宫水三叶的刷题日记



Author: 宮水三叶 Date : 2021/10/07 QQ Group: 703311589 WeChat: 0a0aya

刷题自治

公众号: 宫水之叶的刷题日记

**@ 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「三分」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「三分」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「三分」;
- 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 852. 山脉数组的峰顶索引 ,难度为 简单。

Tag:「二分」、「三分」

符合下列属性的数组 arr 称为 山脉数组 :

- arr.length >= 3
- 存在 i (0 < i < arr.length 1) 使得:
 - o arr[0] < arr[1] < ... arr[i-1] < arr[i]</pre>
 - o arr[i] > arr[i+1] > ... > arr[arr.length 1]

给你由整数组成的山脉数组 arr ,返回任何满足

arr[0] < arr[1] < ... arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

的下标i。

示例 1:

输入:arr = [0,1,0]

输出:1

示例 2:

输入: arr = [0,2,1,0]

输出:1

示例 3:

输入:arr = [0,10,5,2]

输出:1

示例 4:

输入:arr = [3,4,5,1]

输出:2

示例 5:

输入:arr = [24,69,100,99,79,78,67,36,26,19]

输出:2

提示:

- 3 <= arr.length <= 10^4
- $0 \le \arcsin[i] \le 10^6$
- 题目数据保证 arr 是一个山脉数组

进阶:很容易想到时间复杂度 O(n) 的解决方案,你可以设计一个 O(log(n)) 的解决方案吗?

二分

往常我们使用「二分」进行查值[,]需要确保序列本身满足「二段性」:当选定一个端点(基准值)后,结合「一段满足&另一段不满足」的特性来实现"折半"的查找效果。

但本题求的是峰顶索引值,如果我们选定数组头部或者尾部元素,其实无法根据大小关系"直接" 将数组分成两段。

但可以利用题目发现如下性质:由于 arr 数值各不相同,因此峰顶元素左侧必然满足严格单调 递增,峰顶元素右侧必然不满足。

因此 以峰顶元素为分割点的 arr 数组,根据与 前一元素/后一元素 的大小关系,具有二段性:

- ・ 峰顶元素左侧满足 arr[i-1] < arr[i] 性质,右侧不满足
- ・ 峰顶元素右侧满足 arr[i] > arr[i+1] 性质,左侧不满足

因此我们可以选择任意条件,写出若干「二分」版本。

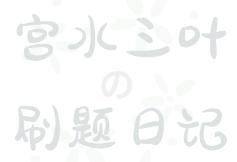
代码:

```
class Solution {
   // 根据 arr[i-1] < arr[i] 在 [1,n-1] 范围内找值
    // 峰顶元素为符合条件的最靠近中心的元素
    public int peakIndexInMountainArray(int[] arr) {
        int n = arr.length;
        int l = 1, r = n - 1;
        while (l < r) {
           int mid = l + r + 1 >> 1;
           if (arr[mid - 1] < arr[mid]) {</pre>
               l = mid;
            } else {
                r = mid - 1;
        }
        return r;
   }
}
```



公众号: 宫水三叶的刷题日记

```
class Solution {
   // 根据 arr[i] > arr[i+1] 在 [0,n-2] 范围内找值
   // 峰顶元素为符合条件的最靠近中心的元素值
   public int peakIndexInMountainArray(int[] arr) {
       int n = arr.length;
       int l = 0, r = n - 2;
       while (l < r) {
           int mid = l + r \gg 1;
           if (arr[mid] > arr[mid + 1]) {
               r = mid;
           } else {
               l = mid + 1;
       }
       return r;
   }
}
```



公众号: 宫水之叶的刷题日记

```
class Solution {
   // 根据 arr[i] < arr[i+1] 在 [0,n-2] 范围内找值
   // 峰顶元素为符合条件的最靠近中心的元素的下一个值
   public int peakIndexInMountainArray(int[] arr) {
        int n = arr.length;
       int l = 0, r = n - 2;
       while (l < r) {
           int mid = l + r + 1 >> 1;
           if (arr[mid] < arr[mid + 1]) {</pre>
               l = mid;
           } else {
               r = mid - 1;
           }
       return r + 1;
   }
}
```

・ 时间复杂度: $O(\log n)$

・空间复杂度:O(1)

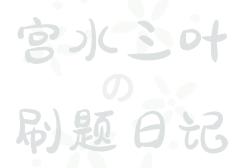
三分

事实上,我们还可以利用「三分」来解决这个问题。

顾名思义[,]「三分」就是使用两个端点将区间分成三份,然后通过每次否决三分之一的区间来逼 近目标值。

具体的,由于峰顶元素为全局最大值,因此我们可以每次将当前区间分为 [l,m1]、[m1,m2] 和 [m2,r] 三段,如果满足 arr[m1]>arr[m2],说明峰顶元素不可能存在与 [m2,r] 中,让 r=m2-1 即可。另外一个区间分析同理。

代码:



公众号: 宫水之叶的刷题日记

・ 时间复杂度: $O(\log_3 n)$

・ 空间复杂度:O(1)

二分&三分&k分?

今天比较忙,没有及时回复评论。

看到了评论区有不少关于「三分」否决 1/3 区间存在疑问,感觉挺有代表性的,补充到题解好了 🤣

但必须说明一点,「二分」和「三分」在渐进复杂度上都是一样的,都可以通过换底公式转化为可忽略的常数,因此两者的复杂度都是 $O(\log n)$ 。

因此选择「二分」还是「三分」取决于要解决的是什么问题:

- 二分通常用来解决单调函数的找 *target* 问题,但进一步深入我们发现只需要满足「二段性」就能使用「二分」来找分割点;
- 三分则是解决单峰函数极值问题。

因此一般我们将「通过比较两个端点,每次否决 1/3 区间 来解决单峰最值问题」的做法称为 「三分」;而不是简单根据单次循环内将区间分为多少份来判定是否为「三分」。

随手写了一段反例代码:

```
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        int left = 0, right = arr.length - 1;
        while(left < right) {</pre>
            int m1 = left + (right - left) / 3;
            int m2 = right - (right - left + 2) / 3;
            if (arr[m1] > arr[m1 + 1]) {
                 right = m1;
            } else if (arr[m2] < arr[m2 + 1]) {</pre>
                 left = m2 + 1;
            } else {
                 left = m1;
                 right = m2;
            }
        return left;
    }
}
```

这并不是「三分」做法,最多称为「变形二分」。本质还是利用「二段性」来做分割的,只不过同时 check 了两个端点而已。

如果这算「三分」的话,那么我能在一次循环里面划分 k-1 个端点来实现 k 分?

显然这是没有意义的,因为按照这种思路写出来的所谓的「四分」、「五分」、「k分」是需要增加同等数量的分支判断的。这时候单次 while 决策就不能算作 O(1) 了,而是需要在 O(k) 的复杂度内决定在哪个分支,就跟上述代码有三个分支进行判断一样。 因此,这种写法只能算作是「变形二分」。

综上,只有「二分」和「三分」的概念,不存在所谓的 k 分。 同时题解中的「三分」部分提供的做法就是标准的「三分」做法。

** 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「三分」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

版权声明:任何形式的转载请保留出处 Wiki。