

讲堂 > 数据结构与算法之美 > 文章详情

## 15 | 二分查找（上）：如何用最省内存的方式实现快速查找功能？

2018-10-24 王争



### 15 | 二分查找（上）：如何用最省内存的方式实现快速查找功能？

朗读人：修阳 14'56" | 6.85M

今天我们讲一种针对有序数据集合的查找算法：二分查找（Binary Search）算法，也叫折半查找算法。二分查找的思想非常简单，很多非计算机专业的同学很容易就能理解，但是看似越简单的东西往往越难掌握好，想要灵活应用就更加困难。

老规矩，我们还是来看一道思考题。

假设我们有 1000 万个整数数据，每个数据占 8 个字节，**如何设计数据结构和算法，快速判断某个整数是否出现在这 1000 万数据中？**我们希望这个功能不要占用太多的内存空间，最多不要超过 100MB，你会怎么做呢？带着这个问题，让我们进入今天的内容吧！

### 无处不在的二分思想

二分查找是一种非常易懂的快速查找算法，生活中到处可见。比如说，我们现在来做一个猜字游戏。我随机写一个 0 到 99 之间的数字，然后你来猜我写的是什么。猜的过程中，你每猜一次，我就会告诉你猜的大了还是小了，直到猜中为止。你来想想，如何快速猜中我写的数字呢？

假设我写的数字是 23，你可以按照下面的步骤来试一试。（如果猜测范围的数字有偶数个，中间数有两个，就选择较小的那个。）

次数	猜测范围	中间数	对比大小
第1次	0-99	49	$49 > 23$
第2次	0-48	24	$24 > 23$
第3次	0-23	11	$11 < 23$
第4次	12-23	17	$17 < 23$
第5次	18-23	20	$20 < 23$
第6次	21-23	22	$22 < 23$
第7次	23		✓

7 次就猜出来了，是不是很快？这个例子用的就是二分思想，按照这个思想，即便我让你猜的是 0 到 999 的数字，最多也只要 10 次就能猜中。不信的话，你可以试一试。

这是一个生活中的例子，我们现在回到实际的开发场景中。假设有 1000 条订单数据，已经按照订单金额从小到大排序，每个订单金额都不同，并且最小单位是元。我们现在想知道是否存在金额等于 19 元的订单。如果存在，则返回订单数据，如果不存在则返回 null。

最简单的办法当然是从第一个订单开始，一个一个遍历这 1000 个订单，直到找到金额等于 19 元的订单为止。但这样查找会比较慢，最坏情况下，可能要遍历完这 1000 条记录才能找到。那用二分查找能不能更快速地解决呢？

为了方便讲解，我们假设只有 10 个订单，订单金额分别是：8，11，19，23，27，33，45，55，67，98。

还是利用二分思想，每次都与区间的中间数据比对大小，缩小查找区间的范围。为了更加直观，我画了一张查找过程的图。其中，low 和 high 表示待查找区间的下标，mid 表示待查找区间的中间元素下标。



看懂这两个例子，你现在对二分的思想应该掌握得妥妥的了。我这里稍微总结升华一下，二分查找针对的是一个有序的数据集合，查找思想有点类似分治思想。每次都通过跟区间的中间元素对比，将待查找的区间缩小为之前的一半，直到找到要查找的元素，或者区间被缩小为 0。

### $O(\log n)$ 惊人的查找速度

二分查找是一种非常高效的查找算法，高效到什么程度呢？我们来分析一下它的时间复杂度。

我们假设数据大小是  $n$ ，每次查找后数据都会缩小为原来的一半，也就是会除以 2。最坏情况下，直到查找区间被缩小为空，才停止。

被查找区间的大小变化：

$$n, \frac{n}{2}, \frac{n}{4}, \frac{n}{8}, \dots, \frac{n}{2^k} \dots$$

可以看出来，这是一个等比数列。其中  $n/2^k=1$  时， $k$  的值就是总共缩小的次数。而每一次缩小操作只涉及两个数据的大小比较，所以，经过了  $k$  次区间缩小操作，时间复杂度就是  $O(k)$ 。通

过  $n/2^k=1$ ，我们可以求得  $k=\log_2 n$ ，所以时间复杂度就是  $O(\log n)$ 。

二分查找是我们目前为止遇到的第一个时间复杂度为  $O(\log n)$  的算法。后面章节我们还会讲堆、二叉树的操作等等，它们的时间复杂度也是  $O(\log n)$ 。我这里就再深入地讲讲  $O(\log n)$  这种**对数时间复杂度**。这是一种极其高效的时间复杂度，有的时候甚至比时间复杂度是常量级  $O(1)$  的算法还要高效。为什么这么说呢？

