

讲堂 > 数据结构与算法之美 > 文章详情

16 | 二分查找（下）：如何快速定位IP对应的省份地址？

2018-10-26 王争



16 | 二分查找（下）：如何快速定位IP对应的省份地址？

朗读人：修阳 11'47" | 5.40M

通过 IP 地址来查找 IP 归属地的功能，不知道你有没有用过？没用过也没关系，你现在可以打开百度，在搜索框里随便输一个 IP 地址，就会看到它的归属地。



这个功能并不复杂，它是通过维护一个很大的 IP 地址库来实现的。地址库中包括 IP 地址范围和归属地的对应关系。

当我们想要查询 202.102.133.13 这个 IP 地址的归属地时，我们就在地址库中搜索，发现这个 IP 地址落在 [202.102.133.0, 202.102.133.255] 这个地址范围内，那我们就可以将这个 IP 地址范围对应的归属地“山东东营市”显示给用户了。

```
1 [202.102.133.0, 202.102.133.255] 山东东营市
2 [202.102.135.0, 202.102.136.255] 山东烟台
3 [202.102.156.34, 202.102.157.255] 山东青岛
4 [202.102.48.0, 202.102.48.255] 江苏宿迁
5 [202.102.49.15, 202.102.51.251] 江苏泰州
6 [202.102.56.0, 202.102.56.255] 江苏连云港
```

[复制代码](#)

现在我的问题是，在庞大的地址库中逐一比对 IP 地址所在的区间，是非常耗时的。假设我们有 12 万条这样的 IP 区间与归属地的对应关系，如何快速定位出一个 IP 地址的归属地呢？

是不是觉得比较难？不要紧，等学完今天的内容，你就会发现这个问题其实很简单。

上一节我讲了二分查找的原理，并且介绍了最简单的一种二分查找的代码实现。今天我们来讲几种二分查找的变形问题。

不知道你有没有听过这样一个说法：“十个二分九个错”。二分查找虽然原理极其简单，但是想要写出没有 Bug 的二分查找并不容易。

唐纳德·克努特 (Donald E.Knuth) 在《计算机程序设计艺术》的第 3 卷《排序和查找》中说到：“尽管第一个二分查找算法于 1946 年出现，然而第一个完全正确的二分查找算法实现直到 1962 年才出现。”

你可能会说，我们上一节学的二分查找的代码实现并不难写啊。那是因为上一节讲的只是二分查找中最简单的一种情况，在不存在重复元素的有序数组中，查找值等于给定值的元素。最简单的二分查找写起来确实不难，但是，二分查找的变形问题就没那么好写了。

二分查找的变形问题很多，我只选择几个典型的来讲解，其他的你可以借助我今天讲的思路自己来分析。

4种常见的二分查找变形问题

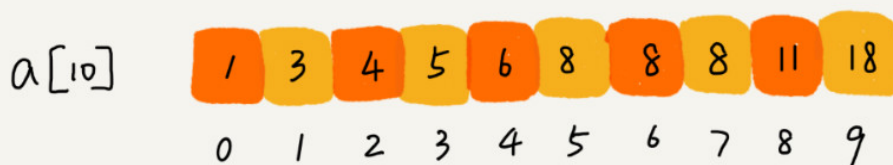
- 查找第一个值等于给定值的元素
- 查找最后一个值等于给定值的元素
- 查找第一个大于等于给定值的元素
- 查找最后一个小于等于给定值的元素

需要特别说明一点，为了简化讲解，今天的内容，我都以数据是从小到大排列为前提，如果你要处理的数据是从大到小排列的，解决思路也是一样的。同时，我希望你最好先自己动手试着写一下这 4 个变形问题，然后再看我的讲述，这样你就会对我说的“二分查找比较难写”有更加深的体会了。

变体一：查找第一个值等于给定值的元素

上一节中的二分查找是最简单的一种，即有序数据集中不存在重复的数据，我们在其中查找值等于某个给定值的数据。如果我们将这个问题稍微修改下，有序数据集中存在重复的数据，我们希望找到第一个值等于给定值的数据，这样之前的二分查找代码还能继续工作吗？


比如下面这样一个有序数组，其中， $a[5]$ ， $a[6]$ ， $a[7]$ 的值都等于 8，是重复的数据。我们希望查找第一个等于 8 的数据，也就是下标是 5 的元素。



如果我们用上一节课讲的分查找的代码实现，首先拿 8 与区间的中间值 $a[4]$ 比较，8 比 6 大，于是在下标 5 到 9 之间继续查找。下标 5 和 9 的中间位置是下标 7， $a[7]$ 正好等于 8，所以代码就返回了。


尽管 $a[7]$ 也等于 8，但它并不是我们想要找的第一个等于 8 的元素，因为第一个值等于 8 的元素是数组下标为 5 的元素。我们上一节讲的分查找代码就无法处理这种情况了。所以，针对这个变形问题，我们可以稍微改造一下上一节的代码。

100 个人写二分查找就会有 100 种写法。网上有很多关于变形二分查找的实现方法，有很多写得非常简洁，比如下面这个写法。但是，尽管简洁，理解起来却非常烧脑，也很容易写错。

