本题数据规模只有 1000 + 1000 = 2000。
    3. 根据数据规模，判断我们的朴素算法计算机是否可以在 1s 内处理完 ②，即判断运算次数是否在 10^7 以内 ③ 。例如本题使用双指针算法，指针移动和判断大小算一次运行，由于数据只有 2000，距离 10^7 还很远，所以完全足够了

说明 ①：正规的算法题目都会提供数据规模，LeetCode 上一些旧题目没有提供，是因为当时出的时候不太规范，LeetCode 新题、其他 OJ 平台题目，算法竞赛题目都会有。

说明 ②：即使是最严格的 OJ 中最简单的题目，也会提供 1s 的运行时间，超过这个时间才算超时。

说明 ③：计算器 1s 内极限的处理速度是 10^8 ，但为了尽可能不出现错误提交，使用技巧时尽量和 10^7 进行比较。

注意：这两个技巧，我只推荐在机试或者竞赛（尽可能快 AC 的场景）中使用。平时练习或者和面试的时候必须老实按照题目要求来。

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

💡**更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。**

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「分治」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：

"给作者手机充个电"

YOLO 的赞赏码