<https://imageslr.com/2020/03/15/binary-search.html>

# 引言

二分查找的题目类型有：

* 查找特定值
* 查找第一个大于等于特定值的元素
* 查找最后一个小于等于特定值的元素
* …

二分查找说简单也简单，说难也难。说简单是因为，它无非就是一个循环里嵌套了两三个 if/else。说难是因为，它有很多细节，而且每个细节都不能出错：

* left、right 要初始化为 0、n-1 还是 0、n？
* 循环的判定条件是 left < right 还是 left <= right？
* if 的判定条件应该怎么写？if 的判定条件为真时，应当更新 left 还是 right？
* 更新 left、right 时，mid 要不要 ±1？
* …

可以看到，二分查找不仅有很多类型，还有很多细节。以前每次做二分查找问题的时候，我都会重新推导一遍代码，但是由于细节很多，难免出错。**有没有一个通用的模板，能够一劳永逸地解决所有二分查找问题呢？**

本文首先从「找下界」入手，引出通用的二分查找模板；然后在不同类型的二分查找中套用这个模板，验证其适用性；最后对比了「闭区间」和「左闭右开」两种写法，说明了这两种写法其实是同一种思路。

本文希望通过最自然、最容易理解的方式来描述思路。理解了本文的内容后，我们可以**直接「写」出模板，而不需要「背」会模板**，且无论哪种写法都能信手拈来。

**1.找下界(Find Lower Boundary)**

给定一个升序排列的数组，我们将满足 x ≥ target 的**第一个元素**定义为「**下界**」。给定一个目标值 target，要求返回其下界的下标。如果下界不存在，返回数组长度。

对于数组 [1,2,3,5,5,5,6,7,9]，令 target=5，则满足 x ≥ target 的下界的下标应该是 3，如下图所示： A picture containing diagram

Description automatically generated

可以看到，从这个位置将数组分为左右两部分，**左侧的元素都「小于」target，右侧的元素都「大于等于」target**： Chart, box and whisker chart

Description automatically generated

接下来，我们使用「闭区间」的写法来描述思路。先定义几个变量：

* 区间范围为 [left,right]，left、right 是区间的左右边界的下标
* mid 是 [left,right] 的中间位置
* 初始时，left、right 分别指向数组的第一个和最后一个元素
* **当 left > right 时，表示区间为空**

如果我们在二分查找的过程中，**不断右移 left，左移 right，使得所有「小于」target 的元素都在 left 左侧，所有「大于等于」target 的元素都在 right 右侧，那么当区间为空时，left 就是要查找的下界**：

Diagram, timeline

Description automatically generated

根据上述思路，算法步骤如下：

* 若 nums[mid] >= target，说明 [mid,right] 区间的所有元素均「大于等于」target，因此 right 左移，有 right = mid-1
* 否则，说明 [left,mid] 区间的所有元素均「小于」target，因此 left 右移，有 left = mid+1
* 重复上述步骤，直到区间为空，表示找到了下界，**返回 left**。**因此循环条件为 left <= right，表示“区间不为空”**
* 注意，上述两个赋值语句均跳过了中间元素 mid

上面示例的查找过程如下： Diagram

Description automatically generated with low confidence

### 找下界模板代码

function lowerBound(nums, target) {

    let left = 0;

    let right = nums.length - 1;

// 查找满足 x >= target 的下界的下标

    while (left <= right) {

        const mid = Math.floor(left + (right - left) / 2);

        // 这里的比较运算符与题目要求一致

        if (nums[mid] >= target) {

            right = mid - 1;

        } else {

            left = mid + 1;

        }

    }

// 这里可以根据需要增加下界不存在的结束表达

// e.g

// if (left == nums.length || nums[lo] < target) {

    //    return -1;

    // }

    return left; // 返回下界的下标

}

当区间为空时，left 指向第一个「大于等于」target 的元素，因此要**返回 left**。若下界不存在，有 left == n, 增加一个 if(left >= n) {return -1} 可以满足下界不存在的结束表达。「下界」实际上就是按顺序插入 target 的位置。

上面的代码中，**if 的判定条件和给定的比较规则是一致的**：要找满足 x >= target 的第一个元素，所以是 if nums[m] >= target。**如果要找满足 x > target 的第一个元素，那么只需改为 if nums[m] > target。if 为真时更新 right**。

无论是找下界、还是找上界、还是找特定值，都可以套用这个模板代码。

**2.找上界(Find Upper Boundary)**

定义满足 x ≤ target 的**最后一个元素**为「**上界**」。给定一个 target，要求返回升序数组中上界的下标。

根据上界和下界的定义，我们可以发现：**上界和「互补的complementary」下界是相邻的，并且 上界 = 下界 - 1**。比如 x ≤ target 的上界和 x > target 的下界相邻。因此，**所有找上界的问题，都可以转换为「互补的」找下界的问题。**