 复制代码

```
1 public int bsearch(int[] a, int n, int value) {
2     int low = 0;
3     int high = n - 1;
4     while (low <= high) {
5         int mid = low + ((high - low) >> 1);
6         if (a[mid] >= value) {
7             high = mid - 1;
8         } else {
9             low = mid + 1;
10        }
11    }
12
13    if (low < n && a[low]==value) return low;
14    else return -1;
15 }
```

看完这个实现之后，你是不是觉得很不好理解？如果你只是死记硬背这个写法，我敢保证，过不了几天，你就会全都忘光，再让你写，90% 的可能会写错。所以，我换了一种实现方法，你看看是不是更容易理解呢？

 复制代码

```
1 public int bsearch(int[] a, int n, int value) {
2     int low = 0;
3     int high = n - 1;
4     while (low <= high) {
5         int mid = low + ((high - low) >> 1);
6         if (a[mid] > value) {
7             high = mid - 1;
8         } else if (a[mid] < value) {
9             low = mid + 1;
10        } else {
11            if ((mid == 0) || (a[mid - 1] != value)) return mid;
12            else high = mid - 1;
13        }
14    }
15    return -1;
}
```

```
16 }
```

我来稍微解释一下这段代码。a[mid] 跟要查找的 value 的大小关系有三种情况：大于、小于、等于。对于 a[mid]>value 的情况，我们需要更新 high=mid-1；对于 a[mid]<value 的情况，我们需要更新 low=mid+1。这两点都很好理解。那当 a[mid]=value 的时候应该如何处理呢？

如果我们查找的是任意一个值等于给定值的元素，当 a[mid] 等于要查找的值时，a[mid] 就是我们要找的元素。但是，如果我们求解的是第一个值等于给定值的元素，当 a[mid] 等于要查找的值时，我们就需要确认一下这个 a[mid] 是不是第一个值等于给定值的元素。

我们重点看第 11 行代码。如果 mid 等于 0，那这个元素已经是数组的第一个元素，那它肯定是要找的；如果 mid 不等于 0，但 a[mid] 的前一个元素 a[mid-1] 不等于 value，那也说明 a[mid] 就是我们要找的第一个值等于给定值的元素。

如果经过检查之后发现 a[mid] 前面的一个元素 a[mid-1] 也等于 value，那说明此时的 a[mid] 肯定不是我们要找的第一个值等于给定值的元素。那我们就更新 high=mid-1，因为要找的元素肯定出现在 [low, mid-1] 之间。

对比上面的两段代码，是不是下面那种更好理解？实际上，很多人都觉得变形的二分查找很难写，主要原因是太追求第一种那样完美、简洁的写法。而对于我们做工程开发的人来说，代码易懂、没 Bug，其实更重要，所以我觉得第二种写法更好。

变体二：查找最后一个值等于给定值的元素

前面的问题是查找第一个值等于给定值的元素，我现在把问题稍微改一下，查找最后一个值等于给定值的元素，又该如何做呢？

如果你掌握了前面的写法，那这个问题你应该很轻松就能解决。你可以先试着实现一下，然后跟我写的对比一下。

```
1 public int bsearch(int[] a, int n, int value) {
2     int low = 0;
3     int high = n - 1;
4     while (low <= high) {
5         int mid = low + ((high - low) >> 1);
6         if (a[mid] > value) {
7             high = mid - 1;
8         } else if (a[mid] < value) {
9             low = mid + 1;
10        } else {
11            if ((mid == n - 1) || (a[mid + 1] != value)) return mid;
12            else low = mid + 1;
13        }
14    }
15 }
```

[复制代码](#)

```
14    }
15    return -1;
16 }
```

我们还是重点看第 11 行代码。如果 $a[mid]$ 这个元素已经是数组中的最后一个元素了，那它肯定是要找的；如果 $a[mid]$ 的后一个元素 $a[mid+1]$ 不等于 $value$ ，那也说明 $a[mid]$ 就是我们要找的最后一个值等于给定值的元素。

如果我们经过检查之后，发现 $a[mid]$ 后面的一个元素 $a[mid+1]$ 也等于 $value$ ，那说明当前的这个 $a[mid]$ 并不是最后一个值等于给定值的元素。我们就更新 $low = mid + 1$ ，因为要找的元素肯定出现在 $[mid+1, high]$ 之间。

变体三：查找第一个大于等于给定值的元素

现在我们再来看另外一类变形问题。在有序数组中，查找第一个大于等于给定值的元素。比如，数组中存储的这样一个序列：3, 4, 6, 7, 10。如果查找第一个大于等于 5 的元素，那就是 6。

实际上，实现的思路跟前面的那两种变形问题的实现思路类似，代码写起来甚至更简洁。

```
1 public int bsearch(int[] a, int n, int value) {
2     int low = 0;
3     int high = n - 1;
4     while (low <= high) {
5         int mid = low + ((high - low) >> 1);
6         if (a[mid] >= value) {
7             if ((mid == 0) || (a[mid - 1] < value)) return mid;
8             else high = mid - 1;
9         } else {
10             low = mid + 1;
11         }
12     }
13     return -1;
14 }
```

[复制代码](#)

如果 $a[mid]$ 小于要查找的值 $value$ ，那要查找的值肯定在 $[mid+1, high]$ 之间，所以，我们更新 $low = mid + 1$ 。

