

二分查找（上）：如何用最省内存的方式实现快速查找功能？

二分查找算法（Binary Search）算法，也叫折半查找算法。

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，快速判断某个数是否出现在这1000万数据中？功能不要占用太多空间，最多不超过100MB。

二分思想

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；利用二分思想每次与中间数对比大小，如图：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。



二分查找针对的是一个有序的数据集合，查找思想有点类似于分治思想。每次都通过跟区间的中间元素对比，将待查找缩小为之前的一半，知道找到要查找的元素，或者区间被说下为0。

$O(\log n)$ 惊人的查找速度

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，最坏到空才停止。

被查找区间的大小变化：

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

其中 $n/2^k=1$ 时， k 为总共缩小的次数， 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 $k=\log n$ 所以时间复杂度就是 $O(\log n)$ 。

堆、二叉树的操作等等，它们的时间复杂度也是 $O(\log n)$ 。 $O(\log n)$ 这种对数时间复杂度。这是一种极其高效的时间复杂度。

二分查找的递归与非递归实现

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。

```
public int bsearch(int[] a,int value){
    int low = 0;
    int high = a.length - 1;

    while(low <= high){
        int mid = low + (high - low) / 2;
        if(a[mid] == value){
            return mid;
        }else if(a[mid] < value){
            low = mid + 1;
        }else {
            high = mid - 1;
        }
    }
    return -1;
}
```

3个易错的地方

1、循环退出条件

注意是 $low \leq high$ 而不是 $low < high$;

2、mid的取值

$mid = (low + high) / 2$ 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 $low + (high - low) / 2$ 。可以将这里的除以 2 操作转化成位运算 $low + ((high - low) >> 1)$ 。

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。

递归实现

```
public int bsearch1(int[] a,int value){
    return bsearchInternally(a,0,a.length-1,value);
}
private int bsearchInternally(int[] a,int low,int high,int value){
    if(low > high) return -1;
    int mid = low + ((high - low) >> 1);

    if(a[mid] == value) {
        return mid;
    }else if(a[mid] < value){
        return bsearchInternally(a,mid+1,high,value);
    }else {
        return bsearchInternally(a,low,mid-1,value);
    }
}
```

二分查找应用场景的局限性

首先二分查找依赖的是顺序表结构，简单说就是数组 不支持链表，主要原因是二分查找算法需要按照下标随机访问元素。

其次，二分查找针对的是有序数据 二分查找中数据必须是有序的，如果数据没有序，我们需要先排一下序，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。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。

再次，数据量太小不适合二分查找 数据量很小完全没必要二分查找，顺序遍历足够了。

最后，数据量太大也不适合二分查找 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

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于数组中，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数了。

内容小结

二分查找，它的时间复杂度是 $O(\log n)$ 。

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。

底层必须依赖数组，并且还要求数据是有序的。

二分查找更适合处理静态数据。

课后思考

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

```
/**
 * 求平方根
 * @param a
 * @return
 */
public float sqrt(float a){
    if(a == 1.0f){
        return a;
    }
    float low = 0;
    float high = a;
    float mid = (low + high) / 2;

    while(mid * mid - a > 1e-6 || a - mid * mid > 1e-6){
        if(mid * mid > a){
            high = mid;
            mid = (low + high) / 2;
        }else{
            low = mid;
            mid = (low + high) / 2;
        }
    }
    return mid;
}
```

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。

假设链表长度为 n ，二分查找每次都要找到中间点(计算中忽略奇偶数差异)：第一次查找中间点，需要移动指针 $n/2$ 次；第二次，需要移动指针 $n/4$ 次；第三次需要移动指针 $n/8$ 次；..... 以此类推，一直到 1 次为值

总共指针移动次数(查找次数) = $n/2 + n/4 + n/8 + \dots + 1$ ，这显然是个等比数列，根据等比数列求和公式： $\text{Sum} = n - 1$ 。

最后算法时间复杂度是： $O(n-1)$ ，忽略常数，记为 $O(n)$ ，时间复杂度和顺序查找时间复杂度相同

但是稍微思考下，在二分查找的时候，由于要进行多余的运算，严格来说，会比顺序查找时间慢