CSI讲义10:寻找峰值

> 本文试图从一个简单的小题目出发，引入算法的若干基本概念，重点引入一种方法：分治法，并且给出表述算法效率的记号。本文展示了算法教与学的一种常用范式，建议新生遵循这种思路进行学习。阅读本文内容不需要任何大学知识为基础，阅读大概需要一个小时。

题目：寻找峰值

任意给出一组数值，比如：9、12、3、7，13。如果某一个数值同时比它左边与右边的数值都大，则这个数值就是一个峰值。直观上，你可以想象一个山峰的形象。当然，如果某个数在数列的最左边，如果它比它右边的值要大，那么这个数值也是峰值。同理，如果某个数在数列的最右边，如果它比它左边的值大，它也是峰值。在上面给出的数列中，12是一个峰值，同时，13也是峰值。

问题是：如何能在一个n长的数列中找到**任意**一个峰值？要求是，这种方法应该尽可能快。

最普通的解法

因为有n个数值，可以看为放在n个盒子中的数字，如下表：

| a1 |a2| a3 | a4 |...| an |

设想这样一个过程：从第一个数值a1开始，比对a1与其左右两边的数值，如果满足条件，则输出a1的值，结束过程；否则，继续比对a2 。不断迭代地做下去，直得最后一个数值an也被比对过。

这个过程要不就输出某个峰值，要不就比对完所有的数值，找不到结果而停止。必须强调，这是一个必然会终止的过程，无论它是否找到满足条件的峰值。

类似这样，为了完成某个特定的任务而形成的（迟早会终止的）指令系列，称之为**算法**。设计出算法后，通常我们要考虑两个问题：算法的正确性与算法的复杂性。

算法的正确性

算法的**正确性**考虑：算法的执行是否总能给出正确答案。当然并非所有算法的正确性都显而易见。幸运的是，以上方法的正确性很显然。

算法的复杂性

算法的**复杂性**指的是算法执行的**效率**。大致上，衡量效率是看算法从开始到结束共需要执行多少指令。因为指令执行需要时间，所以也往往将效率理解为算法执行所需的**时间**。考虑"寻找峰值"问题的以上解法。最理想的状态是，比对完a1，发现它就是峰值，算法结束，执行了一个比对的过程。最糟糕的状态是，需要比对完an，无论an是否我们想要的，都执行了n次比对过程。那么，在衡量算法的效率时，答案到底是1还是n。

其实，这里时间1是算法的**下界**，即算法执行最少需要的时间，记号是Big-Omega。n是算法的*上界*，即算法最多要用的执行时间，记号是Big O，简记为O(n)。本文暂时不考虑Omega，只考虑Big O。这里我们牺牲了很多的技术细节，以后再补充。现在我们只需要知道，如果算法最多执行n步，就记为O(n)。

此处忽略很多细节，具体可参考课本CSI的第587页的讲解。

我们接下来需要思考的问题就是，要完成任务真的最多需要执行n步吗？最多执行n/2步可以吗？或者，log n步呢？（注，log指以2为底的对数。）

更合理的算法

解题思路

怎么思考更合理的解法呢？还是从以上算法出发，我们知道的是，以上算法遍历了数组中所有的元素，效率是O(n)。我们通常把这种方法称为“暴力法”，每一个元素都查询确实很暴力。要提高效率，无非就是要“不完全遍历”，非暴力。可能吗？题目要求找到一个满足要求的元素即可，也就是说你得到一个答案即可，并不需要遍历所有元素。所以，这是可能的。

其次，要不完全遍历，那么可能我们需要把所有数值分成若干部分，然后在其中某些部分进行搜索，而可以忽略掉某些部分。那么问题的关键点就在于，如何对数值进行分组？

接着就需要思考，既然分组，而且要“丢弃”某些分组，那么分组的主要目标是什么？**确保所得的某个分组中存在至少一个峰值**，然后针对这个存在解的分组进行峰值的寻找，忽略其他分组。比如，如果把所有元素分成左右两部分，如果左边部分必然存在一个峰值，那么右边部分就可以不管了，因为我们的目标是找到一个峰值。

如果第一次分组可以达到目标-- 存在一个解，那么可以对分组中的元素再进行分组，不断进行下去，总能得到解。尽管我们现在还不知道如何分组，但是我们可以分析一下，如果可以这样做，效率是可以极大提高的。为什么呢？

给一个直观，比如查字典。我们是一页一页地查，还是把字典分成两半，再分成两半地不断去查？当然，这里还没有数学，我们接下来需要更数学化的表示。你知道一页一页查的复杂性是多少吗？而分半再分半的复杂性呢？

总之，分析到现在，我们整个解题思路就聚焦在这样一个目标：对所有数值分组，并确保所得的某一个分组中存在至少一个峰值。

提示：分两组，并确保其中一组中存在峰值。如何做？

分治法

以上策略就是把需要处理的数据分成不同的部分，然后逐个部分进行处理，从而得到更好的效率。我们把这种策略称为“**分治法(Divide-and-Conquer)**”。

分治法是算法策略中的一种重要方法，许多重要算法都体现了“**分而治之**”的威力，比如：二分查找、快速排序等。关于分治法，可以通过学习更多的算法来加深体会。

算法描述

正确的做法非常简单： 1、算出数组下标的中值，取中间元素（记为mid）进行判断；数组元素被mid分成两部分； 2、如果mid小于其左边的元素，则**递归**地在左边部分进行查找； 3、如果mid小于其右边的元素，则递归在右边部分查找； 4、否则mid就是答案。