对于 $a[mid]$ 大于等于给定值 $value$ 的情况，我们要先看下这个 $a[mid]$ 是不是我们要找的第一个值大于等于给定值的元素。如果 $a[mid]$ 前面已经没有元素，或者前面一个元素小于要查找的值 $value$ ，那 $a[mid]$ 就是我们要找的元素。这段逻辑对应的代码是第 7 行。

如果 $a[mid-1]$ 也大于等于要查找的值 $value$ ，那说明要查找的元素在 $[low, mid-1]$ 之间，所以，我们将 $high$ 更新为 $mid-1$ 。

变体四：查找最后一个小于等于给定值的元素

现在，我们来看最后一种二分查找的变形问题，查找最后一个小于等于给定值的元素。比如，数组中存储了这样一组数据：3，5，6，8，9，10。最后一个小于等于 7 的元素就是 6。是不是有点类似上面那一种？实际上，实现思路也是一样的。

有了前面的基础，你完全可以自己写出来了，所以我不详细分析了。我把代码贴出来，你可以写完之后对比一下。

```
1 public int bsearch7(int[] a, int n, int value) {
2     int low = 0;
3     int high = n - 1;
4     while (low <= high) {
5         int mid = low + ((high - low) >> 1);
6         if (a[mid] > value) {
7             high = mid - 1;
8         } else {
9             if ((mid == n - 1) || (a[mid + 1] > value)) return mid;
10            else low = mid + 1;
11        }
12    }
13    return -1;
14 }
```

[复制代码](#)

解答开篇

好了，现在我们回头来看开篇的问题：如何快速定位出一个 IP 地址的归属地？

现在这个问题应该很简单了。如果 IP 区间与归属地的对应关系不经常更新，我们可以先预处理这 12 万条数据，让其按照起始 IP 从小到大排序。如何来排序呢？我们知道，IP 地址可以转化为 32 位的整型数。所以，我们可以将起始地址，按照对应的整型值的大小关系，从小到大进行排序。

然后，这个问题就可以转化为我刚讲的第四种变形问题“在有序数组中，查找最后一个小于等于某个给定值的元素”了。

当我们要查询某个 IP 归属地时，我们可以先通过二分查找，找到最后一个起始 IP 小于等于这个 IP 的 IP 区间，然后，检查这个 IP 是否在这个 IP 区间内，如果在，我们就取出对应的归属地显示；如果不在，就返回未查找到。

内容小结

上一节我说过，凡是用二分查找能解决的，绝大部分我们更倾向于用散列表或者二叉查找树。即便是二分查找在内存使用上更节省，但是毕竟内存如此紧缺的情况并不多。那二分查找真的没什么用处了吗？

实际上，上一节讲的求“值等于给定值”的二分查找确实不怎么会被用到，二分查找更适合用在“近似”查找问题，在这类问题上，二分查找的优势更加明显。比如今天讲的这几种变体问题，用其他数据结构，比如散列表、二叉树，就比较难实现了。

变体的二分查找算法写起来非常烧脑，很容易因为细节处理不好而产生 Bug，这些容易出错的细节有：终止条件、区间上下界更新方法、返回值选择。所以今天的内容你最好能用自己实现一遍，对锻炼编码能力、逻辑思维、写出 Bug free 代码，会很有帮助。

课后思考

我们今天讲的都是非常规的二分查找问题，今天的思考题也是一个非常规的二分查找问题。如果有序数组是一个循环有序数组，比如 4, 5, 6, 1, 2, 3。针对这种情况，如何实现一个求“值等于给定值”的二分查找算法呢？

欢迎留言和我分享，我会第一时间给你反馈。



数据结构与算法之美

为工程师量身打造的数据结构与算法私教课

王争
前 Google 工程师



©版权归极客邦科技所有，未经许可不得转载

上一篇 15 | 二分查找（上）：如何用最省内存的方式实现快速查找功能？

下一篇 17 | 跳表：为什么Redis一定要用跳表来实现有序集合？

写留言

精选留言



Smallfly

32

有三种方法查找循环有序数组

一、

1. 找到分界下标，分成两个有序数组
2. 判断目标值在哪个有序数据范围内，做二分查找

二、

1. 找到最大值的下标 x ;
2. 所有元素下标 $+x$ 偏移，超过数组范围值的取模;
3. 利用偏移后的下标做二分查找;
4. 如果找到目标下标，再作 $-x$ 偏移，就是目标值实际下标。

两种情况最高时耗都在查找分界点上，所以时间复杂度是 $O(N)$ 。

复杂度有点高，能否优化呢？

三、

我们发现循环数组存在一个性质：以数组中间点为分区，会将数组分成一个有序数组和一个循环有序数组。

如果首元素小于 mid ，说明前半部分是有序的，后半部分是循环有序数组；
如果首元素大于 mid ，说明后半部分是有序的，前半部分是循环有序的数组；
如果目标元素在有序数组范围中，使用二分查找；
如果目标元素在循环有序数组中，设定数组边界后，使用以上方法继续查找。