因为  $\log n$  是一个非常“恐怖”的数量级，即便  $n$  非常非常大，对应的  $\log n$  也很小。比如  $n$  等于 2 的 32 次方，这个数很大了吧？大约是 42 亿。也就是说，如果我们在 42 亿个数据中用二分查找一个数据，最多需要比较 32 次。

我们前面讲过，用大  $O$  标记法表示时间复杂度的时候，会省略掉常数、系数和低阶。对于常量级时间复杂度的算法来说， $O(1)$  有可能表示的是一个非常大的常量值，比如  $O(1000)$ 、 $O(10000)$ 。所以，常量级时间复杂度的算法有时候可能还没有  $O(\log n)$  的算法执行效率高。

反过来，对数对应的就是指数。有一个非常著名的“阿基米德与国王下棋的故事”，你可以自行搜索一下，感受一下指数的“恐怖”。这也是为什么我们说，指数时间复杂度的算法在大规模数据面前是无效的。

## 二分查找的递归与非递归实现

实际上，简单的二分查找并不难写，注意我这里的“简单”二字。下一节，我们会讲到二分查找的变体问题，那才是真正烧脑的。今天，我们来看如何来写最简单的二分查找。

**最简单的情况就是有序数组中不存在重复元素**，我们在其中用二分查找值等于给定值的数据。我用 Java 代码实现了一个最简单的二分查找算法。

```
1 public int bsearch(int[] a, int n, int value) {
2     int low = 0;
3     int high = n - 1;
4
5     while (low <= high) {
6         int mid = (low + high) / 2;
7         if (a[mid] == value) {
8             return mid;
9         } else if (a[mid] < value) {
10            low = mid + 1;
11        } else {
12            high = mid - 1;
13        }
14    }
15
16    return -1;
17 }
```

[复制代码](#)

这个代码我稍微解释一下，low、high、mid 都是指数组下标，其中 low 和 high 表示当前查找的区间范围，初始 low=0，high=n-1。mid 表示 [low, high] 的中间位置。我们通过对比 a[mid] 与 value 的大小，来更新接下来要查找的区间范围，直到找到或者区间缩小为 0，就退出。如果你有一些编程基础，看懂这些应该不成问题。现在，我就着重强调一下容易出错的 3 个地方。

### 1. 循环退出条件

注意是 low<=high，而不是 low<high。

### 2.mid 的取值

实际上，mid=(low+high)/2 这种写法是有问题的。因为如果 low 和 high 比较大的话，两者之和就有可能溢出。改进的方法是将 mid 的计算方式写成 low+(high-low)/2。更进一步，如果要将性能优化到极致的话，我们可以将这里的除以 2 操作转化成位运算 low+((high-low)>>1)。因为相比除法运算来说，计算机处理位运算要快得多。

### 3.low 和 high 的更新

low=mid+1，high=mid-1。注意这里的 +1 和 -1，如果直接写成 low=mid 或者 high=mid，就可能会发生死循环。比如，当 high=3，low=3 时，如果 a[3] 不等于 value，就会导致一直循环不退出。

如果你留意我刚讲的这三点，我想一个简单的二分查找你已经可以实现了。实际上，二分查找除了用循环来实现，还可以用递归来实现，过程也非常简单。

我用 Java 语言实现了一下这个过程，正好你可以借此机会回顾一下写递归代码的技巧。

```
1 // 二分查找的递归实现
2 public int bsearch(int[] a, int n, int val) {
3     return bsearchInternally(a, 0, n - 1, val);
4 }
5
6 private int bsearchInternally(int[] a, int low, int high, int value) {
7     if (low > high) return -1;
8
9     int mid = low + ((high - low) >> 1);
10    if (a[mid] == value) {
11        return mid;
12    } else if (a[mid] < value) {
13        return bsearchInternally(a, mid+1, high, value);
14    } else {
15        return bsearchInternally(a, low, mid-1, value);
16    }
17 }
```

[复制代码](#)

## 二分查找应用场景的局限性

前面我们分析过，二分查找的时间复杂度是  $O(\log n)$ ，查找数据的效率非常高。不过，并不是什么情况下都可以用二分查找，它的应用场景是有很局限性的。那什么情况下适合用二分查找，什么情况下不适合呢？