对于本题而言，要找 x ≤ target 的上界，首先套用上文的模板代码，实现找 x > target 的下界的函数**(注意不是x > =target)**：

### 找上界模板代码

function upperBound(nums, target) {

    let left = 0;

    let right = nums.length - 1;

    // 先查找满足 x > target 的下界的下标

    while (left <= right) {

        const mid = Math.floor(left + (right - left) / 2);

// 只需将寻找下界模板中的比较运算符 >= 改为 > 即可

        if (nums[mid] > target) {

            right = mid - 1;

        } else {

            left = mid + 1;

        }

    }

    // 将下界的下标减一，就是我们要找的上界

    return left - 1; // 或者 right

}

**3.查找指定值第一次出现的位置(Find Target First Occurrence)**

查找满足 x == target 的第一个元素，如果不存在，返回 -1。

只需要先查找满足 x >= target 的下界，然后再判断下界与 target 是否相等。

### 找指定值第一次出现的位置模板代码

function searchFirst(nums, target) {

    let left = 0;

    let right = nums.length - 1;

    // 查找满足 x >= target 的下界的下标

    while (left <= right) {

        const mid = Math.floor(left + (right - left) / 2);

        // 这里的比较运算符与题目要求一致

        if (nums[mid] >= target) {

            right = mid - 1;

        } else {

            left = mid + 1;

        }

    }

    // **判断一下是否越界**，**下界与 target 是否相等**

    if (left >= nums.length || nums[left] !== target) {

        return -1;

    }

    return left;

}

**4.查找指定值最后一次出现的位置(Find Target Last Occurrence)**

查找满足 x == target 的最后一个元素，如果不存在，返回 -1。

只需要先查找满足 x <= target 的上界，然后再判断上界与 target 是否相等。上文中已经描述了如何将查找上界转化为查找下界

### 找指定值最后一次出现的位置模板代码

function searchLast(nums, target) {

    let left = 0;

    let right = nums.length - 1;

    // 先查找满足 x > target 的下界的下标

    while (left <= right) {

        const mid = Math.floor(left + (right - left) / 2);

// 只需将寻找下界模板中的比较运算符 >= 改为 > 即可

        if (nums[mid] > target) {

            right = mid - 1;

        } else {

            left = mid + 1;

        }

    }

    // 判断一下是否越界，上界与 target 是否相等

    if (right < 0 || nums[right] !== target) {

        return -1;

    }

    return right; // 这里返回 right 而不是 left

}

**5.查找指定值的位置(Find Target Occurrence)**

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

之所以把这道题放在最后面说，是因为这道题**完完全全就是找下界的题目**！模板代码一行都不需要改：

### 找指定值的位置模板代码

function lowerBound(nums, target) {

    let left = 0;

    let right = nums.length - 1;

// 查找满足 x >= target 的下界的下标

    while (left <= right) {

        const mid = Math.floor(left + (right - left) / 2);

        // 这里的比较运算符与题目要求一致

        if (nums[mid] >= target) {

            right = mid - 1;

        } else {

            left = mid + 1;

        }

    }

    return left; // 返回下界的下标

}

target 按顺序插入的位置，满足 x ≥ target 的第一个元素的位置。由于可以返回**任意一个等于**目标值的位置，所以这里还可以增加一个判断，当 nums[mid] == target 时直接返回。代码如下。

function lowerBound(nums, target) {

    let left = 0;

    let right = nums.length - 1;

// 查找满足 x >= target 的下界的下标

    while (left <= right) {

        const mid = Math.floor(left + (right - left) / 2);

        // 这里的比较运算符与题目要求一致

        if (nums[mid] == target) {

            return mid;

        } else if (nums[mid] > target) {

            right = mid - 1;

        } else {

left = mid + 1;

}

    }

// 可以加入如果找不到target就返回-1的判断

// e.g

// if(left == nums.length || nums[left] != target) {

// return -1;

// }

    return left; // 返回下界的下标

}

**总结：模板代码**

二分查找无论是找下界、还是找上界、还是找特定值，都可以套用「找下界」的模板代码：

* 循环条件为 left <= right，表示闭区间不为空
* **if 的判定条件和给定的比较规则是一致的**：比如要找满足 x >= target 的第一个元素，就令 if nums[m] >= target；要找满足 x > target 的第一个元素，就令 if nums[m] > target
* if 为真时，更新 right：right = mid - 1；否则 left = mid + 1
* 当循环结束时，left 就指向下界，right 指向「互补条件」的上界

当题目要查找「最小值」、「第一个」时，就说明要查找「下界」，此时就可以使用这个模板。

上界和下界是相邻的，因此找上界可以转换为「互补的」找下界的问题，从而套用本文的模板

Interestingly, binary search works beyond sorted arrays. You can use binary search **whenever you can make a binary decision to shrink the search range**.

注意是否越界！