时间复杂度为 $O(\log N)$ 。

2018-10-27



Victor

👍 4

今天的IP地址归属地问题，从工程实现的角度考虑，我更偏向于直接使用关系型数据库实现。

也就是将12w条归属地与IP区间的开始、结束存入数据库中。

数据库表ip_table有如下字段：area_name | start_ip | end_ip，start_ip及end_ip 均建立索引

SQL语句：

```
select area_name from ip_table where input_ip >= start_ip and input_ip <= end_ip;
```

学习算法的课程常常和自己工程开发的实际结合在一起，感觉两者是相互促进理解的过程。

2018-10-27

作者回复

数据库可以 单性能会受限

2018-10-28



charon

👍 4

用JavaScript实现的最基本的思考题：

array是传入的数组，value是要查找的值

思路是通过对比low,high的值来判断value所在的区间，不用多循环一遍找偏移量了~

```
function search(array,value){
  let low = 0;
  let high = array.length - 1;

  while(low <= high){
    let mid = low + ((high - low) >> 1);
    if(value == array[low]) return low;
    if(value == array[high]) return high;
    if(value == array[mid]) return mid;

    if(value > array[mid] && value > array[high] && array[mid] < array[low]){
      high = mid - 1;
    }else if(value < array[mid] && value < array[low] && array[mid] < array[low]){
      high = mid - 1;
    }else if(value < array[mid] && value > array[low]){
      high = mid - 1;
    }else{
      low = mid + 1;
    }
  }

  return -1
}
```

2018-10-26



zixuan

👍 3

思考题对应leetcode 33题，大家可以去练习

2018-10-31



komo0104

👍 3

给原来的index加上偏移量。

比如原来的二分查找代码从0开始到n-1结束，现在为x到x - 1 (即n-1+x-n)。

x为开始循环处的索引，例子里为3 (1所在索引)。需要扫描一遍数组找到x，复杂度O(n)。

其余和普通二分查找一样，需要多判断index not out of bound。如果索引超过n了要减n。

总的复杂度还是O(n)

2018-10-26



勤劳的小胖子-libo

👍 2

1. 先二分遍历找到分隔点index，特征是<pre, >=next;

2. 把数组分成二个部分，[0,index-1], [index,length-1];

3. 分别使用二分查找，找到给定的值。

时间复杂度为 $2 \times \log(n)$. 不确定有什么更好的办法。

2018-10-27



狼的诱惑

👍 1

@老师，请老师或其他高人回复指教

/**

* 例如： 4 5 6 1 2 3

* 循环数组的二分查找 总体时间复杂度 $O(n)$

*/

```
public static int forEqualsThan(int[] arr, int num) {
```

```
    if (arr[0] == num) {
```

```
        return 0;
```

```
    }
```

```
    int length = arr.length;
```

```
    int low = 0;
```

```
    int high = length - 1;
```

```
    //找到循环节点
```

```
    for (int i = 0; i < length; i++) {
```

```
        if (i == length - 1) {
```

```
            if (arr[i] > arr[0]) {
```

```
                low = i;
```

```
                high = 0;
```

```
                break;
```

```
            }
```

```
        } else {
```

```
            if (arr[i] > arr[i + 1]) {
```

```
                low = i;
```

```
                high = low + 1;
```

```
                break;
```

```
            }
```

```
        }
```

```
    }
```

```
    //判断第一个节点的大小位置，确定low和high的值，转变为正常有序的二分查找
```

```
    if (arr[0] < num) {
```

```
        high = low;
```

```
        low = 0;
```

```
    }
```

```
    if (arr[0] > num) {
```

```
        low = high;
```

```
        high = length - 1;
```

```
    }
```

```
    while (low <= high) {
```

```
int index = low + ((high - low) >> 1);
if (arr[index] > num) {
    high = index - 1;
}
if (arr[index] < num) {
    low = index + 1;
}
if (arr[index] == num) {
    return index;
}
return -1;
}
```

2018-10-31



Jeson

👍 1

#查找第一个值等于给定值的元素

```
def bsearchFirst(nums, val):
    low, high = 0, len(nums) - 1
    while low <= high:
        mid = low + ((high - low) >> 1)
        if nums[mid] >= val:
            high = mid - 1
        else:
            low = mid + 1
    if nums[low] == val:
        return low
    else:
        return None
```

#查找最后一个值等于给定值的元素

```
def bsearchLast(nums, val):
    low, high = 0, len(nums) - 1
    while low <= high:
        mid = low + ((high - low) >> 1)
        if nums[mid] <= val:
            low = mid + 1
        else:
            high = mid - 1
    if nums[high] == val:
        return high
    else:
        return None
```

#查找第一个大于等于给定值的元素

```
def bsearchFirstLargerEqual(nums, val):
    low, high = 0, len(nums) - 1
    while low <= high:
        mid = low + ((high - low) >> 1)
        if nums[mid] >= val:
            high = mid - 1
        else:
            low = mid + 1
    if nums[low] >= val:
        return low
    else:
        return None
```

#查找最后一个小于等于给定值的元素

```
def bSearchLastsmallerEqual(nums, val):
    low, high = 0, len(nums) - 1
    while low <= high:
        mid = low + ((high - low) >> 1)
        if nums[mid] <= val:
            low = mid + 1
        else:
            high = mid - 1
    if nums[high] <= val:
        return high
    else:
        return None
```