**首先，二分查找依赖的是顺序表结构，简单点说就是数组。**

那二分查找能否依赖其他数据结构呢？比如链表。答案是不可以的，主要原因是二分查找算法需要按照下标随机访问元素。我们在数组和链表那两节讲过，数组按照下标随机访问数据的时间复杂度是  $O(1)$ ，而链表随机访问的时间复杂度是  $O(n)$ 。所以，如果数据使用链表存储，二分查找的时间复杂就会变得很高。

二分查找只能用在数据是通过顺序表来存储的数据结构上。如果你的数据是通过其他数据结构存储的，则无法应用二分查找。

**其次，二分查找针对的是有序数据。**

二分查找对这一点的要求比较苛刻，数据必须是有序的。如果数据没有序，我们需要先排序。前面章节里我们讲到，排序的时间复杂度最低是  $O(n \log n)$ 。所以，如果我们针对的是一组静态的数据，没有频繁地插入、删除，我们可以进行一次排序，多次二分查找。这样排序的成本可被均摊，二分查找的边际成本就会比较低。

但是，如果我们的数据集合有频繁的插入和删除操作，要想用二分查找，要么每次插入、删除操作之后保证数据仍然有序，要么在每次二分查找之前都先进行排序。针对这种动态数据集合，无论哪种方法，维护有序的成本都是很高的。

所以，二分查找只能用在插入、删除操作不频繁，一次排序多次查找的场景中。针对动态变化的数据集合，二分查找将不再适用。那针对动态数据集合，如何在其中快速查找某个数据呢？别急，等到二叉树那一节我会详细讲。

**再次，数据量太小不适合二分查找。**

如果要处理的数据量很小，完全没有必要用二分查找，顺序遍历就足够了。比如我们在一个大小为 10 的数组中查找一个元素，不管用二分查找还是顺序遍历，查找速度都差不多。只有数据量比较大的时候，二分查找的优势才会比较明显。

不过，这里有一个例外。如果数据之间的比较操作非常耗时，不管数据量大小，我都推荐使用二分查找。比如，数组中存储的都是长度超过 300 的字符串，如此长的两个字符串之间比大小，就会非常耗时。我们需要尽可能地减少比较次数，而比较次数的减少会大大提高性能，这个时候二分查找就比顺序遍历更有优势。

**最后，数据量太大也不适合二分查找。**

二分查找的底层需要依赖数组这种数据结构，而数组为了支持随机访问的特性，要求内存空间连续，对内存的要求比较苛刻。比如，我们有 1GB 大小的数据，如果希望用数组来存储，那就需要 1GB 的连续内存空间。

注意这里的“连续”二字，也就是说，即便有 2GB 的内存空间剩余，但是如果这剩余的 2GB 内存空间都是零散的，没有连续的 1GB 大小的内存空间，那照样无法申请一个 1GB 大小的数组。而我们的二分查找是作用在数组这种数据结构之上的，所以太大的数据用数组存储就比较吃力了，也就不能用二分查找了。

## 解答开篇

二分查找的理论知识你应该已经掌握了。我们来看下开篇的思考题：如何在 1000 万个整数中快速查找某个整数？

这个问题并不难。我们的内存限制是 100MB，每个数据大小是 8 字节，最简单的办法就是将数据存储到数组中，内存占用差不多是 80MB，符合内存的限制。借助今天讲的内容，我们可以先对这 1000 万数据从小到大排序，然后再利用二分查找算法，就可以快速地查找想要的數據了。

看起来这个问题并不难，很轻松就能解决。实际上，它暗藏了“玄机”。如果你对数据结构和算法有一定了解，知道散列表、二叉树这些支持快速查找的动态数据结构。你可能会觉得，用散列表和二叉树也可以解决这个问题。实际上是不行的。

虽然大部分情况下，用二分查找可以解决的问题，用散列表、二叉树都可以解决。但是，我们后面会讲，不管是散列表还是二叉树，都会需要比较多的额外的内存空间。如果用散列表或者二叉树来存储这 1000 万的数据，用 100MB 的内存肯定是存不下的。而二分查找底层依赖的是数组，除了数据本身之外，不需要额外存储其他信息，是最省内存空间的存储方式，所以刚好能在限定的内存大小下解决这个问题。

## 内容小结

今天我们学习了一种针对有序数据的高效查找算法，二分查找，它的时间复杂度是  $O(\log n)$ 。

二分查找的核心思想理解起来非常简单，有点类似分治思想。即每次都通过跟区间中的中间元素对比，将待查找的区间缩小为一半，直到找到要查找的元素，或者区间被缩小为 0。但是二分查找的代码实现比较容易写错。你需要着重掌握它的三个容易出错的地方：循环退出条件、mid 的取值，low 和 high 的更新。