伪代码描述如下：

```
# 请注意，以下并非一个正确的、完整的Python代码，是伪代码，仅用于说明算法的过程。
# “#”后的文字代表注释。
n = 100000   #假如有100000个元素
# List 里面装了所有的数值，看成n个盒子
# head和tail分别是数值的第一个下标与最后一个下标
def find_peak(List, head, tail):
    i = (head + tail) / 2    #取中值，i是index的缩写
    if  List[i] < List[i - 1]  :
        find_peak(List, head, i - 1) #递归地在左半部分寻找峰值
    elseif List[i] < List[i + 1]:
        find_peak(List, i + 1, tail)  #递归地在右半部分寻找峰值
    else:
        return List[i]       #已经找到答案，返回
```

建议阅读者使用C语言实现该算法。需要完善的是边界下标的判断。比如，如果下标i已经是0的时候，下标 i 的左边是什么？没有东西了，怎么办？同理，当 i 跑到了右边尽头也一样要处理。改进为**迭代算法**更佳。

算法的正确性

必须承认，现在要说明算法的正确性就不那么显然了。除了要分析、说明以外，最值得信赖的方法是**证明**。

分析

先分析一下为什么算法会正确。首先看第4步，算法能执行到第4步说明mid比它的左右邻居都大，当然是答案。然后看第二步，如果算法执行第2步的递归，说明第三步无需执行，必然在左边的分组中会存在峰值。为什么？第三步的分析类似，不赘述。现在回答这个为什么。

如果mid比左边元素小，则递归进行左边部分的搜索。要说明这种做法的正确性，只需要说明，此时在左边部分确实会至少存在一个峰值。这也是第三步无需执行的原因。假定mid比左边元素（记为a）小，此时分析两种情形：

- a比它的左边大，那么a就是峰值；
- a比它的左边小，此时的情形与我们正在分析的mid的情形相同，但是数组长度少了一。

分析到此，让我们设想一下，如果左边只有a和mid两个元素会如何？即a是数组的最左边的元素会如何？回答：a是峰值。这个结论与上面两点结合起来，答案呼之欲出。

头脑风暴结束，建议大家在草稿纸上比划比划，然后回到正确的思路--证明--上来。

归纳法证明

先说一句题外话。在过去的几年中，我发现大一新生对归纳法证明非常不感兴趣，在他们看来归纳法非常“套路”，非常没有意思，甚至会认为“不科学”、不是证明。要改变这种状况，也许需要更多的工作。真心希望中学里面学归纳法没有学坏胃口。归纳法证明是一种非常值得学习和使用的方法。

证明： 1、归纳基础 数组有1个元素，且mid比该数组最右边（或者左边，以下省略左边不写）的元素小，则存在一个峰值。显然！

2、归纳假设 假设数组有n个元素，且mid比该数组最右边元素小，则该数组必有峰值。

3、归纳证明“数组有n+1个元素，且mid比该数组最右边元素小，则该数组必有峰值”

忽略若干细节，请同学们自己完成。

算法的复杂性

到了最困难的撰写部分。以下内容如果引起新生的不适，可暂时忽略。

用一个函数T(n)来衡量算法的效率，其中n是输入数值的数量，在本文就是数组的大小。立即可以知道：

```
T(0) = 0   #没有输入，不要花时间
T(1) = c   #c代表常数时间
T(n) = T(n/2) + c
#每次算法执行了一下对比的过程，使用c时间，然后进入递归
#递归只针对一半的元素，所以是T(n/2)
```

现在需要算出函数T的具体数值表达。没有什么技巧，只需要不断地做除法：

```
T(n) = T(n/2) + c
T(n/2) = T((n/2)/2)+ c
T((n/2)/2) = T(((n/2)/2)/2) +c
...   #这里代表不断地除，并且假设n是2的某个幂次。
T(1) = c
```

然后，立即得到：（能看出来为什么吗？）

```
T(n) = c + c + ... + c
```

最后一步，这里到底有多少个c？回忆一下十进制的二进制表达。并联想到，除2只是左移一位。那么能“左移”多少次呢？答案：log n 次。所以：

```
T(n) = log n * c = O(log n)
```

进一步的考虑

如果我们把问题进一步扩展，不考虑一维数组，而是考虑二维数组。在二维数组中的峰值，直观上就是一个数值，它比它的上下左右的数值都要大。在二维数组中求峰值，算法应该如何设计？复杂性是多少？

留给同学们思考练习。

关于算法的学习建议

有许多关于算法的基础知识值得立即学习，这些内容非常经典，无论是教材还是网络上的电子资料和代码都非常多，因此不再继续以下专题的写作。建议专题如下：

- 二分查找法
- 排序算法：插入排序、归并排序、快速排序
- 数据结构相关：队列、堆栈、二叉树
- 堆与堆排序
- 二叉查找树
- 图与图遍历算法
- 动态规划基础
- 贪心法

建议新生立即就以上专题制定学习进度计划。

课本CSI第7章有讲相关内容。缺点是缺乏相应的训练，内容广度、深度都不够，需要进一步阅读。个人认为，在大一第一学期及时开展相关内容的学习是可行的，而且强烈建议进行。学习这些是为了尽快开展CLRS的阅读，建议大一寒假进行。

有同学会问，我现在编程基础都没有，学算法会不会太冒进？回答：不会。**算法与程序应该同时学习、训练**，没有算法的程序就是没有灵魂的程序，营养不足。总是训练写没有营养的程序，程序能力不能很好地提高。反过来，学算法不考虑实现，不知道算法与程序之间的差异，也不能真正领悟算法的核心本质。

辅助资料

- 计算机科学的基础
- MIT 6.006: Introduction to Algorithms

2017年8月25日起草 2017年8月27日定稿