#循环数组的二分查找

```
def bSearchRecycle(nums, val):
    n, offset = len(nums), 3
    low, high = 0, n - 1
    while low <= high:
        mid = low + ((high - low) >> 1)
        midIdx = (mid + offset) % n
        if nums[midIdx] == val:
            return midIdx
        elif nums[midIdx] < val:
            low = mid + 1
        else:
            high = mid - 1
    return None
```

```
if __name__ == '__main__':
    nums = [0, 0, 2, 2, 3, 3]
    val = 1
    print(bSearchLastsmallerEqual(nums, val))
```

```
nums = [4, 5, 6, 1, 2, 3]
val = 3
print(bSearchRecycle(nums, val))
```

2018-10-28



锐雨

👍 1

关于循环有序数组的问题，假设array无重复元素的话，我们可以先二分法将array分成两个递增数组，再分别采用二分法。有不妥之处忘赐教哦，java：

```
public static int search(int x, int[] array) {
    if (null == array || array.length == 0) {
        return -1;
    }
    /** Step1. 先找到循环队列新一轮从小变大的起始位置*/
    int low = 1;
    int high = array.length - 1;
    int mid;
    int firstVal = array[0];
    /**到firstVal之后第一个<firstVal的位置*/
    int dividePos = -1;
    while(low <= high) {
        mid = low + ((high - low)>>1);
        if (array[mid] > firstVal) {
            low = mid + 1;
        } else {
            if (mid == 1 || array[mid - 1] > firstVal) {
                dividePos = mid;
                break;
            } else {
                high = mid - 1;
            }
        }
    }
    /** Step2. 对dividePos两边分别进行二分搜索 */
    int targetPos = -1;
    if (dividePos >= 1) {
        targetPos = bSearch(x, array, 0, dividePos - 1);
    }
}
```



```
if (-1 != targetPos) {
    return targetPos;
} else {
    targetPos = bSearch(x, array, dividePos, array.length - 1);
};
} else {
    targetPos = bSearch(x, array, 0, array.length - 1);
}
return targetPos;
}

/** 普通的二分搜索 */
public static int bSearch(int x, int[] array, int low, int high) {
    if (null == array || array.length == 0 || low < 0 || high >= array.length) {
        return -1;
    }
    int mid;
    while(low <= high) {
        mid = low + ((high - low) >> 1);
        if (array[mid] > x) {
            high = mid - 1;
        } else if (array[mid] < x) {
            low = mid + 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

2018-10-27



小喵喵

二分查找的变体问题，在java sdk、net framework中有实现吗？

2018-10-27

| 作者回复

c++有的

2018-10-28

👍 1



淤白

1. 通过二分法算出偏移量；
2. 通过偏移量使 [0, n-1] 和现在有序数组的下标关联起来；
3. 通过二分法算出结果；
4. 对结果进行偏移处理拿到最终位置。

👍 1

2018-10-26



蒋礼锐

1

留言有个地方写错了，不应该在 n/i 二分，应该是 $A(j-i)$ 到 $A(j)$ 。还请老师指点这样的思路是否正确

2018-10-26



tszzsk

1

第二段代码：if ((mid == 0) || (a[mid - 1] != value)) 可否改为 if ((mid == low) || (a[mid - 1] != value))？以及下面变体？

2018-10-26



He110

1

觉得在查找到值之后，使用 while(arr[mid-1] == value) mid--，这种可能好些，就是二分转遍历，如果数据量大而重复的数据量的个数不多的话，这种可能更有优势，如果是十个数据里面七八个需要查找的数据这种就肯定是二分了，但是这样的话，直接遍历可能也不慢

2018-10-26

作者回复

有大量重复数据时 就慢了

2018-10-26



Z.

0

老师，我有遇到一种情况，例如一个投票的功能，已知作品的id 和获得的票数，要获取该作品的排名！我的做法是先获取排名的列表（每个元素包含作品id 和票数），然后用二分法查找到和该作品票数相同的位置，然后比较作品id，如果没有命中，就先沿着当前向前比较，向前比较没有找到（比较到票数不相等的位置），就向后比较，直到找到！功能可以实现！就是想请教下，有没有更好的方法

2018-11-13



kakasi

0

二分查找应用场景：适用于近似查找，如第一个 \geq 给定值的元素，第一个 \leq 给定值的元素，都是二分查找的变体。而一般情况下查找某个给定值，二叉查找树和散列表更适合。

2018-11-08

qinggeouye

0

关于开篇问题，顺着解题思路：

问题也可以转变为“在有序数组中，查找第一个大于等于某个给定值的元素”。

当要查询某个 IP 归属地时，可以先通过二分查找，找到第一个结束 IP 大于等于这个 IP 的 IP 区间；

然后检查这个 IP 是否在这个区间内；如果在，就取出对应的归属地显示；如果不在，就返回未查找到。

2018-11-04





四、二分查找变形问题的代码实现

前提假设：以数据是从小到大排列为前提来实现

1.查找第一个值等于给定值的元素，比如在数组a[10]={1,3,4,5,6,8,8,8,11,18}