二分查找虽然性能比较优秀，但应用场景也比较有限。底层必须依赖数组，并且还要求数据是有序的。对于较小规模的数据查找，我们直接使用顺序遍历就可以了，二分查找的优势并不明显。二分查找更适合处理静态数据，也就是没有频繁的数据插入、删除操作。

## 课后思考



1. 如何编程实现“求一个数的平方根”？要求精确到小数点后 6 位。
2. 我刚才说了，如果数据使用链表存储，二分查找的时间复杂就会变得很高，那查找的时间复杂度究竟是多少呢？如果你自己推导一下，你就会深刻地认识到，为何我们会选择用数组而不是链表来实现二分查找了。

欢迎留言和我分享，我会第一时间给你反馈。



版权归极客邦科技所有，未经许可不得转载

写留言

#### 精选留言



蒋礼锐

👍 2

因为要精确到后六位，可以先用二分查找出整数位，然后再二分查找小数第一位，第二位，到第六位。

整数查找很简单，判断当前数小于+1后大于即可找到，

小数查找从查找小数后第一位来说，从 $x.0$ 到 $(x+1).0$ ，查找终止条件与整数一样，当前数小于，加 $0.1$ 大于，

后面的位数以此类推，可以用 $x \cdot 10^{(-i)}$ 通项来循环或者递归，终止条件是 $i > 6$ ，

想了一下复杂度，每次二分是 $\log n$ ，包括整数位会查找7次，所以时间复杂度为 $7 \log n$ 。空间复杂度没有开辟新的储存空间，空间复杂度为1。

没有具体用代码实现，只是思路，还请多多指正。之后会用js去实际实现。



2018-10-24



TWO STRINGS

👍 2

1000w数据查找这个，在排序的时候不就可以找到了么？

2018-10-24



毛栗子

👍 2

二分法一直在用，知道太小的、非数组、非有序确实不适合用，不过确实没有注意到太大的局限性！get√了~

2018-10-24



Smallfly

👍 1

1. 求平方根可以用二分查找或牛顿迭代法；
2. 有序链表的二分查找时间复杂度为  $O(n)$ 。

2018-10-24



五岳寻仙

👍 1

课后思考题：可以用二分法计算一个数的平方根。对于整数部分和6位小数位分别使用二分查找。

2018-10-24



峰

👍 1

1.最主要的问题是怎么精确第6位停止吧，我觉得首先折半的时候，只计算到第6位，然后和上一次比较，如果相等就可以结束了。当然鉴于浮点数计算不精确，先把原数乘1000000，但这样很容易就溢出了吧，还是用高精度的实现吧2.如果用单链表实现，先遍历一遍知道长度，每次折半都需要定位当前部分链表的中间元素，查询次数不变，总得算起来 $n+n/2+n/4+\dots+1$ ，所以是 $O(n)$ 的复杂度。

2018-10-24



蝴蝶

👍 1

算了下,链表如果使用二分查找法的话,应该是 $O(n)$ 的时间复杂度.

2018-10-24



liangjf

👍 0

二分查找使用特点:

- 1.数据规模够大，且不能过大。因为使用数组连续内存的方式存储数据。会造成需要的连续内存过大，可能会造成申请内存不足。
- 2.待查找数据已排序。因为是通过类似二分法那样折半对比，不是有序会出错。
- 3.退出条件是  $low \leq high$ ，求mid是  $low + (high - low) / 2$ ，防止 $(low + high) / 2$ 溢出。

2018-10-24



Harry

👍 0

二分查找 $mid = (low + high) / 2$

由于 $low + high$ 可能越界，建议优化

```
>> mid = low+(high-low)/2  
>> mid = low+((high-low)>>1)
```

2018-10-24



锐雨

👍 0

求平方根，可以参考0到99之间猜数字的思路，99换成x，循环到误差允许内即可，注意1这个分界线。欢迎交流，Java如下

```
public static double sqrt(double x, double precision) {  
    if (x < 0) {  
        return Double.NaN;  
    }  
    double low = 0;  
    double up = x;  
    if (x < 1 && x > 0) {  
        /** 小于1的时候*/  
        low = x;  
        up = 1;  
    }  
    double mid = low + (up - low)/2;  
    while(up - low > precision) {  
        if (mid * mid > x ) { //TODO mid可能会溢出  
            up = mid;  
        } else if (mid * mid < x) {  
            low = mid;  
        } else {  
            return mid;  
        }  
        mid = low + (up - low)/2;  
    }  
    return mid;  
}
```