```
public int bSear1(int[] a, int val){
    int start = 0;
    int end = a.length - 1;
    int mid = -1;
    while(start <= end){
        mid = start + (end - start) / 2;
        if (a[mid] > val) {
            end = mid - 1;
        } else if (a[mid] < val) {
            start = mid + 1;
        } else {
            if(mid == 0 || a[mid-1] < val) break;
            else end = mid - 1;
        }
    }
    return mid;
}
```

2.查找最后一个值等于给定值的元素，比如在数组a[10]={1,3,4,5,6,8,8,8,11,18}

```
public int bSear2(int[] a, int val){
    int start = 0;
    int end = a.length - 1;
    int mid = -1;
    while(start <= end){
        mid = start + (end - start) / 2;
        if (a[mid] > val) {
            end = mid - 1;
        } else if (a[mid] < val) {
            start = mid + 1;
        } else {
            if(mid == 0 || a[mid-1] < val) break;
            else end = mid - 1;
        }
    }
    return mid;
}
```

3.查找第一个大于等于给定值的元素，比如在数组a[10]={1,3,4,5,6,8,8,8,11,18}

```
public int bSear3(int[] a, int val){
    int start = 0;
    int end = a.length - 1;
```

```
int mid = -1;
while(start <= end){
    mid = start + (end - start) / 2;
    if(a[mid] > val){
        if(start == end) break;
        end = mid - 1;
    } else if (a[mid] < val) {
        start = start + 1;
        if(mid == a.length-1) return -1;
    } else {
        if(mid == 0 || a[mid-1] < val)break;
        else end = mid - 1;
    }
}
return mid;
}
```

4.查找最后一个小于等于给定值的元素，比如在数组a[10]={1,3,4,5,6,8,8,8,11,18}

```
public int bSear4(int[] a, int val){
    int start = 0;
    int end = a.length - 1;
    int mid = -1;
    while(start <= end){
        mid = start + (end - start) / 2;
        if(a[mid] > val){
            end = mid - 1;
            if(mid == 0) return -1;
        } else if (a[mid] < val) {
            if(start == end) {
                break;
            }
            start = mid + 1;
        } else {
            if(mid == a.length-1 || a[mid+1] > val) break;
            else start = mid + 1;
        }
    }
    return mid;
}
```

2018-11-03



姜威

总结：二分查找（下）

一、四种常见的二分查找变形问题

👍 0

- 1.查找第一个值等于给定值的元素
- 2.查找最后一个值等于给定值的元素
- 3.查找第一个大于等于给定值的元素
- 4.查找最后一个小于等于给定值的元素

二、适用性分析

- 1.凡事能用二分查找解决的，绝大部分我们更倾向于用散列表或者二叉查找树，即便二分查找在内存上更节省，但是毕竟内存如此紧缺的情况并不多。
- 2.求“值等于给定值”的二分查找确实不怎么用到，二分查找更适合用在“近似”查找问题上。比如上面讲几种变体。

三、思考

- 1.如何快速定位出一个IP地址的归属地？

[202.102.133.0, 202.102.133.255] 山东东营市

[202.102.135.0, 202.102.136.255] 山东烟台

[202.102.156.34, 202.102.157.255] 山东青岛

[202.102.48.0, 202.102.48.255] 江苏宿迁

[202.102.49.15, 202.102.51.251] 江苏泰州

[202.102.56.0, 202.102.56.255] 江苏连云港

假设我们有 12 万条这样的 IP 区间与归属地的对应关系，如何快速定位出一个IP地址的归属地呢？

- 2.如果有一个有序循环数组，比如4, 5, 6, 1, 2, 3。针对这种情况，如何实现一个求“值等于给定值”的二分查找算法？

2018-11-03



Geek_dbf94a

0

课后思考题

可以先通过二分查找找出来循环数组的最小值和索引。通过最小值的索引把数组分为两个区间

然后每次查找的时候当前值比较 最小值和最大索引值的关系。

不在这个区间就去上一个区间去二分查找。

2018-10-30



君莫笑

0

课后思考，二分确定循环不重复数组定值

```
public int erfenCircleArray(int[] a, int v) {
    int low=0, high=a.length - 1;
    while(low <= high) {
        int mid = low + ((high - low)>>1);
        if(a[mid] > a[0]) {
            if(v > a[mid] || v < a[low]) {
                low = mid + 1;
            } else if(a[mid] == v) {
                return mid;
            } else {

```

```

high = mid - 1;
}
} else {
if(v < a[mid] || v > a[high]) {
high = mid - 1;
} else if(a[mid] == v) {
return mid;
} else {
low = mid + 1;
}
}
}
return -1;
}

```

2018-10-30



Monday

👍 0

二分的四种变种写法。个人觉得都是分三种情况进行讨论，再多注意判断边界值，三种情况为：

`a[mid] < value`

`a[mid] == value`

`a[mid] > value;`

思考题：自己思考了一小段时间没有好的思路，就各学友的留言，亲自实现了Smallfly的方法三，感觉比较好！谢谢！

2018-10-29



Infinite_gao

👍 0

关于变种的3和4，在3的基础上稍加改造，第一个`>=`给定值的前一个元素的位置就是最后一个`<`给定值的位置

2018-10-29



拉欧

👍 0

这道题之前做过，主要难点在于区分有序区间，其他和常规2分相同

2018-10-29



新手

👍 0

Rust实现的在循环有序无重复数组中查找值：

```

/// Binary search the element which equals to value in a sorted cycled slice
///
/// # Arguments
/// * `a` - A sorted cycled slice that hold unique values
/// * `v` - The value to be searched

```



```

///
/// # Return
/// Index of the value
///
/// # Examples
///
/// ```
/// extern crate binary_search;
/// use binary_search::binary_search_cycled;
///
/// let a = vec![5, 6, 7, 8, 9, 1, 2, 3, 4];
/// assert_eq!(binary_search_cycled(&a, 9), Some(4));
/// assert_eq!(binary_search_cycled(&a, 10), None);
/// ```
pub fn binary_search_cycled<T: PartialOrd>(a: &[T], v: T) -> Option<usize> {
    let mut low = 0i32;
    let mut high = a.len() as i32 - 1;

    while low <= high {
        // prevent overflow
        let mid = low + ((high - low) >> 1);
        let mid_val = &a[mid as usize];

        if *mid_val == v {
            return Some(mid as usize);
        } else if a[low as usize] <= *mid_val {
            // left side is ordered, right side is ordered or cycled
            if *mid_val > v && a[low as usize] <= v {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        } else {
            // left side is ordered or cycled, right side is ordered
            if *mid_val < v && a[high as usize] >= v {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }
}

```

None

}

2018-10-28



Jerry银银

0

我提一个无关算法问题的问题：

Donald E.Kunth有一个正式的中文名，叫：高德纳。顺便提一句：他的夫人也有一个中文名，叫：高精兰(Jill)

2018-10-27

作者回复

哈哈

2018-10-28



Jerry银银

0

对今天的题目有点疑问：

「有序数组是个循环有序数组」，给得例子是这样[4,5,6,1,2,3]，一定是前面环中的数总大于后面环的数吗？有可能是这样吗：[4,5,6,1,2,3,9,10,11,18,17,16,6,7,8]

2018-10-27

作者回复

你这个还是循环有序的吗

2018-10-28



liangjif

0

一开始，对于查找第一个等于查找值的下标这个问题的第二个简例，对于十一行代码，一直理解不了。想到某个情况是，mid是目标查找值多少钱，mid-1非目标查找值，mid-2是目标查找值，那11行代码是错误的啊。

后来马上想起，这是二分查找算法，上面假设是不成立的，因为二分查找面对的数据是有序的。

2018-10-27



舍得

0

第一段代码有漏洞，且不说int能不能表示数组的下标问题，毕竟这个数组能越界说明相当庞大了；

主要问题在于，如果我给定的数大于任何一个数组元素，low就会等于n，n是数组越界后的第一个元素，如果它刚好是要查找的值呢？

2018-10-27

作者回复

谢谢指正 我稍后改下

2018-10-29



YellowMax

0

思考题思路：两次二分查找

- 1.很容易判断循环数组是递增还是递减的（第一个值与第二个值比较或者第一个与最后一个值比较），这一步时间复杂度认为是 $O(1)$;
- 2.第一次二分（假设是递增的），那就找到第一个小于给定值（下标为0的数）的数，该操作不需要跟查找给定值一样刻意区分到底该往右还是往左（比下标为0大就往右，小就往左），记录下标，这一步时间复杂度是 $O(\log n)$;
- 3.第二次二分将第二步找到的下标记为 low，把该值加上数组长度减一作为 high，用唤醒缓冲区加取模的方式二分查找给定的值，这一步的时间复杂度是 $O(\log n)$;

合计 $O(1)+2O(\log n)$ ，用遍历的方式找循环点个人感觉不太可行，既然是遍历，为什么不直接遍历给定值呢？虽说循环点比整个数组长度小，但是这是概率性事件，我认为它的概率跟要找的值恰好在循环体后一半的概率差不多，所以先遍历循环点再二分不如直接遍历查找给定值。

2018-10-27



猫头鹰爱拿铁

0

刚动手把今天说的全写了一遍，变形真的好容易写错啊，特别是放在一快写。我感觉掌握了普通的二分针对变形的二分要点就是找first的就是要尽可能的降低高位指针，然后关注index有没有比0小，找last就是尽可能的增大低位指针，然后关注index有没有高位越界。

题目的解法是这样先遍历一遍数组，找到边界index，然后在0和index以及index+1到array.length-1这里分别进行二分查找。如果第一个找到了就不找第二个了。

`return (key=find(0,index,value))!=-1?find(index+1,array.length-1,value):key;`

2018-10-26



铁皮

0

我发现第一段代码(简洁版)好像有bug。

测试数组 `int[] nums = new int[] {1, 3, 3, 3, 3, 6, 7, 9, 12, 14, 18};`

执行 `bsearch(nums, nums.length, 20);`

出现 `java.lang.ArrayIndexOutOfBoundsException: 11`

但是执行 `bsearch(nums, nums.length - 1, 20);` 就没有问题。

但是第二段代码就是你提倡的写法，执行 `bsearch(nums, nums.length, 20);` 就没有问题。

但是执行 `bsearch(nums, nums.length - 1, 18);` 返回 "-1"。就不正确了。

可能还需要大量的测试

2018-10-26



charon

0

思考题：

我觉得不用找数组的偏移量，通过和low,heig,mid三个值的对比，也是可以确定要找的数值所在的区间的~

2018-10-26



Kudo

0

Python实现:

1. 查找第一个值等于给定值的元素