2018-10-24



你好可爱哟

👍 0

王老师，考研的话可以以这个课程作为数据结构第一轮的基础复习吗。如果可以，还需要补充其他概念知识吗

2018-10-24



夜芽徽

👍 0

有序链表时间复杂度我是这么算的，等于查找的次数+每次查找需要遍历的链表长度/2  
即 $k+n/2+n/4+...+n/2^k$  ( $k=\log n$ ，为查找的次数) 推算后为 $\log n+n-1$ 所以时间复杂度为 $O(n)$

2018-10-24



虎虎❤️

👍 0

用  $(a+b)$  的平方展开公式把数字分为前4位和后3位，然后再分别用2分查找。

2018-10-24



三忌

👍 0

```
def sqrt(x):
```

```
    ...
```

```
    求平方根，精确到小数点后6位
```

```
    ...
```

```
    low = 0
```

```
    mid = x / 2
```

```
    high = x
```

```
    while abs(mid ** 2 - x) > 0.000001:
```

```
        if mid ** 2 < x:
```

```
            low = mid
```

```
        else:
```

```
            high = mid
```

```
            mid = (low + high) / 2
```

```
    return mid
```

2018-10-24



三忌

👍 0

```
def bin_search(arrs, val):
```

```
    ...
```

```
    二分查找
```

```
    ...
```

```
    return bin_serach_inner(arrs, 0, len(arrs) - 1, val)
```

```
def bin_serach_inner(arrs, low, high, value):
```

```
    if low > high: return -1
```

```
    mid = low + ((high - low) >> 1)
```

```
    if arrs[mid] == value:
```

```
        return mid
```

```
    elif arrs[mid] < value:
```

```
        return bin_serach_inner(arrs, mid + 1, high, value)
```

```
    else:
```

```
        return bin_serach_inner(arrs, low, mid - 1, value)
```

2018-10-24



冯剑

👍 0

```
public class Main {
```

```
public static void main(String[] args) {  
  
    int value = 120;  
    float low = 1f;  
    float high = (1.000000f * value) / 2;  
    System.out.println(getSquaerRoot(value, low, high));  
}
```

//如何编程实现“求一个数的平方根”？要求精确到小数点后 6 位

```
public static float getSquaerRoot(int value, float low, float high) {  
    float mid = low + ((high - low) / 2);  
    if (Math.abs(value - mid * mid) < 0.00001F) {  
        return mid;  
    } else if (mid * mid < value) {  
        return getSquaerRoot(value, mid, high);  
    } else {  
        return getSquaerRoot(value, low, mid);  
    }  
}
```

2018-10-24



猫头鹰爱拿铁

👍 0

第二题 $n\log n$ ，需要 $\log n$ 次二分查找但每次遍历low和high的时候需要 $n$ 次，所以是 $n\log n$

2018-10-24



一念逍遥、

👍 0

2.时间复杂度是 $O((1/2+1/4+1/8+\dots+1/2^n)N)=O(N)$ ，主要操作浪费在移动指针上，比较中点元素大小倒无所谓了。

2018-10-24



yaya

👍 0

1.不考虑浮点数计算复杂度的话，在0到  $n/2$  的区间上用二分

2.时间复杂度为  $o\log n$ ，虽然顺序访问的复杂度是 $o(n)$ ，但是没有二分必须比顺序小的规则啊

2018-10-24



王小李

👍 0

平方根可以用牛顿迭代实现。

2018-10-24



蒋礼锐

👍 0

想了一下，空间复杂度不为1，因为需要开辟遍历数组， $n$ 小于等于6位时，最大的应该是小数后六位那个连续数组， $n$ 大于6位时最大遍历的为 $n$

2018-10-24



Smallfly

👍 0

有序链表逐个查找复杂度也就  $O(n)$ ，二分查找怎么可能算出来  $n\log(n)$ .....

2018-10-24



有朋自远方来

👍 0

需要面试

准备从头看到尾

那么问题来了，算法题选哪家？

2018-10-24



单人孤剑

👍 0

求平方根，将所有可能的值增量0.000001存数组里，用2分法查找。比较条件修改为该数的2次方==需要开平方的数。

2018-10-24



拉欧

👍 0

1.才用二分法，在选定上下界时需要考虑这个数大于0和小于0两种情况。

2. $n\log n$

2018-10-24



彬

👍 0

1000条订单记录，无重复的从小到大排列，找到19的订单最多需要19次吧？

2018-10-24