```
def bsearch(a, value):
    low, high = 0, len(a)-1
    while low <= high:
        mid = low + (high - low) // 2
        if a[mid] < value:
            low = mid + 1
        elif a[mid] > value:
            high = mid - 1
        elif a[mid] == value:
            if mid == 0 or a[mid-1] != value:
                return mid
            else:
                high = mid - 1
    return -1
```

2. 查找第一个值大于等于给定值的元素

```
def bsearch(a, value):
    low, high = 0, len(a)-1
    while low <= high:
        mid = low + (high - low) // 2
        if a[mid] < value:
            low = mid + 1
        if a[mid] >= value:
            if mid == 0 or a[mid-1] < value:
                return mid
            else:
                high = mid - 1
    return -1
```

2018-10-26



ALAN

0

1. 将循环数组划分成两个区间，第一个区间的数值都大于等于第二个区间，第一个区间的下标都小于第二个区间；首先比较数组中第一个位置的值a[0]和要查找的值val，若相等则直接return；若val大于a[0]，则val位于第一区间，若val小于a[0]，则val位于第二区间；

2. 确定第一和第二区间分界线位置x。可知分界线位置x满足a[x] > a[x+1] 条件；通过如下方法查找位置x

```
int low=0, high=n-1;
while(low <= high){
    int mid=low+(high-low)>>1;
    if(a[mid] > a[mid+1]) {
```

```

x=mid;
return x;
}
else if(a[mid]>a[low]){
low=mid+1;
}
else{
high=mid-1;
}
}

```

3.利用所确定的区间和x，在此区间内应用二分查找法计算"值等于给定值"。所确定区间为[0, x] 或[x+1,n-1]

2018-10-26



Sharry

0

思考题

```

int cycleArraySearch(int *arr, int len, int val) {
// 计算循环周期
int cycle = len - 1;
for (int i = 0; i < len - 1; i++) {
if (arr[i] > arr[i + 1]) {
cycle = i + 1;
break;
}
}
// 确定元素位于的区间
int low = 0, high = low + cycle - 1;
while (high <= len - 1) {
// 找到区间后, 进行二分查找
if (arr[low] <= val && arr[high] >= val) {
return binarySearch(arr, low, high, val);
}
low += cycle;
high += cycle;
}
return -1;
}

```

1. 计算循环周期
 2. 确定元素位于的区间
 3. 确定了区间后, 进行二分查找
- 时间复杂度为: $O(n)$

2018-10-26





您的好友William

0

不行不行，我之前想法有问题，既然已经遍历连接点了那么连接点之前的那段其实就已经查找过了，所以如果遍历到连接点没有找到就在后面用二分。这样做会保证算法复杂度比 $O(n)$ 小，但区别不大。

2018-10-26



传说中的成大大

0

关于思考题 竟然是循环数组 可以把他看作一个环，我们取出最小的那个作为环的起点和终点 可以通过起始坐标点求得中点坐标即可求出最小的那个点，用两个指针指向一个从中点往前走，一个指针从中点往后走，然后再根据要求值的大小选择挪动指针，即可得出一个区间
伪代码：

假设通过数组起点和终点坐标 p 向左的指针为 pl 向右的指针为 pr 要查找的值 val

如果 $pr > val \ \&\& \ val < pl$ 则 $pl -= 1 \ pr += 1$

else if $pr > val \ || \ pr < val$ 则说明不存了

所以整个算法复杂度为 $O(\log n)$

通过这种方法 把环分成两部分 满足条件部分和不满足条件部分

2018-10-26



您的好友William

0

思考题，是循环有序，那么我们先找到循环连接点的位置，遍历算法复杂度为 $O(n)$ 。找到位置后我们就可以设置 $low=0$ ，如果 $a[low]$ 小于要查找的数就在连接点前面找，如果 $a[low]$ 大于要查找的数就在连接点后面查找，复杂度是 $O(\log n)$ 。

2018-10-26



蒋礼锐

0

因为数组循环有序的特殊数组，所以首先要找到是多少为一循环，先假设每个循环之间是从小到大排列，每个循环内从大到小排列的数组为 A ，需要查找的数为 $value$

从零遍历， $step$ 为 1，如果 $A[i] < A[i+1]$ ，那么循环就是每 i 个一循环

在 n/i 中使用二分查找，中间点一定是为 i 的倍数，终止条件为 $A[j*i] > value$ ，则 $value$ 值一定在 j 到 $j+i$ 之间，因为还是有序，再用一次二分即可找到。

时间复杂度：

第一次遍历找循环，(不知道有没有可以优化的方法)，复杂度为 $O(N)$ ，假设循环为 c

第二次二分， $\log(n/c)$ ， n/c 是因为不会查找除了整数位以外的

第三次 $\log(c)$

所以整体的时间复杂度应该是 $o(N)$

空间复杂度为 1，没有开辟新数组

2018-10-26

